М. М. Глибовець,
О. В. Олецький

Штучний інтелект

З метою створення сучасних україномовних підручників та навчальних посібників для вищих навчальних закладів Вчена Рада Національного університету "Києво-Могилянська академія" оголосила конкурс із написання україномовних підручників.

На конкурс приймаються рукописи, підготовлені науковцями НаУКМА самостійно або ж у співавторстві з працівниками інших навчальних закладів України.

Для фінансової підтримки цього проекту запрошуються ґрантодавці, які за власним бажанням можуть обрати напрям спеціалізації кращих наукових робіт.

За детальнішою інформацією просимо звертатися до відділу перспективного розвитку НаУКМА (04070, Київ-70, вул. Сковороди, 2).

Тел.: (+38044) 463-71-10

Факс: (+38044) 417-82-15.

М. М. Глибовець,
О. В. Олецький

Штучний інтелект

Затверджено Міністерством освіти і науки України як підручник для студентів вищих навчальних закладів, що навчаються за спеціальностями "Комп'ютерні науки" та "Прикладна математика"

Київ
Видавничий дім
"КМ Академія"
2002

ББК 32.813я73
Г54

Висловлюємо вдячність
Фундації родини Фещенко-Чопівських
за підтримку проведення в Національному
університеті "Києво-Могилянська академія"
конкурсу з написання україномовних підручників,
у рамках якого здійснюється це видання.

Підручник призначений для студентів вищих навчальних закладів, які навчаються за спеціальностями "Комп'ютерні науки" та "Прикладна математика". Метою підручника є ознайомлення студентів з основами теорії штучного інтелекту. Розглядаються основні методи роботи зі знаннями, зокрема з нечіткими та недостовірними; планування цілеспрямованих дій та прийняття рішень; ігрові задачі; розпізнавання образів та нейронні мережі.

Гриф надано
Міністерством освіти і науки України
від 13 листопада 2001 р. №1/11-4303

Рекомендовано до друку рішенням
Вченої ради НаУКМА
Протокол № 2 (10.9) від 15 лютого 2001 р.

Рецензенти:

Анісімов А. В. — завідувач кафедри математичної інформатики факультету кібернетики Київського національного університету ім. Тараса Шевченка, професор, доктор фізико-математичних наук

Кривий С. Л. — доктор фізико-математичних наук, старший науковий співробітник Інституту кібернетики НАНУ

ISBN 966-518-153-X

ВІД АВТОРІВ

Сучасний рівень і темпи розвитку природознавства висувають перед студентом університету, випускником, молодим та й кваліфікованим вже спеціалістом основну вимогу — не тільки вміти вчитися, вільно оперувати знаннями і науковим потенціалом в обраній галузі діяльності, а й ефективно застосовувати нові знання, наукові відкриття, технології. Особливо це стосується такого розділу науки, як штучний інтелект або інтелектуальні системи обробки інформації. Динамізм розвитку сучасних інформаційних технологій вимагає інтеграції суміжних дисциплін на загальній для них фундаментальній основі. Інтелектуальні системи нового типу повинні створювати той інструментарій, який забезпечить освічену людину засобами якіснішого і глибшого опанування новітніми досягненнями інформатики й природознавства.

З іншого боку, тепер важко назвати розділ теоретичної науки, який би не знайшов застосування в системах штучного інтелекту. Останні досягнення в сфері розпізнання образів, робототехніки, експертних систем свідчать про перехід з площини теоретичних досліджень у площину практичного застосування.

Навчальні плани багатьох спеціалізацій університетів України містять нормативні спеціальні курси, що мають назву "Системи штучного інтелекту", "Інтелектуальні системи" або щось подібне. Але жодного україномовного підручника на відповідну тему ми не знаємо. Тому ще одним стимулом написання цього підручника було прагнення запровадження і ширшого застосування україномовної термінології у відповідній галузі. Враховуючи великий обсяг матеріалу розгляду, ми змушені були зупинитись на такій концепції методології опису матеріалу: постановка завдання, короткий екскурс в історію розв'язання, стислий розгляд теоретичних досягнень, алгоритм(и) розв'язання, задачі та вправи на застосування цих алгоритмів. При огляді теоретичних матеріалів наводяться, в переважній більшості випадків, тільки основні результати (теореми, твердження) без доведення. Для ознайомлення з доведеннями даються посилання на відповідну літературу.

Вагомий внесок у розвиток багатьох розділів штучного інтелекту був зроблений київською школою науковців, серед яких назвемо Глушкова В. М., Михалевича В. С., Сергієнка І. В., Летічевського О. А., Капітонову Ю. В., Ющенко К. Л., Андона Ф. І., Анісімова А. В., Цейтліна Г. Є., Гороховського С. С., Проценка В. С., Довгялло О. М., Вінцюка Т. К., Бублика Б. М., Кириченка М. Ф. Усі вони тією чи іншою мірою вплинули на формування нашого наукового світогляду, вироблення концепції написання цього підручника. Доля подарувала можливість безпосередньо вчитися в багатьох із них.

Автори протягом багатьох років читали лекції і проводили практичні заняття з нормативного курсу "Системи штучного інтелекту" на факультеті кібернетики Київського національного університету ім. Тараса Шевченка та факультеті інформатики Національного університету "Києво-Могилянська академія". Тому більшість матеріалів підручника пройшла апробацію в студентських аудиторіях цих університетів. Накопичені знання і досвід втілені в цьому підручнику. Книга може бути корисна студентам університетів спеціалізацій "Прикладна математика" та "Комп'ютерні науки". Вона орієнтована на аспірантів, інженерів, наукових працівників та широке коло спеціалістів, інтереси яких пов'язані з розробкою і дослідженнями у галузі штучного інтелекту.

Ідея написання цієї книги знайшла підтримку керівництва університету "Києво-Могилянська академія", а потім і спонсорську підтримку видання родиною меценатів Фещенко-Чопівських (США). Публікації книги Видавничим домом "КМ Академія" автори зобов'язані рекомендаціям Вченої ради університету. Автори дякують Кривому С. Л. та Анісімову А. В. за копітке наукове рецензування підручника. В ході наукової редакції автори врахували ряд цінних зауважень і порад, які сприяли підвищенню якості тексту рукопису.

Авторам надавали технічну допомогу: при написанні книги — Проценко В. С., Гороховський С. С., Омельченко В. В., Кислоокий В. М., Глибовець А. М.; в роботі над верстkою — Романенко В. Ю., Слєсаренко Н. І., Ярошенко І. Г.

Автори висловлюють найщирішу подяку студентам, що слухали лекції авторів і внесли найбільшу кількість зауважень та пропозицій для поліпшення методологічних аспектів висвітлення матеріалу підручника.

Книга присвячується нашим дружинам, Галині та Валентині, в знак подяки за їх терпимість, любов і підтримку.

ВСТУП

Курс "Штучний інтелект" є одним з базових у підготовці майбутніх фахівців у галузі комп'ютерних технологій. У той же час на сучасному етапі в Україні практично відсутні підручники з теорії і практики побудови інтелектуальних систем. Дуже гостро відчувається і нестача сучасної літератури. Пропонований підручник "Штучний інтелект", спрямований на заповнення згаданої прогалини, призначений для студентів вищих навчальних закладів, які навчаються за спеціальностями "Комп'ютерні науки" та "Прикладна математика". Підготовлено його на основі матеріалів лекцій і практичних занять, які протягом кількох років читалися і проводилися авторами в Національному університеті "Києво-Могилянська академія" та Київському національному університеті ім. Тараса Шевченка.

Підручник знайомить студентів з основними підходами до вирішення інтелектуальних задач, до яких насамперед належать недостатньо формалізовані задачі. Висвітлюються основні принципи побудови та функціонування інтелектуальних систем, методи та алгоритми вирішення типових інтелектуальних задач, а також основні сучасні досягнення у цій галузі (експертні системи розпізнавання образів та ін.). Робиться також спроба систематизувати і подати у вигляді, доступному для студентів, останні світові наукові досягнення, накопичені наприкінці 90-х років.

Підручник орієнтовано на ітераційну схему викладання. Спочатку на досить неформальному рівні вводяться основні поняття і формулюються загальні принципи роботи інтелектуальних систем. Потім детально вивчаються моделі подання знань і основні алгоритми вирішення типових інтелектуальних задач. Після цього базові парадигми проектування і функціонування систем штучного інтелекту уточнюються і розглядаються більш формалізовано і систематизовано. Робиться огляд сучасних практичних досягнень.

Підручник складається з восьми частин, які поділяються на логічно завершені розділи, а розділи — на параграфи. Кожний розділ закінчується набором контрольних запитань і тем для обговорення, до яких іноді додаються задачі і вправи.

У першій частині розглядаються базові поняття, пов'язані з природним і штучним інтелектом, аналізуються основні сучасні визначення штучного інтелекту та наводиться їх критика. Розглядається методика тесту Тьюринга, спрямована на визначення рівня інтелектуальності, та фатичний діалог. Вивчається одне з центральних понять теорії штучного інтелекту — квазіалгоритм, реалізація якого залежить від неконтрольованих факторів. Наводяться типові джерела квазіалгоритмічності. Дається систематизована характеристика інтелектуальних систем із загальнокібернетичних позицій. На цій основі описується типовий цикл функціонування інтелектуальної системи: сприйняття зовнішньої ситуації та формування первинного опису — зіставлення зі знаннями та формування вторинного опису у термінах знань системи — планування цілеспрямованих дій і прийняття рішень — реакція системи. Особливо наголошується на ітеративності цього процесу.

У другій частині наводиться характеристика знань як інформаційної основи інтелектуальних систем. Досліджуються відмінності між знаннями та традиційними даними; розглядаються основні властивості знань. Наголошується на інтенсіональному характері знань і на їх структурованості. Детально розглядаються основні сучасні моделі знань: семантичні мережі, фрейми, логічні моделі, продукційні системи; значна увага приділяється зв'язкам між цими моделями знань. Вивчається один з основних механізмів логічного виведення — метод резолюцій; розглядаються інші механізми, характерні для семантичних мереж і фреймів. На прикладі семантичних мереж і фреймових моделей вивчаються деякі підходи до вирішення важливої практичної задачі — розуміння нової інформації, зокрема природної мови, і поповнення цієї інформації на основі існуючих знань.

Третя частина присвячена методам та алгоритмам логічного виведення в умовах неточної і недостовірної інформації. Розглядаються модальні логіки, методи роботи з недостовірною інформацією, зокрема схема EMYCIN; вивчається теорія нечітких множин і методи нечіткого логічного виведення.

У четвертій частині детально розглядаються методи планування цілеспрямованих дій і прийняття рішень. Розглядаються основні алгоритми планування в просторі станів і просторі задач. Значна увага приділяється бектрекінговим алгоритмам, динамічному програмуванню, методу Харта, Нільсона і Рафаеля та ін. Розглядаються типові вирішувачі інтелектуальних задач, зокрема "Загальний вирішувач задач". Вивчаються класичні методи вирішення ігрових задач: повна мінімаксна процедура та мінімаксна процедура на усіченому дереві, альфа-бета-відтинання, алгоритм ретроспективної розстановки поміток. Дається аналіз сучасних шахових програм. Розглядається підхід людини до вирішення ігрових задач та обговорюються можливості втілення цього підходу в системах штучного інтелекту.

У п'ятій частині аналізуються основні загальноінтелектуальні метапроцедури. Наводиться характеристика стимульно-реактивної теорії, яка ілюструється навчанням автоматів з лінійною тактикою, лабіринтної та модельної теорій. Значна увага приділяється моделям і методам навчання і самонавчання; задачам формування понять і гіпотез та пошуку аналогій. Приділяється увага генетичним алгоритмам — сучасному напряму в теорії навчання.

У шостій частині вивчаються основні методи розпізнавання образів: дискримінантні та структурні.

У сьомій частині дається характеристика конекціоністського підходу до побудови систем штучного інтелекту, який лежить в основі нейрокомп'ютерної техніки. Вивчаються модельні нейрони як порогові елементи та персептрони, побудовані на базі цих елементів. На основі аналізу, зробленого М. Мінським та С. Пейпертом, розглядаються можливості персептронів та їх обмеженість. Описуються сучасні підходи до побудови нейронних мереж, зокрема мережі Хопфілда.

У восьмій частині описуються основні підходи до обробки природної мови.

Автори сподіваються, що підручник стане у пригоді не тільки студентам, аспірантам і викладачам навчальних закладів III—IV рівнів акредитації, а й широкому колу фахівців з комп'ютерних наук і прикладної математики.

ЧАСТИНА I
НЕФОРМАЛЬНИЙ ВСТУП ДО ПРЕДМЕТА ШТУЧНОГО ІНТЕЛЕКТУ

Розділ 1
ПРИРОДНИЙ І ШТУЧНИЙ ІНТЕЛЕКТ

Я не можу сказати, що це таке, але
завжди впізнаю, коли побачу.

Д. Мічі, Р. Джонстон.
"Комп'ютер-творець"

1.1. Інтуїтивне розуміння поняття "інтелект"

З давніх-давен людині були необхідні помічники для полегшення виконання тих чи інших операцій. Тож людська думка винаходила різноманітні механізми, машини та інші витвори homo sapiens. Поява електронно-обчислювальних машин дала змогу автоматизувати трудомісткі розрахункові операції. Згодом стало зрозуміло, що ці машини можна використовувати не тільки для обчислень, а й для керування різними пристроями, складними автоматизованими виробництвами тощо. Поширення набули ***роботи*** — програмно керовані пристрої, здатні безпосередньо взаємодіяти з фізичним світом та виконувати в ньому певні задані дії [153]. Такі роботи широко застосовуються на багатьох виробництвах. Вони можуть працювати і в умовах, яких людина не в змозі витримати,— на дні океану, всередині ядерного реактора та ін. Але такі помічники вимагають автоматизованого керування, тобто їм постійно потрібно "підказувати", що і як вони мають робити у визначеній ситуації.

Природна мова на сучасному етапі малопридатна для цього через свою складність і неоднозначність. Один з шляхів вирішення згаданої проблеми — це формулювання інструкцій мовою, зрозумілою виконавцеві, навіть якщо він — робот, тобто написання програм. Програмування полягає у перекладі інструкцій, написаних мовою, близькою до природної, мовою, яку здатна сприйняти обчислювальна система. Відомі складності сучасного програмування, пов'язані з необхідністю надмірної алгоритмізації, тобто детального ретельного розписування своєрідних програмних кроків,— інструкцій з урахуванням усіх можливих ситуацій. З цієї ситуації існує єдиний вихід — підвищення рівня "розумності", інтелектуальності сучасних комп'ютерів і роботів. Тож постає питання: що розуміється під такими поняттями, як "інтелектуалізація", ***"штучний інтелект"***?

Можна стверджувати, що "штучний" інтелект у тому чи іншому розумінні повинен наближатися до інтелекту природного і у ряді випадків використовуватися замість нього, так само, як, наприклад, штучні нирки працюють замість природних. Чим більше існуватиме ситуацій, в яких штучні інтелектуальні системи зможуть замінити людей, тим інтелектуальнішими вважатимуться ці системи.

Навряд чи є сенс протиставляти штучний інтелект і інтелект людини. Слід спробувати визначити поняття "інтелект" взагалі, незалежно від його походження.

Людина вважається інтелектуальною від природи — її інтелект вироблявся протягом мільйонів років еволюції. Тож людина вміє вирішувати багато інтелектуальних задач. Кожна людина вважає, що розуміє значення слова "інтелект", але якщо попросити її навести тлумачення цього слова, в переважній більшості чіткої відповіді не дістанемо. І дійсно, дати визначення поняття "інтелект", яке б задовольняло всіх, очевидно, неможливо (далі будуть проаналізовані деякі такі спроби). Але відсутність чіткого трактування не заважає оцінювати інтелектуальність на інтуїтивному рівні.

Можна навести як мінімум два методи такої оцінки: ***метод експертних оцінок*** і ***метод тестування*** [222].

У разі застосування методу експертних оцінок рішення про ступінь інтелектуальності приймає досить велика група експертів (незалежно або у взаємодії, способів організації якої відомо досить багато).

У разі використання методу тестування пропонується розв'язати ті чи інші тестові завдання. Існує значна кількість інтелектуальних тестів, апробованих практикою, для оцінки рівня розумових здібностей людини, що знайшли застосування в психології та психіатрії [6, 7, 109, 140]. Наведемо кілька прикладів.

Приклад 1.1. Вставте пропущене число:

36 30 24 18 6

Приклад 1.2. Викресліть зайве слово:

лев лисиця жираф щука собака

Серед психіатрів інколи можна почути вислів: "Він розмірковує логічно вірно, але неправильно" [222]. Наприклад, дається тестове завдання: "Серед слів **соловей, чапля, перепілка, стілець, шпак** виокремити зайве". Більшість людей, не замислюючись, наводить відповідь **стілець**, оскільки всі інші слова — це назви птахів. І раптом хтось дає відповідь **шпак**, пояснюючи це тим, що це єдине слово, в якому відсутня літера "л". Обидві класифікації є логічно вірними і формально рівноправними. Але чому ж перевага віддається одній з них? Тому, що так гадає більшість людей. А чому вони так роблять? Очевидно, тому, що **перша класифікація вважається важливішою для практичної діяльності людини**. Це положення приймається на аксіоматичному рівні, без доведення (запропонувати щось інше, очевидно, неможливо).

Потрібно наголосити, що поняття "штучний інтелект" не можна зводити лише до створення пристроїв, які повністю або частково імітують діяльність людини. Не менш важливим є інше завдання: **виявити механізми, які лежать в основі діяльності людини, щоб застосувати їх при вирішенні конкретних науково-технічних завдань**. І це лише одна з можливих проблем.

1.2. Приклади інтелектуальних задач

Розглянемо і проаналізуємо в загальних рисах деякі проблеми, які доводиться постійно вирішувати людському розуму: ***розпізнавання образів***, ***мислення*** та ***обчислювальні задачі***.

На інтуїтивному рівні можна сформулювати кілька типових завдань розпізнавання образів (або просто розпізнавання):

- завдання ***ідентифікації*** [1] полягає в тому, що об'єкт, який спостерігається людиною, потрібно вирізнити з-поміж інших (наприклад, побачивши іншу людину, впізнати у ній свою дружину);
- проблема ***розпізнавання в класичній постановці***: визначити належність об'єкта, що спостерігається, до одного із заздалегідь відомих класів об'єктів (наприклад, відрізнити легковий автомобіль від вантажного).

Людина класифікує просто. Наприклад, чоловік, повернувшись додому з роботи, відразу ж впізнає свою дружину, але більшість людей не зможе пояснити, як він це робить, у повному обсязі. Як правило, раціонального пояснення немає. Теорія розпізнавання, яка інтенсивно розвивається, необхідна для того, щоб навчити штучні інтелектуальні системи розв'язувати завдання розпізнавання на основі досвіду розпізнавання людиною. Зокрема, сформульовано такий ключовий принцип [264]: ***будь-який об'єкт у природі — унікальний; унікальні об'єкти — типізовані.*** Відповідно до цього принципу розпізнавання здійснюється на основі аналізу певних характерних **ознак**. Вважається, що в природі не існує двох об'єктів, для яких збігаються **абсолютно всі** ознаки, і це теоретично дозволяє здійснювати ідентифікацію. Якщо ж для якихось об'єктів **деякі** ознаки збігаються, ці об'єкти теоретично можна об'єднувати в групи або класи саме за цими ознаками.

Проблема полягає у тому, що різноманітних ознак існує незліченна кількість. Незважаючи на легкість, з якою людина проводить розпізнавання, вона дуже рідко в змозі виокремити суттєві для цього ознаки. До того ж об'єкти, як правило, змінюються з часом. Тому далі ми спробуємо показати, що розпізнавання об'єктів і ситуацій має виняткове значення для орієнтації людини в навколишньому світі і прийняття нею вірних рішень. Роз-

[1] Таке застосування терміна "ідентифікація" не є єдино можливим. Він часто вживається в іншому значенні, а саме: знаючи вхідні та вихідні сигнали деякої системи, можна визначити її структуру.

пізнавання, як правило, здійснюється людиною на інтуїтивному, підсвідомому рівні — адже людина вчилася цьому мільйони років.

Інша інтелектуальна задача — моделювання ***мислення***, зокрема формулювання наслідків з фактів, які безпосередньо спостерігаються або вважаються відомими. Можна виокремити два типи процесів мислення:

- підсвідоме ***інтуїтивне*** мислення, механізми якого на сучасному етапі вивчені недостатньо і яке дуже важко формалізувати та автоматизувати;
- ***дедуктивні*** логічні побудови за формалізованими законами логіки.

Дедукцією *називається перехід від загального до часткового, виведення часткових наслідків із загальних правил.*

Тут пересічна людина рідко в змозі пояснити, за якими алгоритмами вона здійснює логічне виведення. Але методики та алгоритми, за якими **можна** автоматизувати виведення наслідків з фактів або логічну перевірку тих чи інших фактів, відомі. Насамперед це формальна логіка Арістотеля, сформована на основі конструкцій, які дістали назву ***силогізмів***. Вони практично неподільно панували в логіці аж до початку XIX століття, тобто до появи булевої алгебри. Немає ніякої необхідності наводити якісь формальні визначення силогізмів. Наведемо лише один класичний приклад.

Перше твердження: *Усі люди смертні.*
Друге твердження: *Сократ — людина.*
Висновок: *Сократ смертний.*

Якщо перше і друге твердження у силогізмі істинні та задовольняють певним загальним формальним вимогам, тоді й висновок буде істинним, незалежно від змісту тверджень, що входять до силогізму. При порушенні цих формальних вимог легко припуститися логічних помилок, подібних до нижченаведених:

Усі студенти вузу А знають англійську мову.
Петров знає англійську мову.
Отже, Петров — студент вузу А.

Або:

Іванов не готувався до іспиту і отримав двійку.
Сидоров не готується до іспиту.
Отже, і Сидоров отримає двійку.

Арістотелем було запропоновано кілька формальних конструкцій силогізмів, які він вважав достатньо універсальними. Лише у XIX столітті почала розвиватися сучасна математична логіка, яка розглядає силогізми Арістотеля як один з часткових випадків. Основою більшості сьогоднішніх систем, призначених для автоматизації логічних побудов, є ***метод резолюцій*** Дж. Робінсона (буде описаний далі). Але практична реалізація логічних побудов зіткнулася з серйозними проблемами. Головна з них — це феномен, який Р. Беллман назвав ***прокляттям розмірності***. На перший погляд, описати знання про зовнішній світ можна було б, наприклад,

таким чином: *"Об'єкт А має риси X, Y, Z. Об'єкт В відрізняється від А тим, що має рису Н, і т. д."*. Але зовнішній світ є винятково складним переплетінням різноманітних об'єктів і зв'язків між ними. Для того, щоб тільки ввести всю цю інформацію до пам'яті інтелектуального пристрою, може знадобитися не одна тисяча років. Ще триваліший період потрібен, щоб врахувати всі необхідні факти при логічному виведенні. Реальні програми, що здійснюють логічне виведення (вони часто називаються ***експертними системами***), мають досить обмежене застосування, лімітований набір фактів і правил з певної, більш-менш окресленої предметної галузі, і можуть використовуватися лише у цій галузі.

Щодо людини, то якість її логічного мислення також часто буває далекою від бездоганної. Люди часто роблять логічні помилки, а інколи взагалі керуються принципами, невірними з точки зору нормальної логіки. Дуже часто свідоме логічне виведення на певному етапі обривається, і рішення знову ж таки приймається на підсвідомому, інтуїтивному рівні. Зрозуміло, що таке рішення може бути помилковим. Але якби в основі поведінки людини лежали спроби проводити дедуктивні побудови від логічного початку до логічного кінця, людина була б практично нездатною до будь-якої діяльності — фізичної або розумової. Це вимагало б значного часу для аналізу.

Спільною рисою згаданих вище проблем є їх **погана формалізованість**, відсутність або незастосовність чітких алгоритмів розв'язку. Вирішення подібних задач і є основним предметом в теорії штучного інтелекту.

До зовсім іншого класу належать задачі, пов'язані з обчисленнями. В принципі, важко відповісти на запитання, як саме людина здійснює ті чи інші обчислення. Добре відомими є і низька швидкість, і невисока надійність цього виду людської діяльності. Тож були запропоновані ефективні принципи комп'ютерних обчислень, що радикально відрізняються від тих, які застосовуються людиною. Ці добре формалізовані алгоритмічні методики забезпечили такий рівень вирішення обчислювальних задач, який виявився абсолютно недосяжним для людського інтелекту. Водночас цей високий рівень алгоритмізації значною мірою зумовив слабкість традиційних комп'ютерних систем при розв'язанні тих інтелектуальних задач, з якими людина справляється непогано.

З появою таких обчислювальних потужностей мрійники 60—70-х років XX століття поставили завдання моделювати в повному обсязі роботу людського мозку. Теоретично цю проблему з певними обмеженнями можна було б вирішити. Але складність потрібних обчислень виявилася такою, що змусила більшість дослідників відійти від поставленого завдання і перейти до простіших задач: моделювання принципів роботи людського мозку при розв'язанні конкретно визначених типів задач.

1.3. Аналіз основних визначень поняття "інтелект"

Було зроблено чимало спроб дати формальне визначення поняття "інтелект", зокрема "штучний інтелект". Мабуть, відомішою з них є визначення предмета теорії штучного інтелекту, яке було введено в обіг

видатним дослідником у галузі штучного інтелекту Марвіном Мінським. Воно потрапило до багатьох словників та енциклопедій з невеликими змінами і відображає таку основну думку: *"Штучний інтелект є дисципліна, що вивчає можливість створення програм для вирішення задач, які при розв'язанні їх людиною потребують певних інтелектуальних зусиль"*. Але і це визначення має вади. Головна з них полягає у недосконалій формалізації поняття "певні інтелектуальні зусилля". Таких зусиль вимагає, наприклад, виконання простих арифметичних обчислень, але чи можна вважати інтелектуальною програму, яка здатна до здійснення тільки таких операцій? Відтак у ряді книг та енциклопедій до наведеного визначення додається поправка: *"Сюди не входять задачі, для яких відома процедура їх розв'язання"*. Важко вважати таке формулювання задовільним. Розвиваючи цю думку далі, можна було б продовжити: отже, якщо я не знаю, як виконувати якусь задачу, то вона є інтелектуальною, а якщо знаю,— то ні. Наступний крок — це відомі слова Л. Теслера: *"Штучний інтелект — це те, чого ще не зроблено"* [187]. Подібний парадоксальний висновок лише підкреслює дискусійність проблеми.

Існують і конструктивніші визначення інтелекту. Наприклад, в [282] наводиться одне з них: *"Інтелект є здатність правильно реагувати на нову ситуацію"*. Там наводиться і критика цього визначення: не завжди зрозуміло, що слід вважати новою ситуацією. Уявіть собі, наприклад, звичайний калькулятор. Цілком імовірно, що на ньому ніколи не обчислювали суму двох нулів. Тоді завдання "обчислити нуль плюс нуль" можна вважати ситуацією, новою для калькулятора. Безумовно, він з нею впорається ("правильно відреагує на нову ситуацію"), але чи можна на цій підставі вважати його інтелектуальною системою?

1.4. Тест Тьюринга і фатичний діалог

Відомий англійський учений Алан Тьюринг сформулював тезу, що визначає момент, з якого машину можна вважати інтелектуальною [222]. Нехай експерт за допомогою телефону або подібного віддаленого пристрою спілкується з кимось (або чимось), що може бути як людиною, так і машиною. Експерт дає певні тести-завдання. За результатами відповідей він повинен визначити, з ким має справу — з людиною чи ЕОМ. Якщо він приймає комп'ютер за людину, комп'ютер може вважатися інтелектуальним. Така перевірка дістала назву ***тесту Тьюринга***. Багато спеціалістів вважали, що подібного тесту цілком достатньо для визначення рівня інтелектуальності комп'ютерної системи. Але виявилось, що це не зовсім так.

В основі тесту Тьюринга лежить неявне припущення про те, що необхідною умовою ведення діалогу є розуміння співрозмовника. Але наприкінці 60-х років американський кібернетик Дж. Вейценбаум створив дві програми — ЕЛІЗА і ДОКТОР — для використання в психіатрії. Виявилось, що вони можуть невірно зорієнтувати експерта в умовах тесту Тьюринга.

Тому при побудові цих програм в основу була покладена ідея ***фатичного діалогу***.

Фатичним *називається* ***діалог*** *без розуміння співрозмовника, на основі формального перефразування почутого.*

Так, на висловлення пацієнта *"Ви не відверті зі мною"* програма може відповісти: *"Чому Ви думаєте, що я не відверта з Вами?"*. У ряді випадків програма може видати заздалегідь заготовлену фразу. Так, на слова пацієнта *"Мій батько мене не любив"* вона може відповісти *"Розкажіть мені про Вашу сім'ю"*.

Звичайно, це не має жодного відношення ні до розуміння, ні до інтелекту, але багато людей, які проводили з цими програмами тест Тьюринга, вирішили, що мають справу з людиною. У [222] наводиться курйозний приклад фатичного діалогу. Віце-президент великої американської компанії захотів зв'язатись з відомим системним програмістом Д. Бобровим. Відбулась така телетайпна розмова:

Віце-президент: *Чи можу я зараз скористатись комп'ютером?*
Бобров: *А чому ви питаєте?*
Віце-президент: *Може, мені вдасться підписати контракт.*
Бобров: *Чому ви в цьому впевнені?*
Віце-президент: *Мої потенційні покупці ніколи не бачили систему в дії.*
Бобров: *Це має для вас велике значення?*
Віце-президент: *Звичайно.*
Бобров: *Ви маєте в цьому сумнів?*
Віце-президент: *Що за безглузде зауваження?*
Бобров: *Що Ви маєте на увазі, кажучи про безглузде зауваження?*
Віце-президент: *Чи можу я зараз скористатись комп'ютером?*
Бобров: *Розкажіть мені ще про Ваших потенційних покупців.*
Віце-президент: *Передзвоніть мені 491—1850.*

Віце-президент був здивований ще більше, коли йому ніхто не зателефонував. Непорозуміння з'ясувалося пізніше. Виявилось, що телетайп з'єднав віце-президента не з самим Бобровим, а з програмою, яка реалізувала фатичний діалог.

Звичайно, подібний діалог важко вважати особливо осмисленим, але хто може дати чітке визначення рівня осмисленості? Фатичний діалог може бути і менш примітивним, але системи, які здатні тільки на підтримку такого діалогу, не можна вважати інтелектуальними. **Розуміння співрозмовника** є абсолютно необхідним для нормального діалогу, а для такого розуміння **необхідно мати певну суму знань про світ**.

1.5. Метод комп'ютерної реалізації фатичного діалогу

В основі псевдоінтелектуальних програм, які здатні підтримувати фатичний діалог з людиною або іншими аналогічними програмами, можуть лежати дуже прості алгоритми зіставлення зі зразками.

Нижче описується класичний метод, який дозволяє писати подібні програми [254].

Розглянемо "анатомію" програми "ДОКТОР" у дещо спрощеному вигляді.

Наприклад, діалог з цією програмою може бути таким:

Доктор: *Я Вас слухаю.*

Людина: *Мене бентежить погода.*

Доктор: *Як давно Вас бентежить погода?*

Людина: *Відтоді, як померла моя мати.*

Доктор: *Розкажіть про Вашу сім'ю.*

Людина: *Усі вони захоплюються комп'ютерами.*

Доктор: *Машини Вас лякають.*

Людина: *Ні.*

Доктор: *Будь ласка, не будьте так небагатослівні...*

І так далі до нескінченності.

В основі методу, необхідного для реалізації такого діалогу, може лежати **механічне порівняння** речень, які вводяться людиною, зі зразками (шаблонами) речень, що зберігаються програмою. Якщо речення збіглося повною мірою з одним із зразків, видається одна з відповідей, заздалегідь заготовлених для цього зразка.

Можна розглянути кілька варіантів порівнянь, кожний з яких може бути легко запрограмований.

Варіант 1 (повний збіг). Якщо речення, що вводиться, повністю збігається з одним із зразків, може прозвучати відповідь: **"Так, Ви маєте рацію"**, або навпаки: **"Ви помиляєтесь, оскільки..."**, і після **"рацію"** або **"оскільки"** програміст може написати будь-який текст, що імітує глибоке розуміння специфіки предметної області. Наприклад, дуже непогано виглядатиме такий діалог:

Людина: *Квадрат гіпотенузи дорівнює сумі квадратів катетів.*

Програма: *Так, але є подібний результат і для непрямокутних трикутників — це теорема косинусів.*

Навряд чи після кількох подібних відповідей у когось залишаться сумніви в інтелектуальних здібностях програми. Але, звичайно, повні збіги трапляються дуже рідко. Тому доводиться використовувати інші типи порівнянь.

Варіант 2 (використання замінювачів). Типовим є використання замінювачів * і ?. Із замінювачем * зіставляється довільний фрагмент тексту, із замінювачем ? — будь-яке окреме слово.

Наприклад, шаблон (* *комп'ютери* *) успішно зіставляється з будь-яким реченням, в якому згадується про комп'ютери; шаблон *(Я люблю ? яблука)* — з такими реченнями, як *(Я люблю червоні яблука), (Я люблю солодкі яблука)* тощо (але не з реченням *Я люблю їсти зелені яблука*).

Варіант 3 (надання значень змінним у процесі зіставлення). При цьому можливості програми, що реалізує фатичний діалог, значно розширюються.

Вона набуває здатності до генерації відповідей, які залежать від запитань. Так, правило

$$(Я\ ?\ *) \xrightarrow{a:=?} (Що\ Ви\ ще < a > ?)$$

дозволяє на речення *Я люблю яблука* відповісти *Що Ви ще любите?*, а на речення *Я ненавиджу дощі — Що Ви ще ненавидите?* У цьому прикладі при успішному зіставленні змінній *a* надається значення слова, з яким збігається замінювач *?*. Безумовно, при використанні українських фраз замість англійських потрібно ще стежити за узгодженням суфіксів і закінчень.

Варіант 4 (універсальний зразок). Зі зразком (*) зіставляється будь-яке речення. Звичайно, і відповіді, що відповідають цьому зразкові, повинні бути такими ж універсальними. Наприклад:

(Я Вас не дуже розумію)
(Не будьте такими небагатослівними)
(Чому це має для Вас значення?)

і т. п.

Варіант 5 (зіставлення більше ніж з одним зразком). Речення може зіставлятися не з одним зразком, а кількома. Наприклад, якщо в програмі задані підстановки

$$(*\ люблю\ *) \rightarrow (Що\ Ви\ ще\ любите?)$$

та

$$(*\ комп'ютери\ *) \rightarrow (Ви\ маєте\ здібності\ до\ техніки),$$

то речення *(Я люблю комп'ютери)* зіставляється з обома зразками.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Яка потреба в інтелектуалізації комп'ютерних технологій?
2. Назвіть кілька відомих вам інтелектуальних задач. Опишіть їх спільні риси та відмінності.
3. У чому полягає основна проблема при автоматизації логічних побудов?
4. Чи є логічне мислення людини бездоганним?
5. Чи можна дати повне і загальновживане визначення інтелекту взагалі і штучного інтелекту зокрема?
6. Наведіть визначення предмета штучного інтелекту, дане М. Мінським. Дайте критичний аналіз цього визначення.
7. Опишіть процедуру тесту Тьюринга.
8. Охарактеризуйте поняття фатичного діалогу.
9. Яким чином можна запрограмувати фатичний діалог?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Охарактеризуйте відомі методи оцінки інтелектуальних здібностей. Що між ними спільного, а в чому вони різняться?
2. У чому полягають основні проблеми застосування інтелектуальних тестів для оцінки штучного інтелекту?

3. Чи зводиться процес отримання нових знань, зокрема виведення нових фактів з уже існуючих, до чисто логічного виведення?
4. Охарактеризуйте роль інтуїції в розумовій діяльності людини.

ЗАДАЧІ І ВПРАВИ

1. Проаналізуйте текст програми doctor.lsp, яка входить до стандартного комплекту поставки ряду реалізацій Ліспу. Які типи зіставлень використовуються в цій програмі? Покажіть фрагменти програми, які реалізують ті чи інші зіставлення.

Розділ 2
КІБЕРНЕТИЧНІ СИСТЕМИ

Потягни за мотузку — двері відчиняться...
З народної казки

2.1. Поняття кібернетичної системи

Системи, що є носіями природного або штучного інтелекту, можуть бути охарактеризовані з погляду ***кібернетики*** — науки, яка вивчає риси, спільні для складних систем будь-якої природи. Зокрема, центральне місце в кібернетиці посідає вивчення процесів перетворення інформації у системах та керування ними. Основи цієї науки були викладені у 1948 р. видатним американським ученим Норбертом Вінером у його видатній роботі "Кібернетика, або Керування і зв'язок у тварині і машині" [48].

Ключовим поняттям кібернетики є ***кібернетична система***, визначена в [243] як *сукупність пов'язаних один з одним об'єктів (елементів системи), що здатні сприймати, зберігати, перетворювати інформацію, а також обмінюватися нею.* Так, людський організм можна розглядати як систему, що складається з окремих елементів-клітин; органи людського тіла при цьому розглядаються як ***підсистеми***.

Відомим є ще одне визначення системи:

Системою *називається сукупність порівняно простих елементарних структур або процесів, що взаємодіють між собою та об'єднані в єдине ціле для виконання певної спільної функції, що не зводиться до функцій окремих компонент. Система має такі ознаки:*

- *взаємодіє із зовнішнім середовищем та іншими системами як єдине ціле;*
- *є ієрархією підсистем більш низького рівня;*
- *є підсистемою для деякої системи вищого рівня;*
- *зберігає загальну структуру взаємодії елементів при зміні зовнішніх умов і внутрішнього стану.*

Складні кібернетичні системи прийнято описувати на двох рівнях.

З одного боку, систему можна визначити як ***інформаційний перетворювач*** [82], що сприймає від зовнішнього середовища деякі ***вхідні величини*** (подразники, стимули) і залежно від цих подразників та свого ***внутрішнього стану*** генерує ***вихідні величини*** (реакції). Інакше кажучи, якщо через $x_1, \ldots, x_n$ позначити вхідні подразники, через $y_1, \ldots, y_m$ — реакції системи, а через $a_1, \ldots, a_q$ — змінні, що визначають поточний стан, то

$$y_j = f(x_1, \ldots, x_n, a_1, \ldots, a_q), j = 1, \ldots, m.$$

Функція f, що визначає інформаційне перетворення, може бути задана явно або неявно, система — описуватися складними математичними рівняннями: диференційними, інтегральними та ін.

До розгляду можна залучити і внутрішню структуру системи. Тоді остання визначається як пара $S = (M, R)$, де M — сукупність елементів системи, R — сукупність зв'язків між цими елементами. Тоді ***підсистемою*** S' системи S називається ***пара*** $S' = (M', R')$, де $M' \subseteq M$, $R' \subseteq R$. Можна окремо виділяти ***вхідні*** та ***вихідні елементи*** системи, які взаємодіють із зовнішнім світом, та її внутрішні елементи [166].

2.2. Класифікація кібернетичних систем

Поширеною є класифікація кібернетичних систем, яка залучає до розгляду два критерії:

1. **Ступінь визначеності функціонування.** За цим критерієм системи поділяються на:
   - *детерміновані*, елементи якої взаємодіють точно визначеним чином;
   - *імовірнісні*, для яких фактори, що впливають на елементи, або зв'язки між елементами не можуть бути точно описані, і результат функціонування системи стає не повністю визначеним.
2. **Ступінь складності систем**, за яким вони поділяються на:
   - *прості*, які можуть бути описані простими математичними моделями з невеликою кількістю рівнянь і змінних;
   - *складні*, які описуються складними математичними моделями з такою великою кількістю рівнянь і змінних, що їх точне моделювання стає неможливим або недоцільним;
   - *дуже складні*, для яких не існує адекватного задовільного математичного опису.

2.3. Керування кібернетичними системами

Проблема керування складними системами є однією з ключових у кібернетиці. Традиційне завдання ***керування*** *можна неформально сформулювати так: перевести керований об'єкт до бажаного стану або забезпечити перебування керованого об'єкта у потрібному стані. При цьому важливо мінімізувати затрати, пов'язані з керуванням.*

Серед учених, які внесли значний вклад у розвиток теорії керування, можна відмітити Н. Вінера, Р. Беллмана, О. Ляпунова, Р. Калмана, М. Моїсеєва, В. Глушкова. Значне місце в їх працях посідає принцип оптимального керування, відомий як принцип оптимальності Л. Понтрягіна [214]. Важливий вклад у розвиток теорії керування кібернетичними системами був зроблений київською школою вчених (Б. Бублик, М. Кириченко, Ф. Гаращенко, О. Наконечний) [27—29, 60—69, 136—139].

Можна розглядати також завдання структурного керування, які полягають у виборі оптимальної структури системи. Слід зауважити, що написання програм, які можуть бути виконані за допомогою універсальних комп'ютерів або спеціалізованих процесорів, можна вважати окремим видом керування.

Розрізняють такі ситуації:

- коли вдається побудувати точну ***математичну модель***, інакше кажучи, докладно описати керовану систему за допомогою математичних співвідношень і отримати адекватну постановку задачі керування; причому якщо існують ефективні точні методи вирішення цієї задачі, керування може бути **точним**. Як правило, точні методи керування можна застосовувати тільки для порівняно простих систем;
- для складніших систем доводиться застосовувати **наближені** методи керування; пов'язано це з тим, що або вдається побудувати лише приблизну модель об'єкта, або існують лише наближені методи вирішення відповідних рівнянь;
- для типових дуже складних систем (таких, як живий організм, корпорація, суспільство) побудувати математичну модель у принципі неможливо; можна сподіватися на певний приблизний опис лише окремих сторін функціонування таких систем. Керування ними має неформальний характер. Тому завдання такого керування можна вважати одним з основних у теорії штучного інтелекту. Ці проблеми стали предметом розгляду окремого наукового напряму, який дістав назву ***ситуаційного керування*** [221].

2.4. Контур керування та зворотний зв'язок

Ключовим поняттям керування є поняття ***"контур"***. Розрізняють два основні типи контурів керування: ***замкнений*** та ***розімкнений***.

На рис. 2.1 продемонстровано найпростіший розімкнений контур керування, який передбачає наявність двох основних елементів: керуючого та керованого. Для простоти викладу вважається, що кожний елемент має одну вхідну і одну вихідну величину (один вхід і один вихід). Вихідні сигнали керуючого елемента є вхідними для керованого елемента. Отже, для того, щоб забезпечити потрібні вихідні реакції керованого елемента, необхідно перш за все налаштувати керуючий елемент.

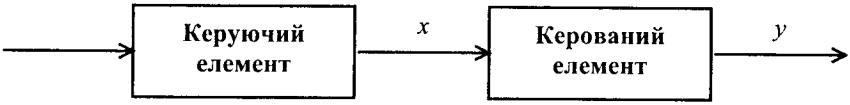


Рис. 2.1. Розімкнений контур керування

Очевидно, що розімкнений контур керування не є достатньо гнучким. Часто доводиться переналагоджувати систему. Виникає питання, чи не можна автоматизувати таке переналагодження? Інакше кажучи, чи не може контур керування набути необхідних ***адаптивних*** характеристик? Так, адаптивне керування може бути здійснено в рамках замкненого контуру на основі одного з найфундаментальніших принципів теорії автоматичного керування і кібернетики — ***принципу зворотного зв'язку***. Цей принцип полягає в тому, що вихідні сигнали системи знову подаються на її вхід (рис. 2.2).

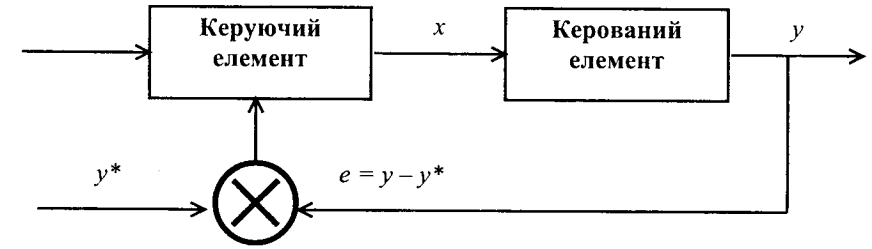


Рис. 2.2. Замкнений контур керування зі зворотним зв'язком

До складу керуючого елемента включено новий компонент, який порівнює еталонне значення виходу y^* з поточним виходом y. Різниця $e = y - y^*$ аналізується керуючим елементом, і на підставі цього аналізу коригується керуючий сигнал x.

Взагалі, на вхід системи може подаватися будь-яка функція $F(y)$ від виходу. Залежно від вигляду цієї функції розрізняють два основні типи зворотного зв'язку:

- ***негативний***, спрямований на ліквідацію відхилення реакції системи від певного усталеного режиму;
- ***позитивний***, спрямований на збільшення відхилення.

Слова "негативний" і "позитивний" не слід розуміти як "поганий" або "добрий". Обидва типи зв'язків можуть бути "добрими" або "поганими", залежно від конкретної системи і мети та умов функціонування цієї системи.

Наведемо приклади зворотних зв'язків у соціально-економічних системах.

Приклад негативного зворотного зв'язку (закон рівноваги попиту та пропозиції). Нехай під впливом тих чи інших факторів зростає ціна на певний товар. З одного боку, у даному разі зростає вигода виробни-

цтва товару, отже, збільшується пропозиція. З іншого боку, за рахунок зростання ціни спадає попит, забезпечений грошима. У результаті ціна на товар починає знову спадати; відбувається повернення до точки рівноваги.

Приклад позитивного зворотного зв'язку (модель розвитку інфляції). Нехай під впливом тих чи інших факторів знижується курс національної валюти, що призводить до падіння життєвого рівня населення. З метою компенсації цього падіння держава може прийняти рішення про грошову емісію (друкування додаткових грошей). У результаті відбувається подальше знецінення національної валюти і т. д.

Кібернетична система може мати будь-яку кількість зворотних зв'язків, як негативних, так і позитивних. Зрозуміло, що негативні зворотні зв'язки необхідні для підтримки деякого стану системи, а позитивні — для переведення системи з одного стану в інший. У цьому зв'язку важливо розуміти поняття "гомеостаз".

Гомеостазом *називається підтримка умов нормального функціонування системи, тобто підтримка найважливіших змінних, від яких залежить стан і поведінка системи, в діапазоні, що забезпечує її нормальне функціонування.*

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Сформулюйте визначення кібернетичної системи.
2. Сформулюйте визначення підсистеми. Наведіть приклади.
3. Наведіть класифікацію кібернетичних систем.
4. До якого типу належать:
   а) математичний маятник;
   б) комп'ютер;
   в) фірма;
   г) людина;
   д) будь-яка інтелектуальна система?
5. Охарактеризуйте задачу керування складними системами. У чому полягає принципова відмінність у керуванні простими, складними та дуже складними системами?
6. Опишіть принцип зворотного зв'язку, наведіть приклади позитивного і негативного зворотного зв'язку.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Поясніть, навіщо при аналізі систем виокремлювати підсистеми. Чи доцільно це робити?
2. Наведіть власні приклади кібернетичних систем з позитивними та негативними зворотними зв'язками.

Розділ 3

ІНТЕЛЕКТ ЯК ВИСОКООРГАНІЗОВАНА КІБЕРНЕТИЧНА СИСТЕМА

Солдат повинен бути розумним,
винахідливим і кмітливим.

Фольклор

3.1. Алгоритмічний і декларативний підходи до керування

Традиційно розрізняють два підходи до керування складними системами та до програмування роботів і комп'ютерів, які могли б розв'язувати ті чи інші задачі. У разі розв'язку задачі на сучасних комп'ютерах в основному реалізовано традиційний ***алгоритмічний підхід***, який можна ще назвати імперативним. Цей підхід вимагає заздалегідь продумати та детально розписати, **як** треба вирішувати певну проблему. Написання програми вимагає задання чітких послідовних інструкцій. Якщо таку послідовність вдається написати, комп'ютер зможе вирішувати поставлену задачу, якою б складною вона не була. Але, зрозуміло, він виконуватиме інструкції, абсолютно не розуміючи їх змісту.

Інший ***підхід*** — ***декларативний.*** Інтелектуальному виконавцеві (людині чи комп'ютеру) досить сказати, **що** треба робити, тобто лише сформулювати завдання, побудувавши всі взаємозв'язки між об'єктами предметної області. **Як** це завдання виконуватиметься — повинен визначити сам виконавець.

Наведемо для прикладу дві задачі [187]. Різниця в складності їх розв'язання зумовлена тим, що в одному випадку вдається формалізувати постановку і запропонувати чіткі алгоритми розв'язку, в іншому — ні. Отже, потрібно запустити космічний корабель так, щоб він приземлився на Місяць, взяв зразки ґрунту та привіз їх назад. Задача дуже складна, але вона піддається точній алгоритмізації. Математичні методи дозволяють чітко розрахувати траєкторію руху корабля. Навіть якщо він випадково відхилиться від цієї траєкторії, існують методи автоматичного регулювання, які дозволять ліквідувати це відхилення.

А от послати слухняного робота до магазину за пляшкою молока — задача на кілька порядків складніша. Не кажучи вже про сам процес спілкування з продавцем, робот повинен вирішити ряд не зовсім формалізованих підзадач. Заздалегідь розрахувати траєкторію руху неможливо, оскільки робот повинен уникнути зіткнень з людьми та автомобілями. Якщо магазин закритий на переоблік, він повинен знайти інший магазин. Неможливо передбачити всі ситуації, які можуть виникнути, а відтак — алгоритмізувати розв'язання задачі. Тому виконання такого завдання під силу лише інтелектуальній системі, яка вміє орієнтуватися в зовнішньому світі, аналізувати

поточні ситуації та коригувати, адаптувати свою поведінку на основі такого аналізу.

3.2. Поповнення первинних інструкцій

Інструкції, написані природною мовою, можуть бути, з одного боку, досить неоднозначними, а з іншого,— вони завжди спираються на те, що виконавець має певний апріорний досвід. Розглянемо приклад з [187]. Уявіть собі дві таблички. На першій, яка знаходиться при вході на будівництво, написано *"Обов'язково одягніть на голову каску"*. На другій, біля входу до лондонського метро,— *"Обов'язково візьміть на руки собаку"*. Ці інструкції однакові за формою, але абсолютно різні за змістом. Людина розуміє не лише те, що написано на табличках, але й те, що "залишається за кадром" і явно не зазначається. Людина уявляє собі ситуації, які можуть виникати на будівництві та в метро, і поповнює первинні інструкції, що сприймаються нею безпосередньо. Зокрема, на будівництві зверху можуть падати цеглини, і тому потрібна каска, щоб захистити голову. До такого висновку людина могла б дійти і самостійно, без нагадування. Без каски на будівництво не пустять, і, якщо каски немає, її потрібно придбати. Але ясно, що купувати собаку перед тим, як заходити в метро, зовсім не обов'язково.

Людина має певну суму **знань про світ**, яка дозволяє їй орієнтуватися в життєвих ситуаціях та приймати вірні рішення. Крім того, людина вміє певним чином **використовувати ці знання**. Цих самих рис повинна набути система штучного інтелекту. **Можна стверджувати, що здатність до поповнення тих первинних описів ситуацій, що сприймаються безпосередньо, є однією з ключових рис інтелектуальної системи.**

Зовнішній світ настільки багатий, що неможливо попередньо описати життєві ситуації з тим ступенем деталізації й однозначності, які б дозволили закласти в системи, що проектуються, жорсткі детерміновані алгоритми поведінки. Тому інтелектуальні системи повинні мати механізми адаптації, які б дозволяли їм вирішувати поставлені перед ними задачі на основі аналізу поточної ситуації.

3.3. Формалізація понять алгоритмічності та декларативності

Спробуємо формалізувати деякі з впроваджених раніше понять. Введемо такі позначення:

q — ***первинні інструкції***, записані без будь-яких змін у тому вигляді, в якому їх сформулював автор; аналогічно можна визначити ***первинний опис ситуації*** як результат її безпосереднього сприйняття;

S — множина факторів, які визначають поточний стан виконавців; розглядатимемо S як об'єднання двох множин: S' та S''. Тут S' — множина ***контрольованих факторів***, які відомі авторові процедури і на які він може мати вплив (в цілому, контрольованість і керованість — це не одне й те саме: фактор може бути контрольованим, проте некерованим, але для

наших цілей це не має суттєвого значення); S'' — множина ***неконтрольованих факторів***;

Z — знання, які має виконавець;

r_A — робочий алгоритм, який формується та реалізується виконавцем при алгоритмічному підході. При цьому, як правило, автоматично формується і програма p_A, що відповідає цьому алгоритму;

$r_Д$ — робочий алгоритм, який формується та реалізується інтелектуальним виконавцем при декларативному підході. Нагадаємо, що згідно з фундаментальними тезами Тьюринга та Чорча (все, що може бути виконано будь-яким виконавцем, може бути промодельовано на машині Тьюринга) сам факт виконання завдання свідчить про існування цього алгоритму. Щоправда, цей алгоритм може і не усвідомлюватися виконавцем, а в даному разі не може бути явно сформульований у вигляді алгоритму чи програми. Якщо ж програма, що відповідає алгоритму $r_Д$, повинна бути згенерована, йдеться про ***задачу синтезу програм***.

Тоді можна записати такі співвідношення:

$$r_A = f(q, S'),$$
$$r_Д = h(q, S', S'', Z),$$

де f і h — певні функції.

Принциповим є наступне питання. Можна вважати, що алгоритм r_A формується на основі первинних інструкцій q однозначно. При цьому також можуть відбуватися зміна і поповнення первинних інструкцій, але цей процес має повністю контрольований і детермінований характер. Як приклад можна навести компіляцію програм, написаних мовами високого рівня. Тому автор процедури може бути впевнений у гарантованому результаті (якщо, звичайно, були дотримані певні формальні вимоги до первинних інструкцій і забезпечено належний стан виконавця).

На противагу цьому, за декларативного підходу такої впевненості немає. Інтелектуальний виконавець певним чином поповнює первинні інструкції, але їх авторові не завжди відомо, яким чином це поповнення відбувається. Тому не можна бути впевненим у результаті виконання інструкцій, і ця втрата гарантованості є неминучою платою за відмову від алгоритмічності. Отже, слід розглянути узагальнене поняття алгоритму, яке дістало назву ***"квазіалгоритм"***.

3.4. Квазіалгоритми та джерела квазіалгоритмічності

Ми пересвідчилися, що чисто алгоритмічний підхід виявив свою неспроможність у разі створення інтелектуальних систем, призначених для вирішення досить складних неформалізованих задач. Тому для теорії і практики штучного інтелекту важливе значення має узагальнене поняття ***"квазіалгоритм"***, або ***"квазіалгоритмічна процедура"***.

Нагадаємо, що алгоритмом називається чітка недвозначна послідовність інструкцій, що зрозуміла виконавцеві і обов'язково приводить до

гарантованого результату за скінченний час. На відміну від цього, інструкції квазіалгоритму можуть бути не зовсім чіткими, тож результат виконання квазіалгоритмічної процедури не обов'язково є гарантованим.

Можна окремити як мінімум **чотири основні джерела квазіалгоритмічності**. Вони можуть доповнювати один одного, а у деяких випадках — описані схожими математичними моделями. Проте ці джерела мають принципово різну природу, а саме:

1. ***Дія випадкових чинників***, що не залежать від виконавця. Формально, з огляду на цей фактор, квазіалгоритмом слід вважати будь-який алгоритм, що розрахований на роботу за певних умов. Якщо ці умови зміняться, алгоритм може не привести до потрібного результату. Так, навіть програма на Паскалі, яка дає змогу обчислювати суму двох чисел, не буде завершена, якщо під час її виконання в комп'ютері випадково зникне напруга. Якщо ж не вдається чітко окреслити межі застосування алгоритму або забезпечити додержання необхідних умов для його роботи, ми маємо справу зі справжнім квазіалгоритмом.
2. ***Недостатнє врахування автором алгоритмічної процедури особливостей виконавця***. Це приводить до того, що виконавець неправильно розуміє, що від нього вимагається. Процедура може бути в принципі алгоритмічною, за нормальних умов приводити до гарантованого результату, але цей результат, можливо, виявиться передбачуваним для автора і навіть несподіваним. Наприклад, програміст бажає обчислити суму перших 50 натуральних чисел, але не знає, як працюють цикли. Він може вважати, що цикл виконується, поки лічильник **менший** за верхню межу (хоча насправді цикл виконується, поки лічильник менший за верхню межу або **дорівнює їй**). Тоді він може написати такий помилковий фрагмент програми на Паскалі:

$$S := 0; \textit{for } i := 1 \textit{ to } 51 \textit{ do } S := S + i.$$

3. ***Нечіткість формулювань***, що фігурують в описі процедури. Розглянемо будь-який кулінарний рецепт. Наприклад: *"Розтерти тісто, змішати з дрібно порізаними яблуками, додати солі і перцю за смаком і смажити до появи рум'яної скоринки"*. Це є типовим прикладом нечіткості формулювань, що приводить до невизначеностей. До якої межі слід розтирати тісто? Що означає "дрібно порізані"? "за смаком"? Як формалізувати момент "появи рум'яної скоринки"? Неінтелектуальна система, орієнтована на чисто алгоритмічне керування, просто не зрозуміє цього опису і тому не зможе його виконати. Інтелектуальна ж система, наприклад, людина, повинна спробувати поповнити й уточнити цей опис на основі наявних знань і досвіду. Сам по собі факт виконання завдання свідчитиме про те, що квазіалгоритмічний первинний опис процедури був зведений до деякого внутрішнього алгоритму, але навряд чи виконавець зможе чітко пояснити і розписати цей алгоритм. Крім того, такі внутрішні

алгоритми для кожного виконавця будуть різними, що зумовить неоднакові результати — у даному разі різні смакові якості виробу.

4. ***"Свобода волі"***, яку можуть мати високоорганізовані, по-справжньому інтелектуальні системи. У них є власні цілі, і якщо дана процедура їм суперечить, ці системи можуть відмовитися її виконувати або виконати не так, як це від них вимагається. Тобто "свобода волі" полягає у тому, що інтелектуальна система самостійно приймає рішення про свою поведінку і залежно від цих рішень може виконувати або не виконувати зовнішні розпорядження, або ж виконувати так, як вона вважає за потрібне.

Слід зазначити, що справжньою квазіалгоритмічністю найчастіше вважають ту, що зумовлена третім і четвертим чинниками.

3.5. Інтелектуальні системи із загальнокібернетичних позицій

Природно вважати інтелектуальну систему різновидом кібернетичної. Можна розглядати такі ситуації:

- чисто ***зовнішнє керування***: система сприймає зовнішні інструкції у вигляді речень природної мови, програм, написаних алгоритмічними мовами високого рівня, та ін. і намагається їх виконати. Ці зовнішні інструкції раніше були ***первинними інструкціями***. Тоді ці інструкції природно вважати зовнішніми чинниками для системи, і саме вони формують її мету;
- ***самокерування***: система на основі аналізу зовнішньої ситуації приймає рішення, орієнтовані на досягнення власної мети. Первинною при цьому є зовнішня ситуація у тому вигляді, в якому вона безпосередньо сприймається системою. Результат такого сприйняття раніше був названий ***первинним описом ситуації***;
- ***комбіноване керування***, за яким система в принципі може бути самокерованою, тобто керуватися власною метою. Однак вона може припускати і зовнішнє керування. Тоді вона розглядає мету керуючої системи як свою власну. Дуже важливим при цьому є питання пріоритетів, а саме: система повинна визначити, яка мета є для неї важливішою: власна чи зовнішня. Зіставлення зовнішньої мети з власною і визначає "свободу волі", про яку йшлося вище.

*Отже, **принциповою рисою інтелектуальної системи** є те, що вона намагається планувати свої дії для досягнення певної мети на основі первинного опису ситуації.*

Розглянемо дане положення детальніше. Система має певну мету і прагне так планувати свої дії, щоб досягти її. Вхідними стимулами системи можна вважати поточну ситуацію, що сприймається та аналізується нею. Результатом реакції системи стає зміна зовнішньої ситуації, і поведінка системи коригується залежно від того, бажаною чи небажаною є ця зміна.

Саме таке коригування і є реалізацією одного з основних зворотних зв'язків інтелектуальної системи.

Необхідною компонентою інтелектуальної системи є ***знання про світ***, на основі яких вона поповнює первинний опис зовнішньої ситуації для глибшого розуміння цієї ситуації, аналізує можливі наслідки своїх дій і, можливо, формує або коригує свою мету (див. рис. 3.1).

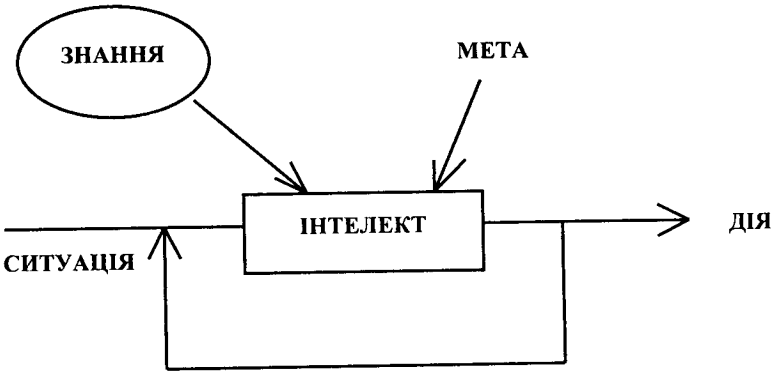


Рис. 3.1. Схема самокерування інтелектуальної системи

Усе це стосується будь-якої інтелектуальної системи, незалежно від того, чи йдеться про природний розум людини або інопланетянина, чи про штучний інтелект. За докладнішого трактування можна вважати, що чим **інтелектуальнішою є система, тим ефективніше вона може досягати мети**.

Людина не отримує всі свої знання і вміння відразу. Тож неможливо так само негайно передати всі необхідні знання і вміння системі штучного інтелекту, призначеній для вирішення досить складних задач. Тому винятково важливе значення має здатність інтелектуальної системи до ***навчання***, тобто до поповнення своїх знань і набуття нових можливостей. Таке навчання може відбуватися шляхом спілкування інтелектуальної системи з людиною або іншою інтелектуальною системою. Якщо ж навчання відбувається самостійно, йдеться про ***самонавчання***.

Справді розвинені інтелектуальні системи повинні мати здатність безпосередньо сприймати оточуючий світ за допомогою ***спеціальних органів чуттів***.

Більше того, це сприйняття має бути пов'язаним з тими поняттями, які входять до знань системи або які формуються в ній. Без такого зв'язку слова залишаються лише словами. Людина ж у процесі мислення істотно спирається на чуттєві асоціації, пов'язані з цими словами. Наприклад, слово "пожежа" викликає у свідомості людини цілий пласт уявлень, пов'язаних з такими зовнішніми подразниками, як "яскраво", "гаряче" і т. п. На основі цих асоціацій виникають інші. Все це має рефлекторний характер і зумов-

лює активну участь підсвідомих інтуїтивних процесів у людському мисленні, при прийнятті людиною рішень і т. п. Крім того, без набуття таких рис важко говорити про те, що система штучного інтелекту може мати якусь власну мету.

Для людини характерний тісний зв'язок "лівопівкульних" понять, якими вона оперує в процесі логічного мислення, з "правопівкульними" образами, які підкріплюються сигналами, що надходять від органів сприйняття зовнішнього середовища[1]. У сучасних системах штучного інтелекту такий зв'язок практично відсутній. На цій підставі Х. Дрейфус у 70-ті роки взагалі піддав сумніву можливість створення штучних систем, подібних до людини за своїми можливостями [106]. У зв'язку з цим слід знову згадати про ***інтелектуальних роботів***, а саме про їх здатність безпосередньо сприймати зовнішній світ за допомогою органів чуття. Проте можливості використання цього світосприйняття для активізації підсвідомих рефлекторних процесів у роботів поки що залишаються дуже слабкими.

До переліку інтелектуальних задач можна віднести такі:

- аналіз ситуацій;
- розпізнавання образів;
- логічне мислення;
- розуміння нової інформації;
- навчання і самонавчання;
- планування цілеспрямованих дій.

Інтелектуальною системою *називається самокерована кібернетична система, яка має певну суму знань про світ і здатна на основі безпосереднього сприйняття і подальшого аналізу поточної ситуації планувати дії, спрямовані на досягнення певної мети, а також поповнювати свої знання.*

Це визначення не може претендувати на повноту та універсальність. Воно фокусує увагу на основних завданнях, які повинна вміти вирішувати будь-яка інтелектуальна природна, технічна або програмна система.

3.6. Типова схема функціонування інтелектуальної системи

Зрозуміло, що функціонування інтелектуальної системи можна описати як постійне прийняття рішень на основі аналізу поточних ситуацій для досягнення певної мети. Тому природно виокремити стани, які утворюють ***типову схему функціонування інтелектуальної системи***:

1. Безпосередньо сприймає зовнішню ситуацію; результатом є формування первинного опису ситуації.

[1] Відомо, що мозок людини складається з двох півкуль: ліва відповідає за процеси логічного мислення, права — за виникнення зорових, слухових та інших образів, чуттєвих асоціацій і т. п.

2. Зіставляє первинний опис зі знаннями системи і поповнює цей опис; результатом є формування вторинного опису ситуації в термінах знань системи. Цей процес можна розглядати як процес *розуміння ситуації* або як процес перекладу первинного опису на внутрішню мову системи. При цьому можуть змінюватися внутрішній стан системи та її знання.

   Вторинний опис може бути не єдиним, і система має змогу вибирати між різними вторинними описами. Крім того, система в процесі роботи може переходити від одного вторинного опису до іншого. Якщо ми можемо формально задати форми внутрішнього представлення описів ситуацій та операції над ними, то сподіватимемося на певний автоматизований аналіз цих описів.
3. Планує цілеспрямовані дії та приймає рішення, тобто аналізує можливі дії та їх наслідки і вибирає ті дії, що найкраще узгоджуються з метою системи. Таке рішення зазвичай формулюється певною внутрішньою мовою (свідомо або підсвідомо).
4. Зворотно інтерпретує прийняте рішення, тобто формує робочий алгоритм для реагування системи.
5. Реалізація реакції системи; наслідками є зміни зовнішньої ситуації і внутрішнього стану системи і т. д.

Проте особливо зауважимо: не слід вважати, що зазначені етапи є повністю відокремленими у тому розумінні, що наступний етап починається тільки після того, як повністю закінчиться попередній. Навпаки, для функціонування інтелектуальної системи характерним є взаємне проникнення цих етапів. Наприклад, ті чи інші рішення можуть прийматися вже на етапі безпосереднього сприйняття ситуації. Насамперед це рішення про те, на які зовнішні подразники слід звертати увагу, а на які не обов'язково. Зовнішніх подразників так багато, що їх сприйняття повинно бути вибірковим.

3.7. Класифікація основних напрямів досліджень

Однією з найтрадиційніших класифікацій напрямів досліджень у галузі штучного інтелекту є класифікація Е. Ханта [264].

Перший напрям — ***біологічний***. В його основі лежать спроби вирішення інтелектуальних задач шляхом безпосереднього моделювання психофізіологічних особливостей мозку людини засобами електронно-обчислювальних машин. З часом стало зрозумілим, що розв'язати цю проблему в повному обсязі неможливо і недоцільно. Подібне моделювання особливостей людського розуму призведе до копіювання не тільки позитивних, але й негативних якостей. Стало очевидно, що таке моделювання, хоча й має певне значення для створення штучного інтелекту, більше підходить для вивчення інтелекту природного. Таке вивчення оформилося у вигляді самостійної науки, яка дістала назву ***когнітивної психології*** [247].

Інший напрям дослідження проблем штучного інтелекту ***прагматичний***. У ньому майже не розглядається психофізіологічна діяльність люд-

ського мозку, а розвиваються підходи до вирішення інтелектуальних задач, незалежно від того, як співвідносяться ці підходи з тими процесами, що відбуваються в мозку людини.

Інша класифікація систем штучного інтелекту визначається моделями і методами, що використовуються для вирішення практичних задач. Йдеться про ***символьний*** та ***конекціоністський підходи***, що детально розглядатимуться в ході вивчення даного курсу. Зараз же можна відмітити, що символьний підхід орієнтований на моделювання процесів свідомого логічного мислення, а конекціоністський — підсвідомих рефлекторних процесів. Ці підходи можуть застосовуватися як роздільно, так і в комбінації, проте найліпших успіхів можна досягти, лише комбінуючи їх. Зрозуміло, що без використання рефлекторних методів жодна серйозна інтелектуальна проблема не може бути вирішена через *прокляття розмірності.* З іншого боку, рефлекторні рішення далеко не завжди правильні та оптимальні, і тому людина не може обійтись без логічного мислення, як не можуть без нього обійтись і системи штучного інтелекту. Характерною особливістю сучасних досліджень є зближення та взаємопроникнення символьного та конекціоністського підходів.

І, нарешті, дослідження в галузі штучного інтелекту можна класифікувати ***за типами задач.*** Серед практичних інтелектуальних задач, у розв'язку яких можна відмітити реальні здобутки — розпізнавання образів, автоматизація логічних побудов, створення діалогових систем типу "людина—машина" та систем автоматизованого перекладу текстів, робототехніка. Останнім часом бурхливо розвиваються *інтелектуальні агенти* — автономні програмні системи [58, 321, 331].

Слід зазначити, що жодна з існуючих систем штучного інтелекту, які працюють у перелічених галузях, не може реалізовувати згадані вище функції інтелектуальної системи в достатньому обсязі. Тому про них краще говорити як про ***інтелектуалізовані системи***, які мають певні риси, що наближають їх до дійсно інтелектуальних систем.

3.8. Соціальні наслідки інтелектуалізації комп'ютерних технологій

Нарешті можна коротко зупинитися на соціальних наслідках інтелектуалізації інформаційних технологій. Позитивні наслідки зрозумілі та загальновідомі. З наслідків, не дуже сприятливих для людини, найочевидніший — це поступове витіснення людей машинами як у процесі виробництва, так і в управлінні суспільством. Подібні тенденції почали проявлятися ще задовго до початку комп'ютерної ери. Щодо майбутнього, то тут письменники-фантасти малюють похмурі картини повного захоплення влади комп'ютерами і навіть фізичного винищення людства.

Було запропоновано ряд принципів, орієнтованих на зменшення ризиків, пов'язаних з комп'ютеризацією та інтелектуалізацією. Найвідомішими

з них є сформульовані американським фантастом Айзеком Азімовим. Ось ці принципи:

- *комп'ютер або робот не повинен заподіяти шкоди людині;*
- *комп'ютер або робот має виконувати накази людини, якщо це не суперечить першому принципу;*
- *комп'ютер або робот повинен прагнути до самозбереження, якщо це не суперечить першим двом принципам.*

Проблема полягає в іншому: як ці або подібні їм принципи втілювати в життя. Слід зауважити, що час, коли штучний інтелект розвинеться настільки, що зможе реально загрожувати людському, настане ще дуже не скоро. Цей висновок, зроблений у 60—70-ті роки XX століття, залишається незмінним і сьогодні (якщо, звичайно, не розглядати можливості, пов'язані з "доступом до ядерної кнопки" і подібні їм).

Серед інших негативних аспектів інтелектуалізації можна відмітити те, що сучасні комп'ютерні технології полегшують здійснення не тільки корисних справ, але й реалізацію злих умислів, зловживань, злочинів тощо.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте алгоритмічний і декларативний способи керування; наведіть їх формалізований опис.
2. Що таке первинні інструкції, чому може виникати необхідність їх поповнення? Наведіть приклади.
3. У чому полягає принципова відмінність між поповненням первинних інструкцій за алгоритмічного і декларативного підходів?
4. Поясніть, чому реалізація декларативного способу керування пов'язана з втратою визначеності результату.
5. Що таке квазіалгоритм; у чому полягає відмінність квазіалгоритмічної процедури від алгоритмічної?
6. Охарактеризуйте інтелектуальну систему як самокеровану кібернетичну систему. Яким чином пов'язані найважливіші зворотні зв'язки з прагненням досягти певної мети?
7. Що виступає метою інтелектуальної системи при чисто зовнішньому керуванні нею?
8. Опишіть типову схему функціонування інтелектуальної системи.
9. Чому сприйняття зовнішнього світу інтелектуальною системою повинно бути вибірковим?
10. Охарактеризуйте основні риси інтелектуальної системи: самокерованість, наявність знань про світ, здатність досягати мети, здатність планувати свої дії, здатність до поповнення знань.
11. Охарактеризуйте біологічний і прагматичний напрями досліджень у галузі штучного інтелекту.
12. Дайте загальну характеристику символьного і конекціоністського підходів до створення систем штучного інтелекту.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. У чому, на вашу думку, проявляється квазіалгоритмічність людського мислення?
2. Охарактеризуйте основні завдання, які повинна вміти виконувати інтелектуальна система.
3. Поясніть, яким чином сприйняття зовнішньої ситуації може впливати на внутрішній стан і знання інтелектуальної системи. Наведіть кілька прикладів.
4. Охарактеризуйте поняття "свободи волі". Наведіть приклади.
5. Охарактеризуйте позитивні і негативні соціальні наслідки інтелектуалізації комп'ютерних технологій.
6. Наведіть відомі вам приклади практичних досягнень у створенні систем штучного інтелекту.

ЧАСТИНА II МОДЕЛІ ПОДАННЯ ЗНАНЬ І МЕТОДИ ЛОГІЧНОГО ВИВЕДЕННЯ

Розділ 4

ЗНАННЯ ЯК ІНФОРМАЦІЙНА ОСНОВА ІНТЕЛЕКТУАЛЬНИХ СИСТЕМ

Інтелектуальне казино "Що, де, коли?" є єдиним місцем, де кожен може заробити гроші завдяки власному розуму.

В. Ворошилов, беззмінний голова клубу знавців "Що, де, коли?"

4.1. Знання і деякі підходи до їх подання

У "Філософському словнику" [258] знання визначаються як "відображення об'єктивних властивостей та зв'язків світу".

***Знання** є інформаційною основою інтелектуальних систем, оскільки саме вони завжди зіставляють зовнішню ситуацію зі своїми знаннями і керуються ними при прийнятті рішень. Не менш важливим є те, що знання — це систематизована інформація, яка може певним чином поповнюватися і на основі якої можна отримувати нову інформацію, тобто нові знання.*

Існує багато підходів до визначення поняття "знання". На сучасному етапі домінуючою парадигмою, що лежить в основі найвідоміших моделей подання знань у системах штучного інтелекту, можна вважати парадигму (певну сукупність ключових принципів), характерну для ***символьного підходу*** [1]. Цю парадигму можна охарактеризувати як ***вербально-дедуктивну***, або ***словесно-логічну***, через певні чинники:

- будь-яка інформаційна одиниця задається вербально, тобто у формі, наближеній до словесної, у вигляді набору явно сформульованих тверджень або фактів;
- основним механізмом отримання нової інформації на базі існуючої є ***дедукція***, тобто висновок від загального до часткового. Такий дедуктивний підхід є бездоганним з логічного погляду. В його основі лежать транзитивність імплікації (якщо з a випливає b, з b випливає c,

[1] Символьний підхід до побудови та опису систем штучного інтелекту полягає у тому, що кожне поняття задається окремим словом (символом), зміст якого розкривається через зв'язок з іншими поняттями.

то з a випливає c) і дія квантора узагальнення (якщо деяка властивість P виконується для будь-якого елемента множини M, а $x \in M$, то P виконується для x).

Але вербально-дедуктивне задання знань не є повним, оскільки:

- дедуктивний висновок не виступає єдино можливим. Мислення людини багато в чому є рефлекторним, інтуїтивним. Воно, як правило, спирається на підсвідомі процеси. Людина часто робить висновки за аналогією, асоціацією. Ці висновки не завжди вірні, але вони істотно доповнюють процеси дедуктивного мислення. Провести дедуктивні логічні побудови з самого початку до кінця на практиці, як правило, неможливо. Без підсвідомого мислення стає неможливим будь-яке відкриття — як правило, підсвідоме породження гіпотези, яка потім перевіряється дедуктивним або експериментальним шляхом;
- далеко не всі знання є вербальними. Так, жодне твердження не зберігається в пам'яті людини явно. Відомо, що в основі діяльності мозку людини лежить передача сигналів між нервовими клітинами. Часто людина не може сформулювати свої знання. Наприклад, будь-хто знає, що таке "стіл". Але якщо попросити людину дати визначення цього поняття, можуть виникнути проблеми. Таким чином, поняттями можна оперувати і не знаючи чіткого визначення. Або типовими є слова: "Я не можу пояснити чому, але мені здається, що...".

Тому необхідно розвивати інші моделі знань, окрім вербально-дедуктивних. Наприклад, у рамках конекціоністського підходу [1] можна розглядати моделі на основі однорідного поля знань. ***Однорідним полем знань*** у даному разі називатимемо сукупність простих однорідних елементів, які обмінюються між собою інформацією: нові знання народжуються на основі певних процедур, визначених над полем знань.

Знання про відповідні факти або поняття ***зберігаються полем знань вербально***, якщо реалізація фіксованого набору процедур, заданих над полем знань, дозволяє за розумний час отримати вербальне формулювання того чи іншого факту або вербальний опис поняття. Знання ***зберігаються полем невербально***, якщо застосування фіксованого набору процедур, заданих над полем знань, дає змогу використовувати факт або поняття без отримання їх вербального опису.

Людина у своїй діяльності комбінує дедуктивні висновки з рефлекторними рішеннями. Таке комбінування повинно бути притаманне і розвиненим інтелектуальним системам. Наприклад, можна організувати паралельне функціонування механізмів дедуктивного виведення і конекціоністських механізмів.

---

[1] Конекціоністським називається підхід, у рамках якого система штучного інтелекту розглядається як однорідна сукупність порівняно простих елементів, що взаємодіють між собою.

4.2. Вербально-дедуктивне визначення знань

У рамках вербально-дедуктивної парадигми можна навести таке визначення знань.

Знаннями інтелектуальної системи називається трійка <F, R, P>, де F — сукупність ***явних фактів****, які зберігаються в пам'яті системи в явному вигляді, R — сукупність* ***правил виведення****, які дозволяють на основі відомих знань набувати нових знань, P —* ***сукупність процедур****, які визначають, яким чином слід застосовувати правила виведення.*

Базою знань *інтелектуальної системи називатимемо сукупність усіх знань, що зберігаються в пам'яті системи.*

У базах знань прийнято розрізняти ***екстенсіональну*** та ***інтенсіональну*** частини.

Екстенсіональною частиною *бази знань називається сукупність усіх явних фактів,* ***інтенсіональною частиною*** *— сукупність усіх правил виведення та процедур, за допомогою яких з існуючих фактів можна виводити нові твердження.*

Приклад. Розглянемо таку базу знань.

Ф1. *Заєць їсть траву.*
Ф2. *Вовк їсть м'ясо.*
Ф3. *Заєць є м'ясом.*
П1. *Якщо А є м'ясом, а В їсть м'ясо, то В їсть А.*
П2. *Якщо А їсть м'ясо, то А є хижаком.*
П3. *Якщо А їсть В, то В є жертвою.*
П4. *Якщо хижак вимирає, жертва швидко розмножується.*
П5. *Якщо жертва вимирає, хижак вимирає.*

Даний приклад є фрагментом неформалізованого опису моделі Лотки-Вольтерра, добре відомої в екології. Літерами **Ф** позначені **явні факти**, які відображають елементарні знання. Їх безпосередній аналіз дозволяє експертній системі відповідати на найпростіші запитання, такі, як *"Вовк їсть м'ясо?"*. Для позитивної відповіді на це та подібні йому запитання досить просто знайти відповідний факт в екстенсіональній частині бази знань.

Принципово іншим є запитання типу *"Вовк є хижаком?"*. Відповідь на нього на основі простого пошуку знайти неможливо: такого явного факту в екстенсіональній частині бази знань немає. Тому для відповіді на це запитання необхідно застосовувати правила виведення, які дозволяють на основі існуючих явних фактів набувати нових знань, тобто нових фактів. Такі правила позначені у нашому прикладі літерою **П**. Наприклад, для відповіді на поставлене запитання досить знання факту **Ф2** і правила **П2**.

Безумовно, до бази знань можна було б відразу включити твердження *"Вовк є хижаком"*. Але легко побачити: якщо, крім вовків, розглядати інших хижаків (ведмедів, лисиць та ін.), то безпосередньо включати до бази знань явні факти, що вони є хижаками, недоцільно. Все, що може бути виведене логічним шляхом, як правило, не варто оголошувати фактами.

Надалі ми називатимемо явні факти просто фактами, якщо це не викликатиме непорозумінь.

Нарешті, **процедури** визначають, яким саме чином слід застосовувати правила. Часто ці процедури є складовою мови, якою програмується експертна система (це стосується насамперед спеціалізованих мов штучного інтелекту, таких як **Лісп, Пролог, Пленер**). Якщо ж таких процедур недостатньо, програміст має сам подбати про їх написання.

4.3. Експертні системи

База знань, наведена в п. 4.2, може лягти в основу ***експертної системи***, тобто інтелектуалізованої інформаційної системи, яка здійснює дедуктивне виведення на основі наявних знань. Існують різні визначення експертних систем, основна суть яких зводиться до такого.

Експертні системи *— це інтелектуальні програмні засоби, здатні у ході діалогу з людиною одержувати, накопичувати та коригувати знання із заданої предметної галузі, виводити нові знання, розв'язувати на основі цих знань практичні задачі та пояснювати хід їх розв'язку. Експертні системи акумулюють знання* ***експертів*** *— провідних спеціалістів у даній предметній галузі. В основі роботи експертних систем лежить дедуктивне виведення нових тверджень з існуючих. Типове застосування експертних систем — консультації для фахівців середньої кваліфікації і неспеціалістів у тій галузі, для якої вона розроблена.*

Необхідно усвідомити, що створити універсальну експертну систему практично неможливо. По-перше, це пов'язано з прокляттям розмірності — зовнішній світ надто багатий і різноманітний. По-друге, експертна система повинна акумулювати знання людей-експертів, а "універсальних" експертів не існує і не може існувати. По-третє, жодне логічне виведення не може замінити інтуїцію та досвід експерта.

Можна створювати лише експертні системи, які належать до конкретної предметної галузі. Остання передбачає лімітований набір явищ і понять з певної сфери людської діяльності та обмежене коло задач, які вирішуються в цій галузі. Існує чимало експертних систем у таких сферах життєдіяльності, як медична і технічна діагностика, пошук корисних копалин, юриспруденція, аналіз інвестицій і комерційних ризиків і т. п. [58, 94, 129, 145, 168, 175, 212, 216, 223, 227, 235, 266, 280, 281, 305].

У створенні експертних систем повинні брати участь фахівці як мінімум двох категорій:

- ***експерт***, що є висококваліфікованим фахівцем у даній предметній галузі і знання якого потрібно передати експертній системі;
- ***інженер знань***, завдання якого — формалізувати знання експерта і привести їх до вигляду, придатного для занесення до бази знань.

Дуже серйозна проблема у даному разі пов'язана з тим, що знання експерта важко формалізувати. Експерт часто не може сформулювати свої знання в явному вигляді, робить правильні висновки, але не може пояснити, **як саме** він їх робить. Якщо ж експерт і формулює певні правила ви-

ведення, вони далеко не завжди відзначаються точністю та адекватністю. З цього випливає ряд інших труднощів, які так чи інакше виявляють себе на етапі формалізації знань або застосування готових експертних систем.

4.4. Дані та знання

Відомо, що сучасний етап розвитку інформатики характеризується еволюцією моделей даних у напрямі переходу від традиційних (реляційна, ієрархічна, мережна) до моделей знань [131, 268, 270]. Знання, як будь-яку інформацію, можна вважати даними. Але знання є високоорганізованими даними, для яких характерна певна внутрішня структура та розвинуті зв'язки між різними інформаційними одиницями. Іншим принциповим аспектом є те, що інформаційні системи, які ґрунтуються на знаннях, повинні мати можливість отримувати нові знання на основі існуючих. У цьому ми вже пересвідчилися на прикладах дедуктивного виведення нових знань з явних фактів.

Виокреслюють основні властивості знань, які відрізняють їх від традиційних даних (див. п. 4.7). Але спочатку детальніше розглянемо особливості знань, пов'язані з дедуктивним логічним виведенням та відношеннями між різними інформаційними одиницями.

Як уже зазначалося в п. 4.2, база знань складається з екстенсіональної та інтенсіональної частин. Перша містить факти, які запам'ятовуються явно, а друга — правила, які дозволяють отримувати нові факти. Можна вважати, що саме наявність розвинутої інтенсіональної частини є однією з основних рис, які відрізняють знання від традиційних даних. Взагалі, ***інтенсіональним відношенням***, яке описує базу даних, часто називають правило або сукупність правил, яким підпорядковується кожний запис у базі даних.

Наведемо ще один *приклад*.

Уявимо собі базу даних методиста деканату, що містить дані про те, які курси прослухав кожний студент. Відомо, що жоден студент не має права слухати курс *"Бази даних"*, якщо він не прослухав курсу *"Комп'ютерні технології"*. Нехай у базі даних зберігається інформація про Іванова, Петрова та Сидорова, які прослухали обидва курси.

Інтенсіональна база даних (або її вже можна назвати базою знань) може мати такий вигляд:

Екстенсіональна частина:

Прізвище	Дисципліна
Іванов	Бази даних
Петров	Бази даних
Сидоров	Бази даних

***Правило**: Якщо Прослухав (Х, Бази даних), то Прослухав (Х, Комп'ютерні технології).*

Наявність такого правила дозволяє скоротити базу знань: твердження, істинність яких можна встановити за допомогою правил, не обов'язково запам'ятовувати в явному вигляді.

Резюмуючи сказане, можна дійти висновку, що пошук в інтенсіональній базі знань складається з двох етапів:

- пошук потрібного факту в екстенсіональній частині;
- дедуктивне виведення факту на основі правил інтенсіональної частини.

У принципі реалізовувати інтенсіональні правила можна навіть у рамках традиційної реляційної моделі даних шляхом програмування відповідних запитів. На сучасному етапі розвиваються моделі і технології, які дозволяють полегшити проектування та підтримку баз знань, якщо розглядати їх як інтенсіональні бази даних. Зокрема, розвивається теорія ***дедуктивних баз даних*** [99]. Запропонована мова **Дейталог** [10, 99, 273], яка об'єднує підходи, характерна для логічного програмування (див. розд. 7) і дедуктивних баз даних.

4.5. Зв'язки між інформаційними одиницями

У базі знань можуть зберігатися відомості про різні об'єкти, поняття, процеси і т. п., тобто відповідні описи, які ми неформально назвемо ***інформаційними одиницями***.

Опис, який утворює інформаційну одиницю*, розглядається як деяка абстракція реальної сутності, що існує в дійсному світі.*

Це означає, що дана абстракція описує певні риси реальної сутності. Звичайно, бажано, щоб абстракція описувала найважливіші з них.

У реальному світі все тісно взаємопов'язане. Відповідні зв'язки повинні знайти адекватне відображення і в базі знань. Якщо ми виокремлюємо деякі типові зв'язки, ми можемо впровадити операції, пов'язані з цими зв'язками, і запрограмувати їх на рівні системи керування базами знань (СКБЗ). Тоді у всіх базах знань, які були створені на основі даної СКБЗ або підтримуються нею, відповідні зв'язки інтерпретуватимуться однотипно.

Існує велика група зв'язків, за допомогою яких можна описувати просторові відношення ("знизу", "зверху", "поруч", "між" і т. п.); часові відношення ("раніше", "пізніше", "під час", "одночасно" і т. п.), причинно-наслідкові відношення тощо. Ці відношення є спільними для всіх предметних областей. Логічні системи, що ґрунтуються на цих відношеннях, прийнято називати ***псевдофізичними логіками***. Відомі, зокрема, просторова логіка, часова логіка, каузальна логіка і т. п. Використання псевдофізичних логік (так само, як і більш загальних предикатів, таких, як узагальнення та агрегація) дає можливість інтелектуальній системі поповнювати первинні описи.

Так, якщо інтелектуальна система сприймає текст: *"У 1985 році почалася перебудова. У 1991 році була проголошена незалежність України"*, врахування часових властивостей, що визначається часовою логікою, дає можливість відповідати на запитання типу: *"Що було раніше: початок перебудови чи проголошення незалежності України?"*

До того ж, для кожної предметної області існують відповідні специфічні відношення.

Основними зв'язками між інформаційними одиницями є ***узагальнення*** та ***агрегація***.

Агрегація дозволяє задавати будову об'єкта та його складових. Наприклад, агрегація дає змогу визначити в базі знань аудиторію як об'єкт, що має вікна, двері і т. п. Вікна, в свою чергу, теж можна розглядати як агрегований об'єкт: вони мають ручки, рами і т. п. Інакше кажучи, відношення *"Має"*, яке описує агрегацію, означає, що певний об'єкт містить у своєму складі деякий інший об'єкт. За цим відношенням закріпилася спеціальна назва **HAS_PART**. По суті, це відношення "ціле — частина".

Таким саме чином ввести і зворотне відношення (*"Є частиною"*, або **IS_PART**).

Узагальнення (відношення *"Є"*) задає **ієрархію класів** ***(екземпляр — клас — надклас)***. Так, можна говорити, що об'єкт **Вася Петров** належить до класу **Студент**. Відповідно інформаційна одиниця, яка описує **Васю Петрова**, може бути описана на основі більш загальної інформаційної одиниці **Студент** (точніше, вона задається як екземпляр інформаційної одиниці **Студент**). Клас **Студент**, у свою чергу, є ***підкласом*** типу **Людина** і т. п. Надклас об'єднує атрибути, або ознаки, спільні для його підкласів; підкласи можуть успадковувати ті чи інші властивості надкласів.

Саме з відношенням *"Є"* пов'язаний один з основних відомих механізмів логічного виведення — механізм ***виведення за успадкуванням*** (інша назва — виведення ***за наслідуванням***). Суть цього механізму можна сформулювати так: якщо деяка умова виконується для всього класу, то вона виконується і для кожного представника цього класу, а також для всіх підкласів цього класу (якщо інше не задано явним чином). Інакше кажучи, екземпляри успадковують властивості класів, підкласи успадковують властивості надкласів.

В усіх системах керування базами знань повинна бути забезпечена належна підтримка успадкування. Розглянемо *приклад* [223].

У базі знань міститься така інформація:

Усі птахи літають.
Ластівка є птахом.
Юкко є ластівкою.

Маємо загальний клас *"Птахи"*, його підклас *"Ластівки"* та об'єкт *"Юкко"*, який є екземпляром класу *"Ластівка"*. На підставі цих знань будь-яка система, що підтримує успадкування, **повинна** зробити висновки, що *Юкко є птахом*; *всі ластівки літають*; *Юкко теж літає*.

Насправді, замість єдиного відношення *"Є"* необхідно розглядати цілу систему споріднених, але не ідентичних відношень. Інакше легко припуститися логічних помилок, подібних до таких:

Юкко є ластівкою.
Ластівка є видом, який вивчається натуралістами.

Отже, *Юкко вивчається натуралістами.* (Або: *Юкко є видом, який вивчається натуралістами.*)

Слід розглядати такі відношення:

- ***"клас — підклас"***, яке є відношенням часткового порядку (як правило, строгого). Відтак, для цього відношення виконуються всі властивості часткового (строгого або нестрогого) порядку, зокрема **транзитивність**;
- ***"екземпляр — клас"***. Екземпляри успадковують властивості своїх класів, але саме відношення "екземпляр — клас" не є транзитивним. Так, якщо *"Ластівка"* є екземпляром класу *"Види, які вивчаються натуралістами"*, а *"Юкко"* є екземпляром класу *"Ластівка"*, то *"Юкко"* у жодному разі не є екземпляром класу *"Види, які вивчаються натуралістами"* (хоча б тому, що *"Юкко"* взагалі не є видом).

Крім того, необхідно чітко розрізняти поняття ***"клас"*** і ***"множина", "належність до класу"*** і ***"належність до множини"***. Поняття "клас" має відкритий, інтенсіональний характер; поняття "множина" — закритий, екстенсіональний. Множина передбачає певну сукупність об'єктів. З іншого боку, клас характеризує набір властивостей, які мають бути притаманні екземплярам цього класу. Клас об'єднує однотипні елементи, множина — необов'язково.

Варто виокремити такі типи множин:

- ***екстенсіонал класу*** — множина екземплярів класу; можна розглядати ті чи інші підмножини множини екземплярів класу;
- ***агрегатна множина*** — множина, яка складається з різнотипних елементів.

Звідси — відношення "елемент — множина" належить до агрегації, а не до узагальнення. Це відношення, так само, як і відношення "екземпляр — клас", не є транзитивним.

Повернемося до останнього парадоксу:

Юкко є ластівкою.

Ластівка є видом, який вивчається натуралістами.

Отже, *Юкко вивчається натуралістами.* (Або: *Юкко є видом, який вивчається натуралістами.*)

Якщо переформулювати ці твердження коректніше, їх можна записати у вигляді:

Юкко є екземпляром класу "Ластівка".

Ластівка є екземпляром класу "Види, які вивчаються натуралістами".

Або:

Юкко є екземпляром класу "Ластівка".

Ластівка є елементом множини "Види, які вивчаються натуралістами".

В обох випадках транзитивності немає і висновки *"Юкко вивчається натуралістами"* та *"Юкко є видом, який вивчається натуралістами"* стають невірними і з формальної точки зору.

Між узагальненням та агрегацією існує тісний взаємозв'язок. У [31] розглядається поняття ***"канонічна форма складної системи"***. Згідно з ним

складні системи можуть бути описані у вигляді певної канонічної форми, яка являє собою композицію двох ієрархій — ***ієрархії класів*** та ***ієрархії об'єктів***.

Ієрархія класів *задається відношенням узагальнення, ієрархія об'єктів — відношенням агрегації.*

Дійсно, з одного боку, такі об'єкти, як "ручки", "рами"та інші, утворюють новий об'єкт — "вікно". "Вікна", "двері" тощо — це об'єкти, які утворюють ще один новий об'єкт: "аудиторію". Таким чином, можна розглядати певну ієрархію об'єктів. З іншого ж боку, всі ці об'єкти є екземплярами своїх класів. Так, конкретне вікно належить до класу "Вікна", клас "Вікна" є підкласом класу "Дерев'яні вироби" і т. п.

Відношення "клас — підклас" є характерним для ієрархії класів, відношення "елемент — множина" та "підмножина — множина" — до ієрархії об'єктів, а відношення "екземпляр — клас" об'єднує обидві ієрархії.

4.6. Проблема винятків

З успадкуванням пов'язана дуже серйозна проблема — ***проблема винятків***. Вона полягає в тому, що деякі підкласи можуть не успадковувати ті чи інші властивості надкласів. Інакше кажучи, характерні риси класу успадковуються всіма його підкласами, **крім деяких**.

Нехай відомо, що *літають всі птахи, крім пінгвінів* (існують деякі інші види птахів, які не літають, але для наших цілей це не має суттєвого значення). Якби це твердження відразу потрапило до бази знань саме в такому вигляді, особливих проблем не виникало б (хоча і в цьому випадку слід було б передбачити належну обробку винятків).

Але, як було зазначено раніше, експерт не завжди може сформулювати свої знання в явному вигляді. Зокрема, він може не знати або не пам'ятати всіх винятків. Тому він може спочатку включити до бази знань твердження про те, що *всі птахи літають*, а **потім** пригадати, що *пінгвіни не літають*, і додати це до бази знань.

У результаті ми могли б отримати базу знань, подібну до такої:

Усі птахи літають.
Ластівка є птахом.
Юкко є ластівкою.
Пінгвін є птахом.
Пінгвіни не літають.
Бакс є пінгвіном.

Якби три останні твердження **не були** включені до бази знань, система просто дійшла б хибного висновку, що *Бакс літає*. Але включення даних відомостей до бази знань ще більше ускладнює ситуацію. **Система знань стає суперечливою**: з одного боку, система повинна дійти висновку, що Бакс літає, а з іншого — що Бакс не літає. У даному разі кажуть про ***втрату монотонності*** дедуктивної системи [167].

Система дедуктивного виведення *називається* ***монотонною****, якщо виконується така властивість: якщо з набору тверджень* $(q_1, \ldots, q_n)$ *випливає твердження* v*, то* v *випливає і з набору тверджень* $(q_1, \ldots, q_n, r)$.

Інакше кажучи, в монотонній теорії додавання нових фактів і правил не повинно впливати на істинність висновків, які могли бути отримані без них.

Додавання ж винятків до наявної бази знань може порушити монотонність. Існує багато підходів до вирішення цієї проблеми. Розвиваються спеціальні ***немонотонні логіки***: Рейтера, Мак-Дермотта та ін. [10, 167, 223]. Проте існують досить прості практичні прийоми. Наприклад, список виключень можна підтримувати явним чином. Інший корисний прийом сформульований в [320]: **у разі виникнення суперечностей підклас успадковує відповідну властивість лише від найближчого попередника, тобто від класу, найближчого до нього в ієрархії класів**.

Далі буде показано, як проблема винятків може вирішуватись в рамках продукційних систем і семантичних мереж.

4.7. Властивості знань

У [129] сформульовані такі особливості знань, які відрізняють їх від традиційних даних:

- ***внутрішня інтерпретованість:*** кожна інформаційна одиниця повинна мати унікальне ім'я, за яким інформаційна система її знаходить, а також відповідає на запити, в яких це ім'я згадується;
- ***структурованість:*** знання повинні мати гнучку структуру; одні інформаційні одиниці можуть включатися до складу інших (відношення типу "частина — ціле", "елемент — клас");
- ***зв'язність:*** в інформаційній системі повинна бути передбачена можливість встановлення різних типів зв'язків між різними інформаційними одиницями (причинно-наслідкові, просторові та ін.);
- ***семантична метрика:*** на множині інформаційних одиниць корисно задавати відношення, які характеризують ситуаційну близькість цих одиниць;
- ***активність:*** виконання програм в інтелектуальній системі повинно ініціюватися поточним станом бази знань.

Часто виокремлюють інші властивості знань, наприклад ***шкальованість***, яка означає, що формально неоднакові поняття насправді відображаються на одній і тій самій ***шкалі понять***, різні точки якої відповідають інтенсивності прояву одного й того самого фактора. Так, температура може бути високою або низькою, і це породжує такі поняття, як "холодно", "тепло", "гаряче".

Для формалізації задання знань у системах штучного інтелекту необхідні певні формалізми, які називаються ***моделями задання знань***, або просто ***моделями знань***.

Моделлю знань *називається фіксована система формалізмів (понять і правил), відповідно до яких інтелектуальна система подає знання в своїй пам'яті та здійснює операції над ними.*

Моделі задання знань необхідні:

- для створення спеціальних мов описів знань і маніпулювання ними;
- для формалізації процедур зіставлення нових знань з уже існуючими;
- для формалізації механізмів логічного виведення.

Як уже зазначалося раніше, найрозвинутішими на сучасному етапі є моделі задання знань, які ґрунтуються на вербально-дедуктивній парадигмі. Найвідомішими з них є чотири класи моделей:

- ***семантичні мережі;***
- ***фреймові;***
- ***логічні;***
- ***продукційні.***

Вербально-дедуктивні моделі задання знань мають багато спільних рис. Отже, можна вважати, що всі вони мають єдину концептуальну основу.

4.8. Неоднорідність знань. Області і рівні знань

Для успішного здійснення операцій зі знаннями необхідно відокремлювати в базі знань певні фрагменти, які називаються ***областями знань***. Ці області знань повинні бути відносно незалежними між собою, що означає таке:

- зміни в одній області знань не повинні приводити до суттєвих змін в інших областях;
- вирішення складної задачі можна, як правило, звести до підзадач таким чином, що для вирішення кожної з цих підзадач достатньо знань з однієї області.

Такий розподіл є важливим для полегшення проектування і використання бази знань. Зокрема, різні області знань можуть проектуватися незалежно одна від одної.

Виходячи з постановки задачі можна по-різному розділяти знання на області. Так, для експертних систем, які ведуть діалог з користувачем мовою, наближеною до природної, можна виокремити такі області знань [216]:

- ***предметна область,*** яка містить знання про конкретну предметну галузь, в якій працює експертна система;
- ***область мови,*** яка містить знання про мову, якою ведеться діалог;
- ***область системи,*** яка містить знання експертної системи про власні можливості;
- ***область користувача,*** яка містить знання про користувача. Наявність їх дозволяє враховувати індивідуальні особливості кожного користувача. Наприклад, пояснення користувачеві можуть надаватися залежно від рівня його підготовленості;
- ***область діалогу,*** яка містить знання про мету діалогу, а також про форми та методи його організації.

Знання можуть різнитися за ***рівнем задання*** і ***рівнем детальності***. За ***рівнями задання*** розрізняють знання нульового рівня (конкретні і абстракт-

ні знання) і знання вищих рівнів знання про знання ***(метазнання)***. Число рівнів може бути продовжено.

У разі класифікації знань за ***рівнями детальності*** до уваги береться ступінь деталізації знань. Кількість рівнів детальності залежить від специфіки задачі, обсягу знань і моделі їх задання.

Нехай користувач запитує в експертної системи, як здійснити певну операцію. У разі отримання відповіді на найвищому рівні детальності система видає **загальний план**. Якщо цей загальний план не є достатнім, система може спуститися на нижчий рівень і розписати операції детальніше.

4.9. База знань як об'єднання простіших одиниць

База знань є кон'юнкцією більш простих тверджень. Можна вважати, що кожне з них відповідає окремому реченню природної мови. Подібні твердження часто називають ***концептуальними одиницями***. Останні можуть бути або фактами, або правилами виведення.

Якщо концептуальна одиниця являє собою факт, вона може бути описана певним ***предикатом***, тобто логічною функцією, що залежить від тієї чи іншої кількості змінних і може приймати одне з двох можливих значень: *істинність* або *хибність*. Конкретні твердження утворюються з відповідних предикатів шляхом підстановки конкретних значень аргументів.

При цьому для опису одного й того самого твердження можна використовувати різні предикати.

Так, три твердження:

7 + 5 = 12
8 + 9 = 17
3 + 0 = 7

відповідають одному предикатові

Плюс (*a*, *b*, *c*).

Перші дві підстановки конкретних значень замість змінних ***a***, ***b***, ***c*** породжують істинні твердження, третя — хибне. Формально ці твердження будуть мати запис:

Плюс (7, 5, 12)
Плюс (8, 9, 17)
Плюс (3, 0, 7).

Але можна розглядати інший, загальніший предикат:

Операція (назва_операції, *a*, *b*, *c*),

який можна використовувати для запису тверджень не тільки про додавання, але й про будь-яку іншу бінарну операцію: множення, ділення і т. п. Зокрема, можливим є запис:

Операція (Плюс, 7, 5, 12).

Який тип розглянутих предикатів слід обирати, залежить від проектувальника бази знань і специфіки конкретної задачі.

4.10. Бінарні предикати і тріада "об'єкт—атрибут—значення"

Концептуальна одиниця може бути достатньо складною. Тому концептуальну одиницю, що є фактом, часто зручно розглядати як кон'юнкцію елементарніших тверджень, а саме — ***бінарних фактів***, кожний з яких описується ***бінарним предикатом***, тобто предикатом, який залежить від двох змінних. Якщо формалізувати правила об'єднання бінарних предикатів у складніші одиниці, на основі бінарних предикатів можуть бути сконструйовані які завгодно складні твердження та системи знань. Такий підхід, зокрема, розвивається в [167].

Розглянемо *приклад*.

Маємо твердження *"Студент Іванов отримав п'ятірку на іспиті зі штучного інтелекту"*.

Якщо переписати цю фразу у вигляді кон'юнкції бінарних фактів, запис може мати вигляд:

Є (Іванов, Студент)

Є (Ісп_Шт_Інт, Іспит)

Ісп_Шт_Інт (Іванов, 5)

Предикат **Ісп_Шт_Інт** був введений для задання зв'язку між різними інформаційними одиницями.

У вигляді бінарного можна переписати й ***унарний предикат***, тобто предикат, який залежить від однієї змінної.

Наприклад, предикат

Птах (X),

який означає, що **X** є птахом, можна переписати в еквівалентному вигляді

Є (X, Птах).

Якщо певний унарний предикат описує **властивість** об'єкта, наприклад:

Літає (X),

її можна переписати у вигляді

Літає (X, так)

або у вигляді

Є (X, Кл_Літ),

де **Кл_Літ** — клас об'єктів, які літають.

Бінарним предикатам і бінарним фактам безпосередньо відповідає ***тріада "об'єкт — атрибут — значення"***. Об'єкт у цій тріаді — це, як правило, назва деякої інформаційної одиниці, атрибут — назва певної ознаки, а значення — конкретне значення цієї ознаки для даного об'єкта.

Тріада "об'єкт — атрибут — значення" є просто іншою формою запису бінарного факту. Так, якщо переписати наш приклад у вигляді сукупності згаданої тріади, запис матиме вигляд:

Іванов — Є — Студент

Ісп_Шт_Інт — Є — Іспит

Іванов — Ісп_Шт_Інт_ — 5.

Остання тріада цього прикладу задає зв'язок між різними об'єктами.

4.11. Проблема неточних і неповних знань

Раніше ми розглядали проблеми, які необхідно було вирішувати при проектуванні та розробці баз знань. Серед інших проблем можна відмітити такі.

Знання можуть бути **неповними**. Це означає, що для доведення або спростування певного твердження може не вистачати інформації. У багатьох системах логічного виведення прийнято ***постулат замкненості світу***. Це означає, що на запит про істинність деякого твердження система відповідає *"так"* тоді і тільки тоді, коли його можна довести; якщо ж довести це твердження неможливо, система відповідає *"ні"*. Водночас *"неможливо довести через нестачу інформації"* і *"доведено, що ні"* — це зовсім не одне й те саме. З огляду на це бажано, щоб експертна система запитувала у користувача про факти, яких не вистачає.

Знання можуть бути **недостовірними**. Наприклад, на результат виконання продукції можуть впливати випадкові чинники (об'єктивна невизначеність) або ж експерт може бути не зовсім упевненим у деякому факті чи правилі виведення (суб'єктивна невизначеність).

Ненадійність знань і недостовірність наявних фактів обов'язково повинні враховуватися в процесі логічних побудов. Звичайно, можна було б просто відкидати факти та правила виведення, які викликають сумнів, але тоді довелося б відмовитися від цінної інформації. Тому необхідно розвивати процедури, які дозволяють здійснювати логічні побудови при недостовірних даних, і використовувати ці процедури в експертних системах. Необхідно враховувати **модальності**, а саме: необхідність або можливість того чи іншого факту, ставлення суб'єкта до деякого твердження і т. п. Крім того, в таких системах часто доводиться мати справу з неточно визначеними, нечіткими поняттями, такими, як *"великий"*, *"маленький"* тощо.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте вербально-дедуктивний підхід до опису поняття "знання". Чи є він єдино можливим?
2. Наведіть інтуїтивне визначення поняття "знання".
3. Наведіть вербально-дедуктивне визначення знань. Що означають поняття "факти", "правила виведення" та "процедури" в цьому визначенні?
4. Що означають екстенсіональна та інтенсіональна частини бази знань?
5. Наведіть визначення експертної системи.
6. Охарактеризуйте поняття "інформаційна одиниця".
7. Які існують зв'язки між інформаційними одиницями?
8. Що таке псевдофізичні логіки?
9. Охарактеризуйте поняття "агрегація".
10. Опишіть поняття "узагальнення".
11. Опишіть поняття "клас" та "екземпляр класу".
12. Охарактеризуйте принцип логічного виведення за успадкуванням.
13. Наведіть власні приклади ієрархії класів.

14. Чим відрізняється відношення "екземпляр — клас" від відношення "підклас — клас"?
15. Охарактеризуйте відношення "підклас — клас" як відношення часткового порядку.
16. Чим відрізняється відношення "екземпляр — клас" від відношення "елемент — множина"?
17. Охарактеризуйте проблему винятків. Як вона пов'язана з монотонністю логічного виведення?
18. Що таке канонічна форма складної системи (за Бучем)?
19. Перелічіть основні властивості знань.
20. Перелічіть моделі задання знань.
21. Перелічіть області знань, характерні для діалогових експертних систем.
22. У чому полягає необхідність у виокремленні областей і рівнів знань?
23. Опишіть поняття "концептуальна одиниця".
24. Яким чином можна розкласти деякий складний предикат на бінарні предикати? Наведіть приклади.
25. Яким чином можна перетворити унарний предикат на бінарний? Наведіть приклади.
26. Охарактеризуйте поняття "об'єкт — атрибут — значення".
27. Що означає "постулат замкненості світу"?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Охарактеризуйте основні проблеми, які доводиться вирішувати при розробці експертних систем і баз знань. Наведіть приклади.
2. Чому знання експерта важко формалізувати, тобто подати їх у вигляді, придатному для занесення до бази знань?
3. Наведіть приклад бази знань з довільної предметної області, що включає до свого складу факти і правила.
4. Наведіть власний приклад бази даних, яку можна розкласти на інтенсіональну та екстенсіональну частини.

Розділ 5
СЕМАНТИЧНІ МЕРЕЖІ

Ти мені, я тобі.
Назва кінофільму

5.1. Визначення та класифікація семантичних мереж

Семантичні мережі є зручним способом графічного подання знань. Особливий акцент при цьому робиться на зв'язках між різними інформаційними одиницями та між різними фрагментами знань. Важливим

є те, що вся інформація про дане поняття групується навколо вузла мережі, який відповідає цьому поняттю.

Семантичну мережу *можна неформально уявляти у вигляді графу, вершини якого, як правило, позначають об'єкти предметної області, а дуги відповідають зв'язкам між ними.*

Це визначення є неформальним і орієнтоване скоріше на людське розуміння, ніж на проектування і створення систем штучного інтелекту. Остання задача вимагає більш формальних визначень [129].

Формально ***семантична мережа визначається*** *як набір* $< I, C_1, C_2, ..., C_n, \Gamma >$. *Тут* I *— множина інформаційних одиниць,* $C_1, C_2, ..., C_n$ *— типи зв'язків між інформаційними одиницями,* Γ *— відображення, що задає зв'язки між інформаційними одиницями.*

На відміну від неформального визначення, останнє вимагає жорсткої фіксації схеми бази знань, тобто набору можливих інформаційних одиниць і типів можливих зв'язків.

Існує значна кількість моделей знань на основі семантичних мереж; історично першою була модель Квілліана [223].

Наведемо *приклад*.

Твердження *"Студент Іванов отримав 5 на іспиті зі штучного інтелекту"* може бути зображено у вигляді такої семантичної мережі:

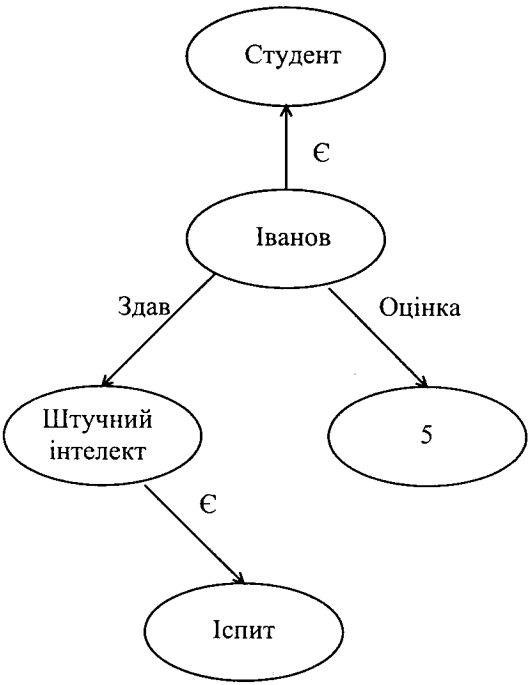


Рис. 5.1. Приклад семантичної мережі

Таке подання, зрозуміло, не є єдино можливим. У п. 5.5 обговорюється, які переваги і недоліки можуть бути пов'язані з різними формами задання баз знань у вигляді семантичних мереж.

Семантичні мережі є найбільш широким класом мережних моделей, в яких поєднані різні типи зв'язків. Часткові випадки семантичних мереж відповідають спеціальним типам зв'язків. Виокремлюють, зокрема, *класифікуючі мережі*, *функціональні мережі* та *сценарії*.

Класифікуючі мережі дозволяють задавати відношення ієрархії між інформаційними одиницями.

Функціональні мережі характеризуються наявністю функціональних відношень, які дозволяють описувати процедури обчислень одних інформаційних одиниць через інші. Їх часто називають *обчислювальними моделями*, оскільки вони дозволяють описувати процедури обчислення одних інформаційних одиниць через інші. Функціональним мережам присвячена монографія [253], гарний опис функціональних мереж можна знайти також у [221].

У *сценаріях* використовуються відношення типу "причина — наслідок", "дія", "засіб дії" та ін.

5.2. Семантичні мережі в пам'яті людини

Є ряд свідчень на користь того, що знання в людській пам'яті зберігаються у вигляді структур, які нагадують семантичні мережі. Про це свідчать, зокрема, результати дослідів, що проводилися школою Жана Піаже [187].

Дівчинці трьох з половиною років показали квадрат і попросили намалювати його. Спочатку вона намалювала фігуру, показану на рис. 5.2, що зовсім не дивно.

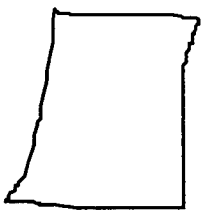


Рис. 5.2. Перше зображення квадрата

Але потім вона намалювала зовсім іншу фігуру:

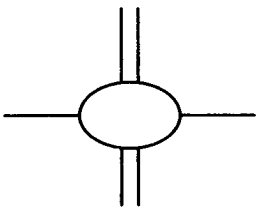


Рис. 5.3. Друге зображення квадрата

Коли дівчинку попросили пояснити свій малюнок, вона вказала на три елементи: "жорсткі речі", "речі, що йдуть вгору — вниз" і "боковинки". Коли вона показала на початковому квадраті, що малося на увазі, з'ясувалося, що "жорсткі речі" відповідають кутам квадрата, "боковинки" — горизонтальним сторонам, а "речі, що йдуть вгору — вниз" — вертикальним.

Жан Піаже та його учні дають цьому таке пояснення. Людина зберігає зображення в своїй пам'яті не так, як воно сприймається органами чуттів, а у вигляді структурного опису. Так, семантична мережа, що задає структурний опис квадрата, могла б мати вигляд:

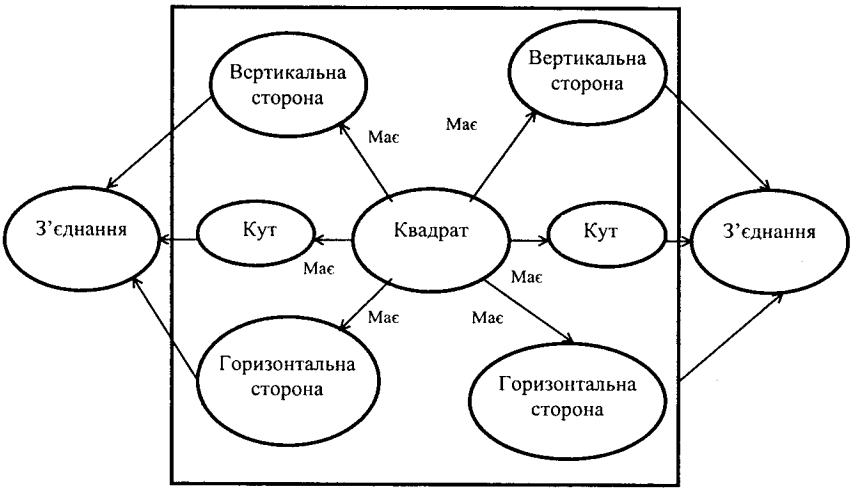


Рис. 5.4. Семантична мережа опису квадрата

Легко побачити схожість цієї структури (особливо фрагмента, обведеного рамкою) з малюнком, зробленим дівчинкою. Очевидно, коли дівчинка робила другий малюнок, вона пропустила стадію реконструкції зображення і намалювала квадрат у вигляді структури, яка збереглася в її пам'яті.

5.3. Трирівнева архітектура семантичних мереж

Семантичні мережі можна розглядати на трьох рівнях ієрархії (див. рис. 5.5) [167, 223].

Повній базі знань відповідає вся семантична мережа; окремій концептуальній одиниці (окремому твердженню або опису окремого поняття) — *концептуальний граф*, що є підмножиною семантичної мережі, яку прийнято розглядати як структуру, що утворюється в результаті композиції окремих концептуальних графів. Запропоновані певні правила, які дозволяють формалізувати таку композицію [167, 325].

Семантична мережа, зображена на рис. 5.1, відповідає окремому реченню. Тому вона фактично є концептуальним графом, який може бути складовою частиною складнішої семантичної мережі.

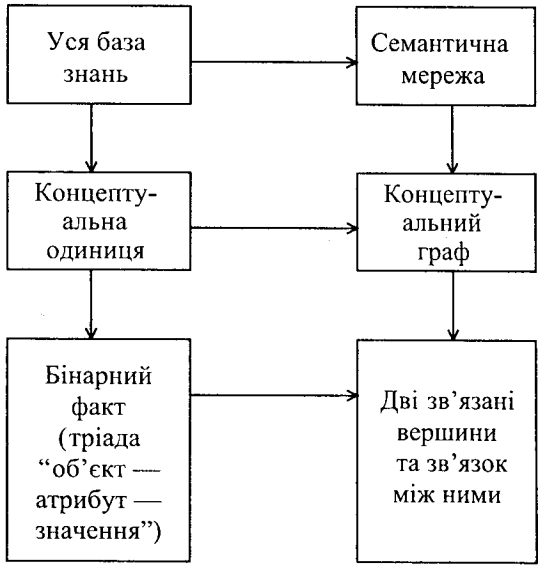


Рис. 5.5. Трирівнева архітектура семантичних мереж

Найнижчому рівню (тріаді "об'єкт — атрибут — значення") відповідають два зв'язані вузли семантичної мережі разом із зв'язком між ними. Найчастіше атрибуту з цієї тріади відповідає зв'язок семантичної мережі.

5.4. Асиміляція нових знань на основі семантичних мереж

Формалізми, які можна застосовувати для "приєднання" нового концептуального графу, що відповідає новому твердженню, до вже існуючої семантичної мережі, можливо використовувати для вирішення на основі мережних моделей дуже важливої задачі — **розуміння інтелектуальною системою нової інформації**.

Ми уже говорили в п. 3.6, що розуміння нової інформації можна уявляти як переклад цієї інформації на внутрішню мову системи шляхом зіставлення зі знаннями системи та формування вторинного опису у термінах знань системи. Цей процес прийнято називати *асиміляцією* нової інформації.

Асиміляція може бути ***повною*** і ***модальною***. За повної асиміляції система сприймає нову інформацію як істинну і просто інтегрує її до бази знань у вигляді сукупності нових фактів. За модальної асиміляції система включає нові факти до своєї бази знань, але з певними помітками, які відображають джерело знань, міру згоди і т. п. Йдеться про те, що система розуміє інформацію і може погоджуватися з нею або ні.

Приклад. Оператор Василь вводить до інтелектуальної системи факт *"Дніпро впадає в Азовське море"*. За повної асиміляції система просто

перекладає цей факт на свою внутрішню мову і заносить його до своєї бази знань. Надалі він може бути використаний у процесі логічного виведення поряд з іншими фактами.

За модальної асиміляції система може сформувати такі знання, як: *"Василь вважає, що Дніпро впадає в Азовське море"* або *"Василь вважає, що Дніпро впадає в Азовське море, але це не так"*, або *"Є свідчення на користь того, що Дніпро впадає в Азовське море; ступінь достовірності цього факту дорівнює 0.1"* тощо. Подібні знання можна враховувати при логічному виведенні, але лише при використанні спеціальних методик.

Для кожної моделі задання знань характерні свої процедури асиміляції. Остання на основі семантичних мереж може бути описана таким чином:

- сприйняття нової інформації і формування її вторинного опису у вигляді одного або кількох концептуальних графів або нового фрагмента семантичної мережі;
- підключення сформованих фрагментів до існуючої семантичної мережі на основі тих чи інших формальних процедур.

5.5. Різні способи задання семантичних мереж

Ми уже зазначали, що одні й ті самі твердження можна зображати різними семантичними мережами і концептуальними графами. Повернемося до речення *"Студент Іванов отримав 5 на іспиті зі штучного інтелекту"*. Концептуальний граф для задання цього речення був наведений у п. 5.1. Повторимо його тут.

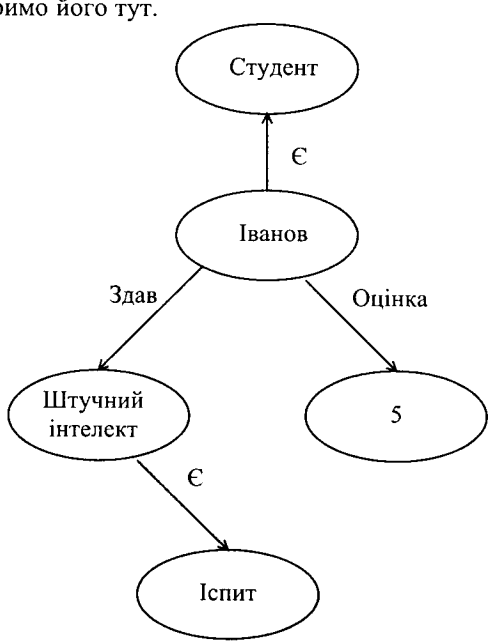


Рис. 5.6. Концептуальний граф у першому варіанті

Наведеному концептуальному графу відповідає такий набір бінарних фактів у формі "об'єкт — атрибут — значення":

Іванов — Є — Студент
Іванов — Здав — Штучний інтелект
Іванов — Оцінка — 5
Штучний інтелект — Є — Іспит

Основна перевага цього рішення — його "природність". Об'єкт "Іванов" відповідає реальній сутності — студентові Іванову.

Проте таке рішення має очевидні вади. Це, зокрема, слабка інтерпретованість дуг "Здає" та "Оцінка", ігнорування того факту, що "Штучний інтелект" — це не тільки "Іспит", але й наукова дисципліна і т. п. Але **найважливіший недолік — це складність задання зв'язків "один до багатьох" і "багато до багатьох".**

Уявіть собі, що в базі знань зберігається ще один факт про цього ж студента, а саме: *"Студент Іванов отримав 2 на іспиті з Ліспу"*. Цей факт може бути описаний аналогічним концептуальним графом, але як об'єднати ці два графи в семантичну мережу? "Очевидний" розв'язок такий:

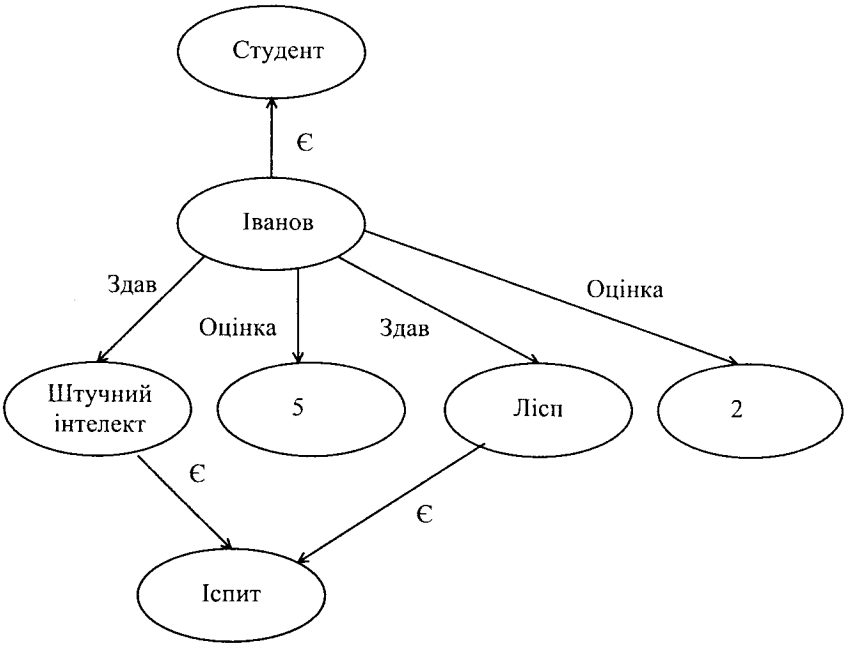


Рис. 5.7. Незадовільне задання зв'язків "один до багатьох"

Він є незадовільним через свою неоднозначність: незрозуміло, яку оцінку з якого предмета отримав Іванов.

Можна запропонувати різні виходи з такого становища. Наприклад, можливе таке рішення:

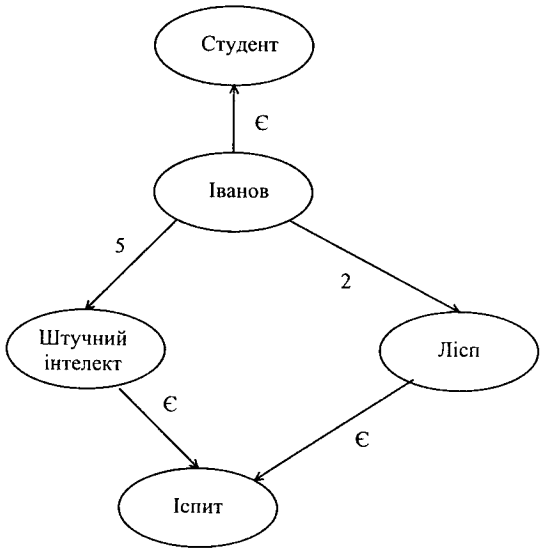


Рис. 5.8. Семантична мережа з неінтерпретованими дугами

Недолік такого рішення — повна неінтерпретованість міток "5" і "2". Вирішити цю проблему можна лише шляхом додавання до відповідних дуг допоміжних зв'язків.

Можна запропонувати розв'язки на основі структурованих семантичних мереж, для яких характерна певна внутрішня структура. Цю структуру можна ввести, наприклад, таким чином:

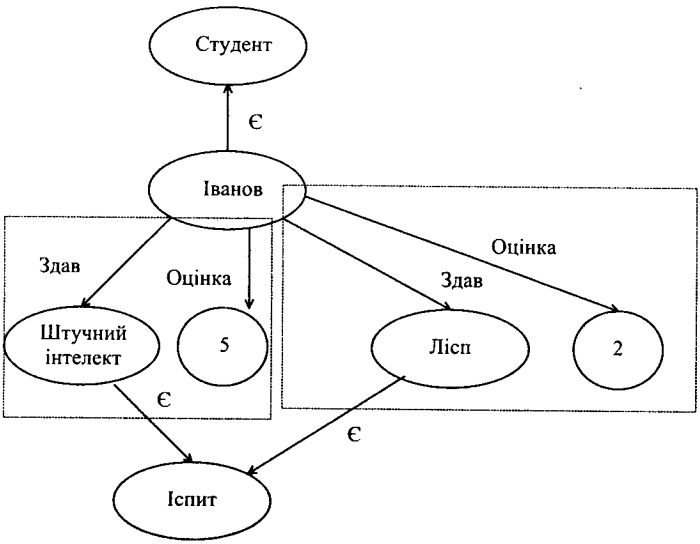


Рис. 5.9. Приклад структурованої семантичної мережі

Рамками обведені фрагменти семантичної мережі, дуги яких об'єднані заданим відношенням (у даному разі відношенням кон'юнкції). Але навряд чи подібна структуризація, яка неминуче ускладнює програмування, є необхідною для таких простих мереж.

Нарешті, можна застосувати такий типовий прийом.

Вводяться додаткові предикати, які відображають відношення *"Студент X здав іспит Y"*. Тоді твердження *"Студент Іванов отримав 5 на іспиті зі штучного інтелекту"* та *"Студент Іванов отримав 2 на іспиті з Ліспу"* можуть бути подані як кон'юнкції таких бінарних фактів:

Іванов — Є — Студент
Исп_1 — Є — Іспит
Исп_1 — Хто_Здав_ — Іванов
Исп_1 — Предмет — Штучний інт
Исп_1 — Оцінка — 5

Іванов — Є — Студент
Исп_2 — Є — Іспит
Исп_2 — Хто_Здав — Іванов
Исп_2 — Предмет — Ліспп
Исп_2 — Оцінка — 2

Зверніть особливу увагу на номери в предикатних іменах: кожному факту здачі іспиту відповідає свій номер; для іншого студента або для іншого предмета предикатне ім'я буде інакшим (у противному разі при описі відношення "багато до багатьох" знову виникне неоднозначність).

Відповідні концептуальні графи матимуть вигляд:

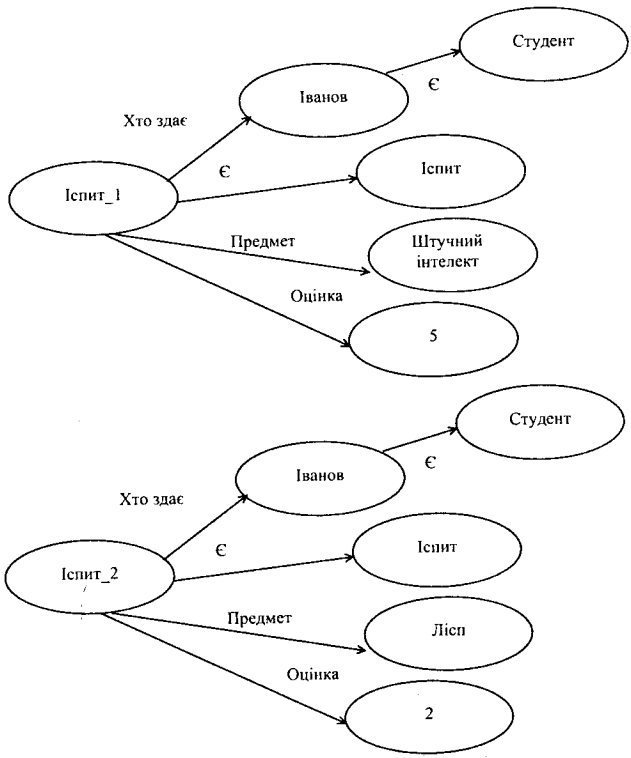


Рис. 5.10. Концептуальні графи з додатковими предикатами

Ці два концептуальні графи можуть бути легко об'єднані в семантичну мережу.

5.6. Логічне виведення на семантичних мережах

Основний механізм логічного виведення на семантичних мережах пов'язаний з відношенням успадкування на основі зв'язків "Є" (див. п. 4.8). Це означає, що властивості, задані для надкласів, передаються донизу за транзитивним ланцюжком відношень "Є".

Як приклад розглянемо набір тверджень

Усі ластівки — птахи.
Юкко — ластівка.

Цьому набору тверджень може відповідати така елементарна семантична мережа:

Рис. 5.11. Наслідування за семантичною мережею

Тоді з семантичної мережі можна вивести твердження: *Юкко — птах.*

Проілюструємо на прикладі аналогічної семантичної мережі механізм обробки винятків, описаний в п. 4.6 (у разі виникнення суперечностей підклас успадковує відповідну властивість лише від найближчого попередника).

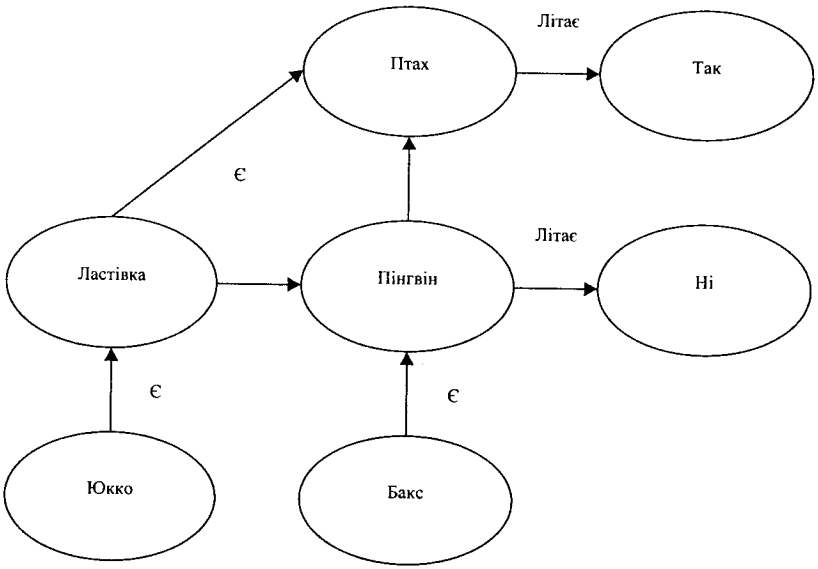


Рис. 5.12. Наслідування та обробка винятків

Відповідь на запитання *"Юкко літає?"* отримується таким чином. Загальна властивість *"літає"*, задана для класу *"Птах"*, передається за ланцюжком відношень "Є" (про можливі порушення транзитивності див. у п. 4.5). У результаті система повинна відповісти: *"Так, Юкко літає"*.

Механізм блокування наслідування вступає в дію при відповіді на запитання *"Бакс літає?"* *"Бакс"* є екземпляром класу *"Пінгвін"*, для якого властивість *"літає"* ***перевизначена***. Це перевизначення блокує передачу відповідної властивості від надкласу *"Птах"*, і *"Бакс"* наслідує її лише від класу *"Пінгвін"* — найближчого класу, для якого властивість *"Літає"* визначена або перевизначена. Звідси — система повинна відповісти: *"Ні, Бакс не літає"*.

У семантичних мережах можна також вводити зв'язки, що задають імплікацію, явно.

Слід відмітити, що формалізм семантичних мереж є зручним для задання знань і не дуже зручним для формалізації логічного виведення. Деякі конкретні методи логічного виведення на семантичних мережах описані в [33, 129, 223]. Багато з них базуються на механізмах дедуктивного виведення, характерних для логічних моделей і продукційних систем (насамперед — метод резолюцій). Ряд методик використовує зіставлення зі зразком — воно характерніше для фреймових моделей (див. розд. 6). Але існують і методики, специфічні для семантичних мереж як графових моделей. В основі цих методик лежить інтерпретація логічного виведення та пошуку потрібної інформації в базі знань, заданій семантичною мережею, як пошуку на графі. Наприклад, у [223] коротко описаний спосіб виведення, що називається ***перехресним пошуком***. Відповідь на запитання формується на основі пошуку шляхів між об'єктами, які фігурують у запитанні, та їх аналізу.

Наприклад, при аналізі мережі, зображеної на рис. 5.12, на запитання *"Що спільного між Баксом та Юкко?"* система може відповісти *"Обидва вони птахи, але різних видів"*.

5.7. Процедурні і розділені семантичні мережі

Чимало досліджень присвячено уніфікації формальних описів семантичних мереж на основі введення єдиної семантики. Зокрема, запропоновані ***процедурні семантичні мережі*** [223].

Процедурна семантична мережа конструюється на основі класу (поняття), а вершини та дуги задані як об'єкти. Процедури визначають такі основні операції над дугами:

- встановлення зв'язку;
- анулювання зв'язку;
- підрахунок кількості вершин, з'єднаних даною дугою;
- перевірка наявності дуги між заданими вершинами.

Ряд процедур визначає основні дії над вершинами, наприклад:

- визначення екземпляра класу;
- анулювання екземпляра;
- підрахунок кількості екземплярів класу;
- перевірка належності екземпляра класу.

Апарат ***розділених семантичних мереж*** [223] призначений для задання кванторів існування і таких кванторів загальності, для яких немає задовільного опису на основі зв'язків "Є". Кожний квантор пов'язується з певною підмережею, тобто вводиться ієрархія підмереж.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте неформальне визначення семантичної мережі.
2. Дайте формалізоване визначення семантичної мережі.
3. Намалюйте семантичну мережу, що описує деякий набір фактів з довільної предметної області.
4. Охарактеризуйте трирівневу архітектуру семантичних мереж.
5. Що таке концептуальний граф?
6. Який елемент семантичної мережі відповідає тріаді "об'єкт — атрибут — значення"?
7. Опишіть процес асиміляції нової інформації в рамках моделі семантичних мереж.
8. Охарактеризуйте повну і модальну асиміляцію нової інформації.
9. Охарактеризуйте відомі підходи до логічного виведення на семантичних мережах. Який з них є основним?
10. Як пов'язане логічне виведення на семантичних мережах з відношенням "Є"?
11. Опишіть механізм блокування наслідування за наявності винятків. Наведіть приклади.
12. Охарактеризуйте поняття перехресного пошуку на семантичних мережах.
13. Що таке розділені семантичні мережі?
14. Що таке процедурні семантичні мережі?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Чи можна хоча б приблизно вважати, що людина запам'ятовує інформацію у вигляді структур, подібних до семантичних мереж?
2. Охарактеризуйте на довільному прикладі різні способи подання концептуальних одиниць за допомогою семантичних мереж. У чому полягають переваги і недоліки цих способів?

Розділ 6
ФРЕЙМОВІ МОДЕЛІ

Вовчика попросили пофарбувати вікна.
Через три години він повертається
і запитує директора школи:
— Іване Петровичу, а рами теж
фарбувати?

Анекдот

6.1. Фрейми та слоти: базові поняття

М. Мінський у 1975 р. запропонував гіпотезу, згідно з якою знання людини групуються в модулі, і назвав ці модулі фреймами. Мінський писав, що коли людина потрапляє в нову ситуацію, вона зіставляє цю ситуацію з тими фреймами, які зберігаються у неї в пам'яті. Саме на теорії фреймів, розробленій М. М. Мінським, і базуються фреймові моделі [181].

***Фреймом** називається структура даних, призначена для опису типових ситуацій або типових понять.*

М. Мінський наводив вужче визначення поняття "фрейм". Він наголошував на тому, що фрейм повинен задавати мінімально можливий опис деякого поняття.

***Фреймом** називається мінімально можливий опис деякої сутності, такий, що подальше скорочення цього опису приводить до втрати цієї сутності.*

Фрейм будь-якого поняття може бути утворений шляхом об'єднання всіх бінарних фактів, пов'язаних з цим поняттям. Формально об'єкт у рамках фреймової моделі описується таким чином:

Ім'я фрейму, ((Атрибут_1, значення_1),
(Атрибут_2, значення_2),
...
(Атрибут_n, значення_n)).

Даним записом підкреслюється, що фрейм — це сукупний опис усіх основних характеристик об'єкта.

*Структури даних, призначені для опису окремих атрибутів у фреймі, називаються **слотами** цього фрейму.*

Як приклад розглянемо поняття *"Студент"*, яке описується відповідним фреймом. Кожний студент може бути охарактеризований такими характеристиками, як прізвище, ім'я, по батькові, факультет, курс. Тоді слоти фрейму *"Студент"* відповідають саме цим характеристикам.

6.2. Конкретизація, ієрархія та наслідування фреймів

Фреймова структура описує узагальнене, родове поняття, тобто групу (клас) однотипних об'єктів з однаковими характеристиками. Конкретні ж об'єкти прийнято називати ***екземплярами*** фреймів.

Опис екземплярів фрейму формується внаслідок *конкретизації* фреймів, що полягає в заповненні слотів конкретними значеннями (інакше кажучи, у визначенні значень атрибутів). Якщо при описі загального фрейму деякі слоти уже заповнені конкретними значеннями, ці значення передаються всім екземплярам цього фрейму. Отже, будь-який студент може бути описаний на основі фрейму *"Студент"*, якщо підставити в слоти конкретні дані.

Для фреймових моделей характерна ***ієрархія понять***. У нашому прикладі фрейм *"Студент"* є похідним від загальнішого фрейму *"Людина"*. Тоді він ***наслідує*** його слоти. У нашому прикладі фрейм *"Людина"* може мати слоти "Прізвище", "Ім'я та по батькові". Тому відповідні слоти в описі фрейму *"Студент"* можна не задавати, а обмежитися лише слотами, специфічними для нього: "Факультет", "Курс". Якщо якісь слоти вже заповнені конкретними значеннями, то ці значення можуть успадковуватися фреймами-нащадками.

6.3. Поповнення первинних описів на основі фреймових моделей

Фрейми виявилися зручним інструментом для опису ситуацій, які сприймаються системою, та для зіставлення цих ситуацій з тими знаннями, що зберігалися в пам'яті (власне, Мінський вважав, що саме в цьому і полягає одне з основних призначень фреймових моделей).

Нагадаємо, що первинним описом ситуації або об'єкта називається їх опис у тому вигляді, в якому вони безпосередньо сприймаються органами чуттів. Ми знаємо, що цей первинний опис, як правило, поповнюється на основі наявних знань. Коли інтелектуальна система сприймає деякий об'єкт, вона формує первинний опис і зіставляє цей опис з фреймами, що зберігаються в її пам'яті (формування первинного опису та зіставлення можуть відбуватися і паралельно). Якщо зіставлення з деяким фреймом пройшло достатньо успішно, система відносить новий об'єкт до поняття, що описується цим фреймом. На основі цього відповідним чином здійснюється поповнення первинного опису, а саме:

- формується загальна структура опису нового об'єкта (фактично тепер він розглядається як екземпляр певного фрейму);
- заповнюються всі слоти нового екземпляра, значення яких визначаються в описі родового фрейму або задаються в первинному описі;
- якщо опис недостатньо повний (не вистачає значень деяких важливих слотів) або суперечливий (це відбувається, наприклад, якщо об'єкт зіставляється з кількома фреймами або інформація, що вводиться, суперечить визначеній за успадкуванням), можуть бути активізовані процедури уточнення.

Розглянемо *приклад*.

Нехай у пам'яті інтелектуальної системи зберігається фрейм опису одного дня навчання в школі:

ЗАНЯТТЯ_В_ШКОЛІ ((Вид_роботи, НАВЧАННЯ),
(Початок, 9.00),
(Кількість_уроків, _),
(Закінчення, Початок + Кількість_уроків * 0.45 +
0.10 * (Кількість_уроків – 1)),
(Коли, _),
(Хто, _),
(Де, <Вул. О.Бендера, 999>)).

Зверніть увагу на те, що атрибут **Закінчення** є обчислюваним, оскільки його значення залежить від кількості уроків.

Нехай на вхід системи подається речення *"Вчора Петро був у школі"*. Система повинна на основі відомих їй алгоритмів виявити, що це речення зіставляється з фреймом **ЗАНЯТТЯ_В_ШКОЛІ** і сформувати опис на основі цього фрейму (фактично, нових екземплярах фрейму). Вона дає цьому опису назву, наприклад, **Заняття_Петра** і відображає в заголовку, що це екземпляр фрейму **ЗАНЯТТЯ_В_ШКОЛІ**. Далі заповнюються слоти, і опис набуває такого вигляду:

Заняття_Петра (ЗАНЯТТЯ_В_ШКОЛІ) ((Вид_роботи, НАВЧАННЯ),
(Початок, 9.00),
(Кількість_уроків, _),
(Закінчення, Початок + Кількість_уроків * 0.45 +
0.10 * (Кількість_уроків – 1)),
(Коли, ВЧОРА),
(Хто, Петро),
(Де, <Вул. О.Бендера, 999>)).

Слоти **Вид_роботи**, **Де** та **Початок** були заповнені на основі базового фрейму, а слоти **Коли** та **Хто** — з введеного тексту. Слоти **Кількість_уроків** і **Закінчення** залишилися незаповненими — для цього не вистачає інформації.

Тепер система в змозі відповідати на ряд запитань, наприклад: *"Що вчора робив Петро?"* або *"Де вчора був Петро?"*. Точнішу відповідь на запитання *"Коли Петро був у школі?"* система може дати, якщо вона має інформацію про поточний час.

Якщо системі потрібно відповісти на запитання *"Коли Петро закінчив навчання у школі?"*, вона має активізувати **процедуру уточнення** (у даному разі спитати у користувача, скільки було уроків). Після відповіді система може здійснити необхідні обчислення та відповісти на поставлене запитання.

Важливою властивістю фреймових моделей є те, що ***фрейми можуть містити процедури, які виконують ті чи інші дії***. Такі процедури прийнято називати ***приєднаними***. Їх можна розглядати як окремі слоти. На інтуїтивному рівні наявність приєднаних процедур дає підстави говорити про те, що фрейм має здатність виконувати ті чи інші операції.

Найбільш використовуваним типом приєднаної процедури є процедура, яка активізується шляхом зовнішнього виклику (за ***повідомленням***, яке надходить від іншого фрейму).

Важливим типом приєднаної процедури є ***демон*** — процедура, яка автоматично активізується, тільки-но стають істинними певні умови (наприклад, проходить певний проміжок часу або значення одного слота стає більшим за значення іншого).

Велике значення мають ***слотові демони***. На відміну від звичайних приєднаних процедур, що описують поведінку фрейму в цілому, слотові демони пов'язані з конкретними слотами. Типовим є використання стандартних слотових демонів, які визначають, що слід робити, якщо деякий слот змінився, або якщо відбулося звернення до незаповненого слота.

Існують різні підходи до зберігання фреймових структур у пам'яті обчислювальних машин, і, відповідно, до мов опису знань на основі фреймів. Таких мов є досить багато; як приклади можна навести **KRL, FRL** (розвитком якого є **HPRL**), **ПРЕФ, ДИЛОС, RLL** [145, 223]. Кожна мова передбачає свої структури для подання фреймів і типові операції над ними.

Можна вважати, що у типовому фреймі зберігається така інформація:

- ім'я і загальний опис фрейму;
- інформація про батьківський фрейм;
- інформація про окремі слоти: імена та значення слотів, а також їх властивості (наприклад, тип слота, умови коректності інформації, яка зберігається в слоті, тощо). Важливою властивістю слота є, зокрема, ***режим успадкування***: можна задати обов'язкове успадкування, успадкування за певної умови певній умові, відмову від успадкування тощо.

І семантичні мережі, і фрейми в першу чергу є **формалізмами для структуризації знань**, на відміну від логічних і продукційних моделей, центральне місце в яких займають процедури дедуктивного виведення. Семантичні мережі основний акцент роблять на описі різноманітних зв'язків між інформаційними одиницями, а фреймові моделі — на відношеннях узагальнення та агрегації.

Семантичні мережі та фрейми мають значно більше спільних рис, ніж відмінностей, і тому на сучасному етапі фреймові моделі та семантичні мережі розглядають, як правило, у спільному контексті. З одного боку, ніщо не заважає розглядати вузли семантичної мережі як фрейми з власною внутрішньою структурою. З іншого боку, можна вводити різноманітні зв'язки між слотами фреймів. Тоді фрейм набуває рис семантичної мережі.

6.4. Мережі подібностей і відмінностей

При пошуку фреймів, які найкраще підходять для опису даного поняття, велике значення мають ***мережі подібностей*** або ***відмінностей***. Кожний вузол такої мережі є фреймом опису певного поняття, а зв'язки відповідають подібностям чи відмінностям між фреймами.

За допомогою мереж подібностей можна визначати семантичну близькість між поняттями [254]. Ця обставина може бути використана, наприклад, при зіставленні. Якщо при зіставленні з фреймом, що описує певне поняття, виникають розбіжності, мережа подібностей дозволяє переходити до сусідніх, схожих з ним, фреймів з метою пошуку точнішої відповідності.

Мережа відмінностей концентрується на описі відмінностей між поняттями. Можна, наприклад, зафіксувати деякий фрейм як ***базовий***, а для інших зберігати лише відмінності від базового або сусіднього фрейму. Так, у [223] показано, як один предмет може описуватися різними фреймами, що відповідають різним умовам спостерігання цього предмета. Риси, спільні для цих різних фреймів, описуються базовим фреймом.

Сценарієм *називається фреймоподібна структура знань, яка визначає послідовність подій, характерних для певного процесу чи для певної ситуації, або причинно-наслідкові зв'язки між подіями.*

Поняття сценарію можна визначити і на основі семантичних мереж (типовий сценарій можна розглядати як семантичну мережу, всі зв'язки якої мають тип "причина — наслідок" або "бути раніше — бути пізніше"). Це ще раз підкреслює тісний зв'язок між семантичними мережами та фреймами.

Наведемо приклад дуже простого сценарію, який визначає послідовність дій при відвідинах ресторану [129]. Для цього сценарію визначені слоти **Мета**, **Ролі** (основні дійові особи), **Змінні** (риси, різні для неоднакових конкретних реалізацій сценарію), **Кількість_сцен** і слоти, що відповідають сценам (тут — подіям), що послідовно змінюють одна одну:

Відвідини_ресторану
(**Мета**: Приймання їжі;
Ролі: *Відвідувач*, *Офіціант*;
Змінні: *Назва_Ресторану*, *Місце*, *Чекання_1*, *Чекання_2*, *Їжа, Сума*;
Кількість сцен: 4

Сцена_1 (Зайняття місця):
Відвідувач заходить до *Назва_Ресторану*
Відвідувач вибирає *Місце* (умова: *Місце* повинно бути вільним)
Відвідувач проходить до *Місця* і сідає

Сцена_2 (Замовлення):
Відвідувач чекає *Чекання_1* хвилин
Офіціант підходить до *Відвідувача*
Офіціант дає *Відвідувачу* меню
Відвідувач читає меню і вибирає *Їжу*
Офіціант відходить

Сцена_3 (Прийом їжі):
Відвідувач чекає *Чекання_2* хвилин
Офіціант приносить *Їжу*

Відвідувач з'їдає *Їжу*

Сцена_4 (Завершальна):
Відвідувач кличе *Офіціанта*
Офіціант підходить до *Відвідувача* і дає йому рахунок на *Суму*
Відвідувач платить *Суму*
Відвідувач залишає *Назва_Ресторану*).

Цей сценарій задає загальну схему розвитку подій: опис конкретної послідовності подій при конкретному відвідуванні ресторану породжується при підстановці замість Ролей і Змінних конкретних значень.

Даний сценарій можливо розглядати як ***базовий***, на основі якого можна будувати схожі сценарії, що можуть виникати при іншому розвитку подій (наприклад, *Офіціант* приніс не ту *Їжу*, яку замовляв *Відвідувач*, або *Відвідувач* не має потрібної *Суми* грошей, або *Відвідувач* намагався втекти з ресторану, не розплатившися, тощо).

Як і інші типи фреймів, сценарії можна використовувати для поповнення опису ситуацій. Так, якщо інтелектуальна система сприймає текст *"Студент Іванов відвідав ресторан"*, вона зіставляє його з відомими їй сценаріями і на основі цього може відповідати на запитання, пов'язані з тим, що саме робив Іванов у ресторані.

Є відомий анекдот про спілкування з інтелектуальною системою. Людина приходить на роботу і питає у робота:

— *Де це?*
— *Що саме? — запитує той.*
— *Сам знаєш, — відповідає людина.*
— *А... — каже машина і приносить те, що потрібно.*

Тепер ми можемо пояснити, що може лежати в основі прийняття рішення інтелектуальним роботом. Механізм може бути, наприклад, таким. Робот спостерігає ситуацію, яка полягає в тому, що людина приходить на роботу і питає *"Де це?"*. Він зіставляє цю ситуацію зі своїми знаннями і знаходить кілька сценаріїв можливого розвитку подій. Він не знає, який з них вибрати, і тому задає уточнююче запитання. Відповідь *"Сам знаєш"* інтерпретується так, що варто використати виведення за замовчуванням, тобто вибрати той сценарій, який реалізується найчастіше.

6.5. Фрейми та об'єктно-орієнтоване програмування

Поняття фрейму, яке розглядається в теорії штучного інтелекту, та поняття об'єкта (в іншій термінології — класу), тісно пов'язані між собою. Парадигма ***об'єктно-орієнтованого програмування*** випливає з об'єктно-орієнтованого сприйняття світу, що складається з великої кількості об'єктів. Вони є порівняно незалежними, але постійно взаємодіють між собою. Кожний об'єкт має певні властивості та вміє виконувати деякі функції. **Можна вважати, що об'єктна модель є конкретизацією абстрактнішої фреймової моделі.**

У [31] сформульовано деякі важливі визначення.

Об'єктом *називається абстракція, що характеризується станом, поведінкою та ідентифікованістю; сукупності схожих об'єктів утворюють клас; терміни "екземпляр класу" та "об'єкт" рівноправні.*

Стан об'єкта *характеризується переліком (як правило, статичним) усіх його властивостей і поточними (як правило, динамічними) значеннями кожної з цих властивостей.*

До цього можна додати, що "статичний перелік властивостей" є характеристикою всього класу, а "поточні динамічні значення" — характеристикою окремого об'єкта, або екземпляра класу. Опис окремого екземпляра на основі загального опису класу можна отримати, якщо визначити конкретні значення властивостей.

Поведінка *визначається тим, як об'єкт функціонує та реагує на зовнішні події; поведінку прийнято характеризувати в термінах зміни станів об'єктів та передачі повідомлень між об'єктами; поточний стан об'єкта є сумарним результатом його поведінки.*

Ідентифікованість *— це така властивість об'єкта, яка відрізняє його від усіх інших об'єктів.*

Важливим є те, що об'єкти слід розглядати як абстракції певних сутностей, тобто об'єкт описує властивості даної сутності, що є найважливішими з певної точки зору. При цьому для екземплярів класу спільними є **всі** характеристики, а не лише деякі. Точніше, **спільним є перелік характеристик**, а не конкретні значення; екземпляри одного класу можна розрізняти між собою саме за рахунок того, що характеристики різних екземплярів класу мають різні значення.

Об'єкти та класи мають такі найважливіші властивості [31]:

1. ***Абстрагування*** — виокремлення в описі класу найважливіших характеристик деякої сутності, що відрізняють її від усіх інших сутностей.
2. ***Інкапсуляція*** — елементи об'єкта, що визначають його властивості та поведінку, повинні бути відокремлені один від одного. Інакше кажучи, клас повідомляє про свої функціональні можливості, але приховує від зовнішнього світу особливості реалізації цих можливостей і свою внутрішню будову.
   Слід відмітити, що в програмуванні термін "інкапсуляція" часто вживається у вужчому значенні, а саме: в класі можуть бути описані процедури ***(методи)***, що визначають його поведінку.
3. ***Модульність*** — об'єкт складається з окремих відносно самостійних модулів.
4. ***Ієрархія*** — об'єкти та класи є ієрархічно упорядкованими. Зокрема, вони утворюють ієрархію класів та ієрархію об'єктів (див. п. 4.5). Успадкування (наслідування) пов'язане саме з цими ієрархіями, в першу чергу — з ієрархією класів.

Існує багато об'єктно-орієнтованих мов програмування, тобто мов, які безпосередньо підтримують об'єктну модель та об'єктно-орієнтовану парадигму програмування. Історично першою була мова **Smalltalk**. Зараз популярними є **Delphi** (візуальна реалізація Паскалю), **C++** (стандарт де-факто в сфері об'єктно-орієнтованого програмування), **Java**. Також розвиваються технології об'єктно-орієнтованих баз даних [99, 147].

6.6. Поняття про мову UML

Мова **UML**, розроблена Г. Бучем, Дж. Рамбо та А. Джекобсоном, є мовою семантичного моделювання предметних областей та подальшого проектування відповідних програмних засобів. В основі цієї мови лежить об'єктно-орієнтований аналіз, тобто виокремлення класів та опис їх характеристик і взаємозв'язків між ними. У 1997 р. ця мова була прийнята консорціумом **OMG (Object Management Group)** як стандарт.

Основним елементом опису предметної області на основі UML є ***діаграма класів***, яка, в свою чергу, виступає описом класів і відношень між ними. У мові UML визначені формалізовані правила такого опису. Діаграма класів може бути легко реалізована в об'єктно-орієнтованих мовах програмування.

Згідно з авторським описом [32] у мові UML визначено такі основні типи відношень між класами: залежність, узагальнення, асоціація.

Асоціація описує структурні взаємозв'язки між класами. Важливим частковим випадком асоціації є агрегація.

Узагальнення реалізується відношенням "Є" (клас-підклас).

Залежність є відношенням, яке описує вплив однієї сутності щодо іншої.

Діаграми об'єктів дозволяють моделювати екземпляри сутностей, що описуються діаграмами класів.

Крім діаграм класів та об'єктів, в UML визначено ряд інших елементів, опис яких можна знайти в [32].

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте визначення фрейма як структури даних для опису певного поняття.
2. Що означає мінімальність опису у визначенні фрейму, що було дано М. Мінським?
3. Яким чином можна описати об'єкт на основі фреймової моделі?
4. Яким чином описи конкретних об'єктів утворюються з описів відповідних фреймів?
5. Опишіть схему поповнення первинних описів на основі фреймових моделей.
6. Що таке приєднана процедура?
7. Що таке демон?
8. Що таке мережі подібностей і відмінностей? Опишіть їх можливі застосування.

9. У чому полягає зв'язок між семантичними мережами і фреймами?
10. Що таке сценарій?
11. Опишіть послідовність подій з деякої предметної області у вигляді сценарію, що залежить від ролей і змінних.
12. Поясніть зв'язок між фреймовими моделями та об'єктно-орієнтованими моделюванням і програмуванням.
13. Дайте визначення об'єкта та класу за Г. Бучем.
14. Перелічіть та охарактеризуйте основні властивості об'єктів і класів.
15. Дайте загальну характеристику мови UML.
16. Що таке діаграма класів?
17. Перелічіть основні відношення між класами, визначені в UML.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Чому фреймові структури відіграють важливу роль у розумінні, зокрема текстів?
2. Яким чином відношення узагальнення, асоціації та залежності можуть бути реалізовані в конкретних мовах програмування?

Розділ 7

ЛОГІЧНІ МОДЕЛІ ТА МЕТОД РЕЗОЛЮЦІЙ

— Підемо шляхом логіки...
— Підемо разом...
З кінофільму "Іронія долі, або З легким паром!"

7.1. Логічні побудови та логічні моделі

Поняття "логіка" має глибокі філософські корені. Для теорії і практики штучного інтелекту особливе значення має таке питання: яким чином можна формалізувати логічні побудови так, щоб вони могли здійснюватися автоматично, без участі людини.

Як уже зазначалося, логічне виведення за дедукцією (від загальних правил до часткових наслідків) є одним з основних механізмів мислення, до того ж цілком бездоганним з погляду логіки. Саме цей вид логічного виведення найбільш вивчений і найкраще піддається формалізації [129, 144, 167, 172, 272].

Можна розглядати, як мінімум, дві задачі пов'язаних з автоматизацією дедуктивних побудов:

- ***задача виведення наслідків***, яка формулюється так: знайти всі вірно побудовані формули, які можна логічним шляхом вивести з аксіом на основі правил виведення; або, у практичнішій інтерпретації — знайти всі наслідки із заданих фактів;

- задача перевірки справедливості даного твердження. Часто вона трактується як задача ***автоматичного доведення теорем*** і формулюється таким чином. Дана деяка теорія (логічна модель); потрібно встановити, чи є певне твердження теоремою, тобто чи можна його вивести з аксіом даної теорії.

Автоматизація дедуктивних побудов знаходить широке практичне застосування, наприклад, в експертних системах. Як ще один приклад можна згадати програмні системи, які аналізують математичні тексти, написані спрощеною математичною мовою, та перевіряють правильність доведень, що містяться у цих текстах.

Механізм автоматичного доведення теорем було покладено в основу однієї з основних парадигм сучасного програмування — ***логічного програмування***. Програма розглядається як набір логічних формул разом з теоремою, що має бути доведена. Найвідомішим представником логічного програмування є мова Пролог [26, 172, 292].

Логічні моделі є одним з основних засобів задання знань. Позитивною характеристикою логічних моделей виступає однозначність теоретичного обґрунтування і можливість реалізації формально точних логічних побудов; недоліком — формальний процедурний стиль мислення. Людська логіка — інтелектуальна модель з нечіткою структурою. У цьому полягає її відмінність від формальної логіки.

Логічною моделлю *називається формальна система, що задається четвіркою* $<T, P, A, B>$. *Тут* T *— множина базових елементів (алфавіт),* P *— множина синтаксичних правил, на основі яких конструюються правильно побудовані формули;* A *— множина аксіом, тобто формул, які приймаються за істинні,* B *— множина правил виведення.*

У рамках логічної моделі істинному твердженню відповідає ***теорема***, тобто правильно побудована формула, яка може бути виведена з аксіом шляхом скінченного числа застосувань правил виведення.

Логічні моделі та продукційні системи (розд. 8) основний акцент роблять саме на дедуктивному логічному виведенні.

У найтиповішому випадку логічна модель задання знань базується на формалізмах ***логіки предикатів першого порядку***.

7.2. Короткий вступ до числення предикатів

Предикатом називається деяка логічна функція від довільного числа аргументів, яка приймає одне з двох можливих значень — *"істина"* або *"хибність"*. Предикат можна розуміти як певне твердження, істинність якого залежить від змінних — об'єктів, про які йдеться в цьому твердженні. Як приклад можна навести фразу *"X більше за 2"*. Цей предикат є функцією від аргументу X і набуває значення *"істина"*, наприклад, при $X = 3$, і *"хибність"* при $X = 1$.

Логіку предикатів деякою мірою можна вважати спеціальним математичним апаратом формалізації людського мислення. Тому визнається,

що мови програмування логічного типу є найзручнішими для роботи з базами знань.

Числення предикатів використовує такі основні елементи:

1) константи (константні терми) $c_1, c_2, \ldots$;

2) змінні (змінні терми) $x_1, x_2, \ldots$;

3) функціональні літери $f_1, f_2, \ldots$;

4) предикатні літери $p_1, p_2, \ldots$;

5) логічні символи $\rightarrow$, $\leftrightarrow$, &, $\vee$, ~, $\forall$, $\exists$;

6) спеціальний символ □.

Елементарне твердження складається з предиката і зв'язаних з ним термів. Складні твердження будуються з елементарних за допомогою логічних зв'язок. Серед них можна виділити логічні зв'язки: *"і" (and, &), "або" (or, $\vee$), "ні" (not, ~)* та імплікацію ($\rightarrow$). Імплікація посідає особливе місце, оскільки вона використовується для побудови правил виведення і читається "якщо …, тоді …".

Наступний перелік містить опис зв'язків, які використовуються в логіці, та їх змістовних інтерпретацій. Тут *a* та *b* означають будь-які твердження.

Зв'язок	Запис	Інтерпретація
Заперечення	$\sim a$	"не є *a*"
Кон'юнкція	$a \,\&\, b$	"*a* та *b*"
Диз'юнкція	$a \vee b$	"*a* або *b*"
Імплікація	$a \rightarrow b$	"з *a* випливає *b*"
Тотожність (еквівалентність, рівність)	$a \leftrightarrow b$	"*a* еквівалентне *b*"

Наприклад, запис:

чоловік (*женя*) $\vee$ *жінка* (*женя*)

є однією з формалізацій запису твердження, що людина Женя може бути або чоловіком, або жінкою. Аналогічно,

чоловік (*X*) $\rightarrow$ *людина* (*X*)

означає, що якщо дехто на ім'я *X* є чоловіком, то він є і людиною.

Для імплікації та тотожності справедливі такі твердження:

$a \rightarrow b$ дорівнює $(\sim a) \vee b$,

$a \leftrightarrow b$ дорівнює $(a \,\&\, b) \vee (\sim a \,\&\, \sim b)$,

$a \leftrightarrow b$ дорівнює $(a \rightarrow b) \,\&\, (b \rightarrow a)$.

З кожним предикатом може бути пов'язаний ***квантор*** — елемент, який визначає, за яких умов предикат перетворюється на істинне висловлювання. Розрізняють ***квантор узагальнення*** ($\forall$) і ***квантор існування*** ($\exists$).

Якщо *u* позначає змінну, а *r* — твердження, то запис $\forall u\ r$ означає, що *"r справджується для всіх u"*, а запис $\exists u\ r$ означає, що *"існує u, для якої r справджується"*. Перший квантор називають квантором узагальнення, оскільки кажуть про будь-що у всесвіті (*"для будь-якого u, ..."*). Для зруч-

ності запису позначатимемо його **all**. Другий називається квантором існування, оскільки вказує на існування деяких об'єктів (*"існує и таке, що ..."*). Для зручності запису його позначатимемо **exists**.

Ось приклади використання цих кванторів:

All (*X, чоловік* (*X*) → *людина* (*X*))

означає, що для будь-якої змінної *X*, якщо вона означає чоловіка, то означатиме і людину також, або ж для будь-якого *X*, якщо *X* — чоловік, то *X* — людина; або ж: всі чоловіки — люди.

За тим самим принципом:

exists (*Z, батько* (*іван, Z*) & *жінка* (*Z*))

означає, що існує певний об'єкт, що може бути позначений змінною *Z*. При цьому **Іван** є батьком *Z* і *Z* — жінка. Ми можемо прочитати це таким чином: *існує Z таке, що* ***Іван*** *є батьком Z і Z — жінка*; або: ***Іван*** *має дочку*.

У логіці предикатів елементарним об'єктом, який має значення "істина", є ***атомарна формула (літерал)***. Атомарна формула містить символьні позначення предиката і термів, які відіграють роль аргументів цього предиката. Узагальнено позначення предиката є ім'ям відношення, яке існує між аргументами.

Атомарна формула записується як позначення предиката і має такий вигляд: $P(t_1, t_2, \ldots, t_n)$,

де P — позначення предиката, а $t_1, t_2, \ldots, t_n$ — терми.

Число термів для кожного предиката фіксоване і називається його *арністю*. Терми визначаються так:

1) константний терм — терм;

2) змінний терм — терм;

3) якщо арність функціональної літери f є n, а $t_1, t_2, \ldots, t_n$ — терми, тоді $f(t_1, t_2, \ldots, t_n)$ також терм.

Вірно побудована формула отримується в результаті комбінування атомарних формул за допомогою логічних зв'язок.

Символ □ позначає хибну замкнуту формулу і визначає поняття "протиріччя". Так, формула $A \rightarrow$ □ означає хибність A, вона еквівалентна формулі $\sim A$.

Серед формул можна виокремити спеціальний клас формул: тотожньо істинні формули, які називають тавтологіями. Приклад тавтології:

$\sim \sim A \leftrightarrow A$.

Літерал називається ***позитивним***, якщо він не стоїть під знаком заперечення.

Літерал називається ***негативним***, якщо він стоїть під знаком заперечення.

Диз'юнктом *(або* ***фразою теорії****) називається диз'юнкція деякої кількості атомарних формул.*

У теорії і практиці автоматичного доведення теорем найвживанішими є фрази спеціального типу, які називаються ***фразами Хорна*** (або ***хорнівськими диз'юнктами***).

Фразою Хорна *називається диз'юнкція довільної кількості атомарних формул, з яких позитивною є не більше ніж одна (тобто у фразі Хорна лише одна атомарна формула може не стояти під знаком заперечення).*

Фрази Хорна по суті є імплікаціями. Справді, розглянемо фразу $\sim A \vee \sim B \vee \sim C \vee D$ (кожний з літералів A, B, C, D може залежати від довільної кількості констант і змінних, тут це несуттєво). Усі літерали, крім D, є негативними, отже, дана фраза являє собою фразу Хорна. Але відповідно до правил де Моргана вона еквівалентна фразі $\sim (ABC) \vee D$. А це, в свою чергу, є іншою формою запису імплікації (твердження D випливає з кон'юнкції тверджень A, B, C).

Для конкретних застосувань іноді потрібно привести логічні формули до спеціального вигляду. Прикладом може послужити ***пренексна нормальна форма***.

Пренексною нормальною формою *називається запис логічної формули у вигляді:*

$$K_1 x_1 \, K_2 x_2 \ldots K_n x_n \, M,$$

де K_i — один з кванторів (існування чи узагальнення), M — деяка кон'юнктивна нормальна форма, тобто кон'юнкція деякої кількості диз'юнктів.

Пренексну нормальну форму легко побудувати шляхом тотожних перетворень логічних формул.

Після отримання пренексної нормальної форми необхідно усунути у ній квантори існування та узагальнення. Перший усувається зі збереженням несуперечливості формули шляхом введення так званих ***констант Сколема*** і ***функцій Сколема***. Розглянемо два приклади, які пояснюють суть методу.

Приклад 7.1. Нехай маємо формулу $\exists x$: $P(x)$. По суті це означає, що можна вказати деяку предметну константу c, невизначену, для якої твердження $P(c)$ є істинним. Тоді квантор існування усувається явним введенням цієї предметної константи. Таким чином, здійснюється перехід від формули $\exists x$: $P(x)$ до формули $P(c)$. Константа c називається константою Сколема.

Приклад 7.2. Нехай маємо формулу $\forall y \, \exists x$: $P(x, y)$. Тут квантор існування перебуває в області дії квантора узагальнення, і процедура дещо ускладнюється. Знову ж таки ми можемо вказати деякий елемент c, для якого твердження $P(c, y)$ є істинним, але тепер c залежить від змінної y. Тому від формули $\forall y \, \exists x$: $P(x, y)$ можна перейти до формули $\forall y \, P(h(y), y)$. Тут $h(y)$ — деяка функція, в цілому невизначена. Вона має назву функція Сколема.

Квантори узагальнення усуваються механічно на основі стандартної домовленості: вважається, що якщо фраза залежить від деяких змінних, то всі вони зв'язані квантором узагальнення. Наприклад, якщо x, y, z — змін-

ні, то від формули $\forall x\ \forall y\ \forall z\ P\ (x, y, z)$ здійснюється перехід до формули $P\ (x, y, z)$.

Для детальнішого ознайомлення з основами числення предикатів рекомендуємо звернутися до відповідного підручника з математичної логіки, наприклад [112, 179].

7.3. Фразова форма запису логічних формул

Як зазначалося раніше, логічна формула, записана за допомогою імплікацій і тотожностей, може бути переписана в термінах кон'юнкцій, диз'юнкцій і заперечень. Розглянемо, як твердження числення предикатів перетворюється на спеціальну уніфіковану фразову форму. Набори фраз, записаних у такій формі, нагадують програми логічних мов програмування, дослідження яких нас також цікавитиме.

Дамо неформальний опис такого переведення, що має шість основних дій.

Дія 1. Вилучення імплікацій і тотожностей.

Вилучення імплікацій "$\rightarrow$" і тотожностей "$\leftrightarrow$" можна провести на основі формул, наведених у п. 7.2. Наприклад, формула

all (*X, чоловік* (*X*) $\rightarrow$ *людина* (*X*))

перетвориться на

all (*X,* ~ *чоловік* (*X*) $\vee$ *людина* (*X*)).

Дія 2. Втягнення заперечення всередину.

Ця дія застосовується для випадків, коли "~" вживана у формулі, що не є частковою. Тоді робиться така заміна:

~ (*людина* (*іван*) & *живий* (*іван*))

переводиться у

~ (*людина* (*іван*) $\vee$ ~ *живий* (*іван*))

та

~ *all* (*Y, люди* (*Y*))

перетворюється на

exists (*Y,* ~ *люди* (*Y*)).

Можливість цієї дії зумовлена такими рівностями:

~ (*a* & *b*) означає те саме, що й (~ *a*) $\vee$ (~ *b*),

~ *exists* (*u, r*) означає те саме, що й *all* (*u,* ~ *r*),

~ *all* (*u, r*) означає те саме, що й *exists* (*u,* ~ *r*).

Після другої дії у наших формулах залишаться лише заперечення, прив'язані до конкретних формул. Часткові твердження з або без "~" попереду ми називаємо ***точними***. У наступних кількох діях ми будемо поводитися з точними твердженнями як з простими одиницями.

Дія 3. Заміна змінних.

Наступна дія включає вилучення існуючих кванторів. Це робиться шляхом вставлення нових константних символів на місце, де були змінні, що являють собою існуючі квантори. Тобто, замість того, щоб говорити, що існує об'єкт з певним набором характеристик, можна створити ім'я для

такого об'єкта і просто сказати, що в нього є характеристики. В цьому і полягає основна суть введення сколемівських констант. Ця дія є не повністю еквівалентною, але вона має одну важливу властивість. Існує інтерпретація символів формули, яка справджує формулу тільки тоді, коли існує інтерпретація для сколемівської версії формули. Для наших потреб цієї форми відповідності достатньо.

Тому, наприклад:

exists (*X, чоловік* (*X*), & *батько* (*X, іван*))

перетворюється за допомогою сколемізації на

чоловік (*g1*) & *батько* (*g1, іван*)

де *"g1"* — деяка нова константа, що ніде раніше не використовувалась. Константа *"g1"* представляє певного чоловіка (*чоловік*), чиїм батьком (*батько*) був Іван (*іван*). Тут важливим є застосування символу, який раніше ніде не вживався, оскільки твердження *"exists"* не каже, що певна людина є сином Івана, а лише стверджує, що є така людина. Може виявитись, що *"g1"* пов'язана з якимось іншим константним символом, але ця інформація не надається даним твердженням.

Коли у формулі використовуються квантори узагальнення, сколемізація стає не такою простою. Наприклад, якщо ми перетворимо

all (*X, людина* (*X*) → *exists* (*Y, мати* (*X, Y*)))

("кожна людина має матір") на

all (*X, людина* (*X*) → *мати* (*X, g2*)),

то це б означало, що всі люди мають одну матір — *"g2"*. Коли змінні зв'язані квантором узагальнення, їх перетворюють на функціональні символи, щоб показати, наскільки *"exists"* залежить від того, з чим зв'язана змінна. Тому останній приклад потрібно перетворити таким чином:

all (*X, людина* (*X*) → *мати* (*X, g2* (*X*))).

У даному разі *"g2"* позначає функціональний символ, який за ім'ям людини видасть вам ім'я її матері.

Дія 4. Переміщення квантора узагальнення назовні.

Ця дія дуже проста. Квантор узагальнення переміщується назовні, при цьому формула зберігає свій попередній зміст. Наприклад,

all (*X, чоловік* (*X*) → *all* (*Y, жінка* (*Y*) → *подобається* (*X, Y*)))

перетворюється на

all (*X, all* (*Y, чоловік* (*X*) → (*жінка* (*Y*) → *подобається* (*X, Y*))))

Оскільки кожна змінна тепер зв'язана квантором узагальнення ззовні формули, то квантори узагальнення більше не несуть додаткової інформації. Отже, можна скоротити формули, опустивши ці квантори. Ми просто мусимо пам'ятати, що кожна змінна зв'язана кванторами узагальнення, які ми опустили.

Дія 5. Перенесення "&" через "∨".

Логічна формула повинна бути зведена до особливої форми — нормальної кон'юнктивної формули, в якій кон'юнкції не з'являються посеред диз'юнкцій. Для цього існують два правила:

$(A \& B) \vee C$ дорівнює $(A \vee B) \& (B \vee C)$,

$A \vee (B \& C)$ дорівнює $(A \vee B) \& (A \vee C)$.

Наприклад, фразу

вихідний_день $(X) \vee$ (*працювати* (*іван*, X) & (*злий* (*іван*) $\vee$ *сумний* (*іван*)))

можна трактувати так: *"для будь-якого X, X може бути вихідним днем, або Іван може працювати в X і бути злим або сумним"*. Останню можна перетворити на фразу

(*вихідний_день* $(X) \vee$ (*працювати* (*іван*, X) &

(*вихідний_день* $(X) \vee$ (*злий* (*іван*) $\vee$ *сумний* (*іван*))))

"для всіх X, по-перше, X або вихідний, або Іван працює в X, по-друге, або X вихідний, або Іван сумний та злий)".

Дія 6. Перехід до фраз.

Тепер наша формула є набором атомарних формул, розділених "$\vee$" або пов'язаних "&". Якщо не брати до уваги "$\vee$", в нас може вийти таке:

$(A \& B) \& (C \& (D \& E))$,

де змінні можуть заміщати складні твердження, але без "&" всередині. По-перше, тепер дужки не потрібні, оскільки для "&" вони не мають значення, і наш вираз еквівалентний до

$A \& B \& C \& D \& E$.

По-друге, порядок теж не має значення. По-третє, нам не потрібні і "&", оскільки ми знаємо, що наша формула — це послідовність тверджень, з'єднаних "&", і останні можна опустити, сказавши, що це є множина $\{A, B, C, D, E\}$. Кажучи, що це множина, ми наголошуємо на тому, що порядок не суттєвий. Елементи цієї множини називаються фразами. Отже, будь-яка формула предикатного числення еквівалентна (в певному розумінні) до множини фраз.

Повернемося до самої фрази. Оскільки елементами фрази є або точні, або розділені точно "$\vee$" твердження, то їх загальний вигляд може набувати такого вигляду:

$((V \vee W) \vee X) \vee (Y \vee Z)$,

де змінні є точними. Тому можна знову перейти до множини розділених точних фраз $\{V, W, X, Y, Z\}$.

Тепер формула остаточно досягла фразової форми. Підсумуємо дещо.

Отже, фразова форма логіки предикатів задає такий спосіб запису формул, який використовують тільки з'єднувачі типу &, $\vee$, $\sim$.

Негативна або позитивна атомарна формула називається літералом. Кожна формула — множина літералів, які з'єднані символом $\vee$. Негативні і позитивні літерали відповідно групуються. Схематично фраза має такий вигляд:

$P_1 \vee P_2 \vee \ldots P_n \vee N_1 \vee N_2 \vee \ldots N_m$,

де $P_1, \ldots, P_n$ — позитивні літерали, а $N_1, \ldots, N_m$ — негативні. З іншого боку, фразу можна розглядати як узагальнення поняття імплікації. Дійсно,

якщо A і B атомарні формули, тоді формулу $A \rightarrow B$ можна переписати і в такому еквівалентному вигляді: $\sim A \vee B$. Звідси отримаємо фразову формулу $B \vee \sim A$.

Фраза має таку семантику. Всі позитивні атомарні формули виступають у ролі альтернативних висновків, а негативні є необхідними умовами. Наприклад, $A \vee B \vee \sim C \vee \sim D$ зазначає, що A і B будуть істинними тоді і тільки тоді, коли є істинними C і D.

Елементарна фраза має тільки один літерал. Теорія записується у вигляді множини фраз, які неявно з'єднані між собою символом &. У літературі зустрічається і друга форма запису фрази за допомогою зворотної стрілки (читається як "імплікується"). Так, остання фраза може мати вигляд $A, B \leftarrow \sim C, \sim D$.

Для скорочення об'ємів обчислень на практиці застосовують спеціальний клас фраз — фрази Хорна. Аналогічно, фразу Хорна можна записати також кількома способами. Наприклад, твердження

$A \vee \sim B \vee \sim C \vee \sim D$

еквівалентне

$A \leftarrow B, C, D$

і в мовах логічного програмування набуває вигляду A:- B, C, D.

Розглянемо ще один приклад.

Нехай маємо формулу

(*особа* (*петро*) & *особа* (*іван*)) & ((*особа* (*X*) $\vee \sim$ *мати* (*X, Y*)) $\vee \sim$ *особа* (*Y*)),

яка після відповідних дій перетворюється на:

особа (*петро*):-.

особа (*іван*):-.

особа (*X*):-*мати* (*X, Y*), *особа* (*Y*).

Останній запис уже нагадує текст прологівської програми.

7.4. Аналіз і доведення теорем

Тепер, коли ми знаємо, як зводити формули числення предикатів до певної стандартної форми, ми повинні дізнатися, що з ними робити, як і де їх можна застосувати. Цілком очевидно, що, маючи набір тверджень, варто дослідити, які наслідки з них випливають. Нагадаємо, що твердження, які ми вважаємо істинними, називають аксіомами та гіпотезами, а ті твердження, що з них випливають,— теоремами.

У цьому розділі ми розглянемо на неформальному рівні процес автоматичного доведення теорем. Як уже зазначалося раніше, звістка про те, що цифрові комп'ютери можуть автоматично доводити теореми, у 60-ті роки спричинила бурхливий сплеск ентузіазму. Перші спроби знайти загальний алгоритм для вирішення задачі автоматизації дедуктивних побудов простежуються в роботах Лейбніца, Пеано, Гільберта. Центральне місце серед сучасних методів автоматичного доведення теорем займає ***метод резолюцій***, запропонований Дж. Робінсоном у 1965 р. Використання

цього методу виявилося ефективнішим, ніж класичне виведення на основі modus ponens.

Оскільки числення предикатів є алгоритмічно нерозв'язним, метод резолюцій дозволяє встановити, що деяка формула є теоремою, якщо вона насправді є теоремою. Для формул, що не є теоремами, метод резолюцій може працювати нескінченно довго. Саме на цій основі з'явилися ідеї, на яких базуються мови логічного програмування. Одним з фундаментальних проривів того часу було відкриття Дж. Робінсоном принципу аналізу та його застосування до автоматичного доведення теорем. Аналіз — це правило висновків, як одне твердження може випливати з іншого. Використовуючи аналітичний принцип, ми можемо на основі аксіом довести теореми стандартним шляхом з наших аксіом. Ми тільки маємо вирішити, до яких тверджень їх застосувати, і відповідний висновок дістанемо автоматично.

Аналіз розроблено і для роботи з формулами у фразовій формі. Маючи дві фрази, що пов'язані відповідним чином, він створить нову фразу, що буде наслідком перших двох. Основна ідея — це те, що якщо одна й та сама часткова формула зустрічається у правій частині однієї структури і в лівій частині іншої, то фраза отримується з'єднанням двох фраз з вилученням дубльованих формул, які з них випливають.

Наприклад:

з

злий (*іван*);
сумний (*іван*):-*робочий_день* (*сьогодні*),
йде_дощ (*сьогодні*).

та

неприємно (*іван*):- *злий* (*іван*),
стомлений (*іван*).

випливає

сумний (*іван*);
неприємно (*іван*):-*робочий_день* (*сьогодні*),
йде_дощ (*сьогодні*),
стомлений (*іван*).

Це виведення можна інтерпретувати так. Із тверджень *"якщо сьогодні будень і йде дощ, то Іван сумний або злий"* та *"коли Іван злий та стомлений, тоді йому неприємно"* можна вивести твердження *"якщо сьогодні робочий_день, або йде дощ, або Іван стомився, тоді він сумний і йому неприємно"*.

Але насправді не все так просто.

По-перше, набагато складніше, коли фрази містять змінні. У такому разі виведення включає додаткову операцію, яка здійснює "затвердження" змінних до такого випадку, коли формули можна вважати ідентичними. Наше друге спрощення — це те, що в загальному аналізі дозволяється кіль-

ком літералам справа відповідати кільком літералам зліва. Але може так трапитися, що лише один літерал вибрано з кожного пункту.

Поглянемо на приклад зі змінними:

студент (*fl* (*X*)); *професор* (*X*):-. (7.1)

професор (*Y*):- *поважати* (*fl* (*Y*), *Y*). (7.2)

поважати (*Z*, *іван*):-*студент* (*Z*). (7.3)

Перші дві фрази задають формулу *"якщо всі студенти поважають одну людину, то це — професор"*. Третя фраза передає твердження, що всі студенти поважають Івана. Аналіз (7.2) і (7.3) приводить (через "поважати") до:

професор (*іван*):-*студент* (*fl* (*іван*)) (7.4)

(*Y* в (7.2) зв'язано з "*іван*" в (7.3) та *Z* в (7.3) зв'язано з *fl* (*Y*) в (7.2)). Тому ми можемо аналізувати (7.1) і (7.4), що дасть нам

професор (*іван*); *професор* (*іван*):-.

Це рівнозначно факту: Іван — професор.

У формальному визначенні аналізу процес "спрощення", яким ми тут скористались, називається ***уніфікацією***. Даний набір часткових формул може бути уніфікованим, якщо їх можна підігнати під одну. Далі ми побачимо, що уніфікація та відповідність — це одне й те саме.

Існує така можливість, що ми маємо змогу продовжувати дії аналізу відносно гіпотез і спостерігати, чи не отримаємо потрібний результат. На жаль, ми не можемо цього гарантувати, навіть якщо те твердження, в якому ми зацікавлені, і справді випливає з тих, що маємо. Наприклад, у попередньому прикладі ми не можемо отримати окремий пункт *"професор (іван)"*, хоча це і логічно. Чи не означає це, що аналіз не є достатньо ефективним засобом для наших цілей? Але це не так. Ми можемо переорієнтувати наші зусилля таким чином, щоб аналіз гарантовано міг вирішувати наші задачі, якщо це взагалі можливо.

Важливою властивістю аналізу є те, що якщо набір пунктів непослідовний, то аналіз зможе отримати з них лише порожній пункт

:-.

Набір формул є непослідовним, якщо не існує можливостей інтерпретації предикатів, константних символів і символів функцій, що дозволило б швидко виразити твердження. Порожній пункт є логічним вираженням помилковості і репрезентує твердження, яке не може бути вірним. Отже, аналіз може нам сказати, коли наші формули неправильні, отримуючи цей доказ суперечності.

Яким чином ця властивість може нам допомогти? Якщо наші гіпотези послідовні, нам потрібно лише додати до них факти заперечення того, що ми хочемо довести, і аналіз видасть нам пусту фразу, якщо наші твердження випливають з гіпотез. Ми називатимемо пункти, які ми додаємо до гіпотез, цільовими твердженнями. Зауважте, що цільові твердження нічим не відрізняються від гіпотез — це такі самі фрази. Отже, якщо ми маємо набір

пунктів *A1*, *A2*, …, *An*, і задачу показу, що вони непослідовні, тоді ми не знатимемо, що мається на увазі: коли ~ *A1* випливає з *A2*, *A3*, *A4*, …*An*, чи коли ~ *A2* випливає з *A1*, *A3*, …, *An*, чи коли ~ *A3* випливає з *A1*, *A2*, *A4*, …, *An* і т. д. Важливо наголосити, яка фраза є цільовим твердженням, оскільки перед аналізом усі пункти рівні.

У прикладі про професора Івана, легко побачити, як можна отримати пусту фразу, якщо ми додамо твердження:

:- професор (іван). (7.5)

оскільки ми вже бачили, як фразу

професор (іван); професор (іван):- (7.6)

було отримано з гіпотез. Аналізуючи (7.5) і (7.6), отримуємо:

професор (іван):-. (7.7)

Аналіз (7.6) і (7.7) дає нам:

:-.

Отже, аналіз довів, що Іван є професором.

Завершеність аналізу означає, що, якщо факти випливають з наших гіпотез, ми можемо довести їх вірність через використання аналізу. Коли ми говоримо, що аналіз зможе отримати пусту фразу, ми маємо на увазі, що існує послідовність дій (кожна з яких включає аксіоми чи фрази, виведені в попередніх діях), які закінчуються виведенням специфічної фрази. Хоча аналіз і дозволяє нам отримувати наслідки з двох фраз, але він ніяк не вказує на те, де шукати наступні відповідності. Звичайно, якщо ми маємо багато гіпотез, то буде і багато варіантів. І кожна з виведених фраз збільшуватиме цю кількість. Більшість з варіантів будуть невірними, і якщо ми не будемо уважними, то витратимо безліч часу на обробку невірних варіантів. Можемо навіть просто не знайти вірного варіанта.

Було запропоновано чимало вдосконалень до процесу аналізу з даного приводу. Розглянемо деякі з них. Наприклад, удосконалення для випадку фраз Хорна. Нагадаємо, що фраза Хорна має не більше одного позитивного літерала. Тобто можуть існувати два типи фраз Хорна: з одним позитивним літералом або без жодного позитивного літерала. Ось їх приклади:

молодий_спеціаліст (X):- вища_освіта (X), знання_іноз_мови (X).
:- молодий_спеціаліст (X).

Насправді, коли ми уявляємо множину фраз Хорна (включаючи цільові твердження), то вони всі, крім однієї, мають бути повними. Будь-яка теорема, виражена через хорнівські фрази, може бути доведена лише тоді, коли вона містить тільки одну неповну фразу. Оскільки ми самі вирішуємо, які фрази є метою, ми можемо вважати метою неповну фразу, а всі інші — гіпотезами. Це має природну причину. По-перше, легко побачити, що має бути принаймні одна неповна фраза, щоб задача мала розв'язок. Це випливає з того, що результатом аналізу двох повних хорнівських фраз завжди є повна фраза. По-друге, якщо існує кілька неповних фраз у наших аксіомах, то доведення аналізом отримання пустої фрази потребує лише однієї з них.

Спробуємо сформулювати наведені вище міркування у формалізованішому вигляді.

Згідно з [179] доведенням теореми називається отримання відповіді на запитання: чи випливає логічна формула B із заданого набору формул $\{F_1, \ldots, F_n\}$. Це еквівалентно доведенню загальнозначимості формули $F_1, \ldots, F_n\ B$ або суперечливості формули $(F_1, \ldots, F_n) \sim B$. З практичних міркувань зручніше доводити суперечливість формули $(F_1, \ldots, F_n) \sim B$, тобто йдеться по суті про доведення "від супротивного".

7.5. Побудова теорії певної області знань

Побудова теорії певної області знань включає [172] дослідження структури цієї області і вибір позначень, що характеризують особливості даної структури. Потім будується множина вірно побудованих формул (ВПФ) для опису цієї структури. Множина ВПФ є теорією цієї області знань, в якій кожна ВПФ — аксіома.

Дослідження області знань включає відокремлення вагомих суттєвостей з цієї області. Дану множину називають "область інтерпретації". Далі визначаються найважливіші функції над елементами області інтерпретації, якщо такі існують, і значимі відношення між елементами області інтерпретації. По закінченні значимі відношення оформлюються синтаксично у вигляді аксіом.

Під функцією розумітимемо відображення n елементів з області інтерпретації (де n — арність функції) на один з елементів цієї області.

Нехай областю дослідження є службові стосунки між людьми у певній фірмі. Областю інтерпретації буде така множина людей:

(*Іван; Ігор, підлеглий Івана; Петро, підлеглий Івана; Микола, друг Петра*).

На цій області можна виокремити унарну функцію "друг". Наприклад,

друг (*Петра*) → *Микола*.

Відношенням називають відображення n елементів із області інтерпретування на елемент множини **(істина, хибність)**. У нашому прикладі можна виокремити бінарне відношення "підлеглий". Так, *підлеглий (Івана, Ігор)* матиме значення "істина", а *підлеглий (Ігор, Микола)* набуде значення "хибність". Якщо на певному конкретному наборі аргументів відношення буде істинним, тоді кажуть, що відношення справджується.

Після фіксації області інтерпретування переходять до вибору позначень елементів області: констант, функцій, відношень. Відмітимо, що функція сама по собі не може мати значення — його може мати тільки конкретне використання функції. Аналогічне можна сказати і про відношення.

Проілюструємо це на такому прикладі.

Спочатку виберемо позначення для констант і присвоїмо їм значення, які відповідають елементам області інтерпретації.

Значенням *a* буде *Іван*.

Значенням *b* буде *підлеглий Івана Ігор*.

Значенням *c* буде *підлеглий Івана Петро.*
Функцію *"друг"* позначимо *f*, тоді семантика цієї функції визначиться:
Значенням *f* (*c*) буде *Микола.*

Якщо позначимо через *P* введене нами відношення *"підлеглий"*, тоді його семантика виражатиме:

Значенням *P* (*a, b*) буде ***істина.***
Значенням *P* (*b, a*) буде ***істина.***
Значенням *P* (*c, a*) буде ***істина.***
Значенням *P* (*a, c*) буде ***істина.***
Значенням *P* (*b, c*) буде ***хибність.***
Значенням *P* (*c, b*) буде ***хибність.***
Значенням *P* (*b, f* (*c*)) буде ***хибність.***
Значенням *P* (*f* (*c*), *b*) буде ***хибність.***
Значенням *P* (*a, f* (*c*)) буде ***хибність.***
Значенням *P* (*f* (*c*), *a*) буде ***хибність.***
Значенням *P* (*c, f* (*c*)) буде ***хибність.***
Значенням *P* (*f* (*c*), *c*) буде ***хибність.***

З даних атомарних формул маємо можливість будувати ВПФ.

Інтерпретація, яка робить ВПФ істиною, називається моделлю цієї ВПФ. Аналогічно визначається і модель теорії. Про ВПФ, або теорію, яка набуває значення "істина" хоча б на одній інтерпретації, кажуть, що вона задовільна. Якщо ВПФ, або теорія, є хибною на всіх інтерпретаціях, тоді її називають незадовільною або непослідовною.

Підведемо деякі підсумки побудови бази знань певної предметної області. Вони будуть сформульовані у вигляді таких досить загальних принципів.

1. Необхідно визначити предметні змінні, предметні константи і предикати, якими описуватиметься база знань. Цей процес має досить неформальний характер і часто вимагає значної винахідливості. Кожний проектувальник повинен ретельно продумати концептуальну структуру бази знань і вибрати з кількох можливих варіантів найоптимальніший.
2. Базу знань часто вдається побудувати без явного виписування пренексної нормальної форми; якщо ж це викликає ускладнення, слід застосувати відповідні тотожні перетворення.
3. База знань описується як кон'юнкція деяких тверджень, які вважаються істинними; твердження може бути явним фактом або правилом виведення.
4. Факти задаються позитивним або негативним літералом, який не містить змінних.
5. Правила виведення задаються фразами Хорна; передумови імплікації задаються негативними літералами, а наслідок — позитивними.
6. Квантори існування усуваються шляхом введення констант і функцій Сколема.

7. Після усунення кванторів існування механічно усуваються квантори узагальнення.
8. Після побудови формалізованої бази знань можна здійснювати логічне виведення на основі методу резолюцій.

Наведемо *приклад*. Нехай маємо такий неформальний опис:

Усі студенти люблять відвідувати лекції. Деякі студенти люблять морозиво. Іванов є студент, але не любить морозива.

Можна ввести, наприклад, такі предикати та предметні константи:

$L(x, y)$ — x любить y.
$S(x)$ — x є студентом.
A — предметна константа, яка означає "відвідування лекцій";
I — предметна константа, яка відповідає Іванову;
Q — предметна константа, яка означає морозиво.

Отримуємо базу знань:

$\forall x: (\sim S(x) \vee L(x, A))$
$\exists x: (S(x) \,\&\, L(x, Q))$
$S(I)$
$\sim L(I, Q)$.

Усунення кванторів і розподіл кон'юнкцій приводить до такої бази знань:

$\sim S(x) \vee L(x, Q)$
$S(c)$
$L(c, Q)$
$S(I)$
$\sim L(I, Q)$.

Тут c — константа Сколема.

7.6. Від формальної логіки до логічного програмування

Як ми вже зазначали, в логіці предикатів існують методи доведення того, буде чи ні конкретна ВПФ наслідком деякої теорії. Природно виникає бажання автоматизувати таке доведення за допомогою числення предикатів. У більшості підходів до цієї проблеми використовується процедура спростування. Розглянемо її основні ідеї.

З визначення тотожної істинності і непослідовності можна дійти висновку, що ВПФ буде тотожно істинною тоді і тільки тоді, коли внесення в теорію заперечення даної ВПФ перетворить цю теорію на непослідовну.

Теорія буде непослідовною, якщо в ній можливо вивести протиріччя такого вигляду: $A \,\&\, \sim A$.

Для будь-якої системи логічного програмування характерною є та обставина, що для виконання програми (побудови виведення результату) використовується вмонтована система автоматичного пошуку. Як вже зазначалося, механізм пошуку логічного висновку бере свій початок від методу

резолюцій Робінсона. Описане в п. 7.4 правило резолюції виведення логічного висновку можна уточнити таким чином. Дві фрази можуть резольвувати між собою, якщо одна з них має позитивний, а друга — негативний літерал з одним і тим самим позначенням предиката та однаковою кількістю аргументів, і якщо аргументи обох літералів можуть бути уніфіковані (погоджені).

Розглянемо дві фрази спеціального вигляду:

$P(a)$ — заключення без умови і $\sim P(a)$ — умова без заключення. Нагадаємо, що наявність цих двох фраз в одній теорії є протиріччям. Якщо вони резольвують між собою, тоді отримана резольвента називається порожньою фразою.

Якщо при резолюції двох фраз, що входять до складу теорії, отримується порожня фраза, тоді теорія буде непослідовною.

Розглянемо формальніше застосування принципу резолюції та уніфікації. Спочатку розглянемо варіант вживання правила резолюції для атомарних формул, які не містять змінних, але можуть набувати значень "істинність" або "хибність" (фактично — для висловлювань). Кажуть, що дві фрази містять ***контрарну пару*** атомарних формул (або просто контрарну пару), якщо одна з них включає деяку атомарну формулу без заперечення, а інша — ту саму формулу під знаком заперечення.

Дві фрази, що містять контрарну пару, можуть бути резольвовані одна з іншою, і результатом резолюції (резольвентою) є ***диз'юнкція літералів****, які залишаються в обох фразах після викреслення контрарної пари.*

Наприклад, нехай ми маємо фрази:

$\sim P \vee M$

$P \vee L,$

де M і L — диз'юнкції довільної кількості атомарних формул. Фрази є резольвованими, і резольвента має вигляд $M \vee L$.

Це правило було запропоновано Девісом і Патнемом. Воно стає очевиднішим, якщо переписати його в термінах імплікацій (спробуйте зробити це самостійно).

Дж. Робінсон розширив правило Девіса і Патнема на випадок, коли аргументами атомарних формул можуть бути змінні, константи і взагалі довільні терми. Основна ідея полягає у підстановці до атомарних формул, що входять до контрарної пари, термів замість змінних, поки набори аргументів обох формул не стануть однаковими (не будуть ***уніфіковані***). Відповідна підстановка називається ***уніфікатором***.

Слід зауважити, що змінну не можна замінювати на терм, який містить ту саму змінну. З іншого боку, фрази теорії є незалежними між собою. Тому перед застосуванням методу резолюцій рекомендується перейменувати змінні так, щоб кожна змінна зустрічалася не більше ніж в одній фразі теорії.

Формальніше, результатом підстановки $\sigma = \{t_1/x_1, \ldots, t_n/x_n\}$ до фрази M (записується $M\sigma$) є фраза, утворена з M заміною змінних x_i на терми t_i. Зверніть увагу: **терми можна підставляти лише замість змінних (не замість констант!)**.

Підстановка σ називається ***уніфікатором*** для множини виразів $\{M_1, \ldots, M_n\}$, якщо $M_1\sigma = M_2\sigma = \ldots = M_n\sigma$.

Уніфікатор σ для множини виразів $\{M_1, \ldots, M_n\}$ називається ***найбільш загальним уніфікатором***, якщо для будь-якого уніфікатора θ цієї множини існує така підстановка λ, що θ являтиме собою композицію підстановок σ і λ.

Композицією підстановок $\sigma = \{t_1/x_1, \ldots, t_n/x_n\}$ і $\lambda = \{u_1/y_1, \ldots, u_q/y_q\}$ називається підстановка $\{t_1\lambda/x_1, \ldots, t_n\lambda/x_n, u_1/y_1, \ldots, u_q/y_q\}$, з якої викреслені всі $t_i\lambda/x_i$, якщо $t_i\lambda = x_i$, і всі u_i/y_i такі, що $y_i \in \{x_1, \ldots, x_n\}$.

Тепер можна сформулювати загальне правило резолюцій.

Дві фрази, що містять контрарну пару, можуть бути резольвовані одна з іншою, якщо літерали, що входять до контрарної пари, можуть бути уніфіковані, тобто якщо для них існує найбільш загальний уніфікатор σ. Результатом резолюції є ***диз'юнкція літералів****, які залишаються в обох фразах після викреслення контрарної пари, причому до цих літералів повинен бути застосований уніфікатор σ.*

Існують формальні алгоритми, які дозволяють отримувати найбільш загальний уніфікатор [121, 272, 320]. Але у більшості випадків достатньо деяких неформальних простих правил, подібних до тих, що були сформульовані в [172].

1. **Будь-який терм зіставляється сам з собою**. Наприклад, дві фрази

$P(a) \vee \sim Q(b, c)$,
$Q(b, c) \vee \sim R(b, c)$

є резольвовані, і резольвента записується як $P(a) \vee \sim R(b, c)$.

2. **Різні константи не зіставляються одна з одною**, тому фрази

$P(a) \vee \sim Q(b, c)$,
$Q(c, c) \vee \sim R(b, c)$

не є резольвованими.

3. **Змінна може бути замінена константою або іншим термом**. Так, фрази

$P(a) \vee \sim Q(a, b)$,
$Q(x, y) \vee \sim R(x, y)$

резольвується, при цьому змінна x зіставляється з константою a, змінна y — з константою b. У резольвенті $P(a) \vee \sim R(a, b)$ змінні замінені на константи, з якими вони зіставлялися.

Якщо в процесі виведення утворилися фрази $P(a)$ та $\sim P(a)$, їх резолюцією є **пустий** диз'юнкт (тотожна хибність, позначається $\square$). **Саме виведення пустого диз'юнкта і означає суперечливість теорії**.

Тепер можна сформулювати алгоритм **автоматичного доведення теорем на основі методу резолюцій**. Нагадаємо, що йдеться про перевірку істинності будь-якого твердження в рамках існуючої системи знань.

Застосовується метод доведення "від супротивного". При перевірці утворюється деяка пробна теорія, що утворюється **шляхом додавання заперечення нового твердження до вже існуючих** (сама база знань при цьому вважається несуперечливою). Послідовно застосовується правило резолюції, і утворені резольвенти додаються до пробної теорії. Якщо при цьому врешті утворюється пуста фраза, то твердження, яке перевіряється, істинне.

Фундаментальною властивістю методу резолюцій є його ***повнота***. Відповідна теорема формулюється так [129]: **множина диз'юнктів *S* суперечлива тоді і тільки тоді, коли з неї можна вивести пустий диз'юнкт.**

Як уже зазначалося, логіка предикатів першого порядку не є розв'язною. Тому для множини диз'юнктів, яка не є суперечливою, процедура, що ґрунтується на методі резолюцій, може працювати нескінченно довго.

Наведемо ще приклади застосування методу резолюції.

Приклад 7.1.

Нехай дано такий набір фраз:

вовк їсть м'ясо,
тварини, які їдять м'ясо, є хижаками.

Потрібно довести, що:

вовк є хижаком.

Введемо предикати $G(x, y)$, який означає *"x їсть y"* і $Q(x)$, який означає *"x є хижаком"*. Далі введемо предметні константи: W — вовк; M — м'ясо.

Тоді маємо базу знань:

$G(W, M)$

$\sim G(x, M) \vee Q(x)$

Потрібно довести формулу $Q(W)$.

Крок 1. Утворюємо пробну теорію та додаємо до неї заперечення:

$G(W, M)$ (7.8)

$\sim G(x, M) \vee Q(x)$ (7.9)

$\sim Q(W)$ (7.10)

Крок 2. Застосовуємо резолюцію до (7.9) та (7.10); при цьому константа W зіставляється зі змінною x:

$G(W, M)$ (7.8)

$\sim G(x, M) \vee Q(x)$ (7.9)

$\sim Q(W)$ (7.10)

$\sim G(W, M)$ (7.11)

Крок 3. Внаслідок резолюції (7.8) і (7.11) утворюється пуста фраза. Це означає суперечливість пробної теорії та істинність твердження, яке перевіряється.

Приклад 7.2:

Нехай дана база знань, що складається з фраз:

Володимир є сином Костянтина,
Костянтин є сином Олега

та правил:

Якщо x є сином y, то x є нащадком y,
Якщо x є нащадком y, а y є нащадком z, то x є нащадком z.

Необхідно довести, що:

Володимир є нащадком Олега.

Введемо предметні константи V — Володимир, K — Костянтин, O — Олег і предикати: $S(x, y)$ — x є сином y; $N(x, y)$ — x є нащадком y.

Крок 1. Утворюємо пробну теорію шляхом додавання заперечення твердження, що перевіряється:

$$S(V, K) \tag{7.12}$$
$$S(K, O) \tag{7.13}$$
$$\sim S(x, y) \vee N(x, y) \tag{7.14}$$
$$\sim N(x, y) \vee \sim N(y, z) \vee N(x, z) \tag{7.15}$$
$$\sim N(V, O) \tag{7.16}$$

Крок 2. Застосовуємо резолюцію до (7.16) і (7.15); при цьому x зіставляється з V, а z — з O:

$S(V, K)$
$S(K, O)$
$\sim S(x, y) \vee N(x, y)$
$\sim N(x, y) \vee \sim N(y, z) \vee N(x, z)$
$\sim N(V, O)$
$$\sim N(K, y) \vee \sim N(y, O) \tag{7.17}$$

Крок 3. Застосовуємо резолюцію до (7.14) і (7.17); в результаті до теорії додається твердження

$$\sim S(V, y) \vee \sim N(y, O) \tag{7.18}$$

Крок 4. Застосовуємо резолюцію до (7.14) і (7.18); додається твердження

$$\sim S(V, y) \vee \sim S(y, O) \tag{7.19}$$

Крок 5. Застосовуємо резолюцію до (7.19) і (7.12); y зіставляється з K:

$$\sim S(K, O) \tag{7.20}$$

Крок 6. У результаті резолюції (7.13) і (7.20) утворюється порожня фраза.

Висновок: твердження, що перевіряється, є істинним.

Приклад 7.3.

Повернемося до бази знань, що складається з фраз:

Усі студенти люблять відвідувати лекції.
Деякі студенти люблять морозиво.
Іванов є студент, але не любить морозива.

Ми ввели такі предикати та предметні константи:

$L(x, y)$ — x любить y.

$S(x)$ — x є студентом.

A — предметна константа, яка означає "відвідування лекцій";

I — предметна константа, яка відповідає Іванову;

Q — предметна константа, яка означає морозиво.

База знань має вигляд:

$\sim S(x) \vee L(x, A)$

$S(c)$

$L(c, Q)$

$S(I)$

$\sim L(I, Q)$.

Спробуємо довести, що деякі люди, які люблять морозиво, люблять і відвідувати лекції.

Остання фраза запишеться у вигляді $\exists x$: $(L(x, A) \,\&\, L(x, Q))$. Її заперечення має вигляд $\forall x$: $(\sim L(x, A) \vee \sim L(x, Q))$. Позбувшися квантора узагальнення, отримуємо фразу $\sim L(x, A) \vee \sim L(x, Q)$.

Подальші резолюції здійснюються звичайним чином. Спробуйте виконати їх самостійно.

У літературі також вживаються такі терміни, як **лінійна і вхідна резолюції**. Пояснимо їх застосування.

Метод резолюцій є перебірною процедурою. Якщо спробувати безпосередньо застосовувати метод резолюцій у тому вигляді, в якому він був сформульований, то для великих баз знань він породжуватиме значну кількість безперспективних резолюцій і тому виявляється досить неефективним. Для запобігання "експоненційного вибуху" були запропоновані ефективніші модифікації [129]: ***семантична резолюція, лок-резолюція, лінійна резолюція***. Суть цих модифікацій полягає у введенні тих чи інших критеріїв, за якими відбираються диз'юнкти, що беруть участь у черговій резолюції. Зазначені модифікації в цілому не усувають бектрекінгового перебору, але дозволяють істотно скоротити цей перебір і роблять метод резолюцій цілком придатним для практичного застосування.

Для даної множини диз'юнктів S і початкового диз'юнкта ***лінійним виведенням*** *диз'юнкта C_n називається* [129, 272] *послідовність $C_0, C_1, \ldots, C_n$, де*

1) *для $i = 0, 1, \ldots, n-1$ диз'юнкт C_{i+1} являє собою резольвенту диз'юнкта C_i (який називається* ***центральним диз'юнктом****) і B_i (який називається боковим диз'юнктом), при цьому:*
2) *будь-який боковий диз'юнкт B_i або належить S, або є C_j для деякого $j < i$.*

Лінійна резолюція є повною.

Важливим варіантом лінійної резолюції є так звана ***вхідна резолюція***, для якої за бокові диз'юнкти беруться тільки диз'юнкти з множини S. Вхід-

на резолюція є простою та ефективною, і саме вона лежить в основі механізму логічного виведення, що реалізовано в мові Пролог.

У загальному випадку вхідна резолюція не є повною. Але повнота вхідної резолюції гарантується, якщо обмежитися тільки хорнівськими диз'юнктами.

Нарешті, видається зручним такий неформальний опис лінійної, зокрема вхідної, резолюції [172].

Алгоритм доведення теорем на основі методу резолюцій повинен бути сфокусований на наслідках внесення до пробної теорії нових резольвент. Це досягається за допомогою таких двох правил:

- **у першій резолюції слід використовувати щойно додане заперечення твердження, яке перевіряється;**
- **кожна наступна резолюція повинна використовувати попередній результат.**

7.7. Мова Пролог і логічне програмування

Розглянемо відповідність між численням предикатів і Прологом [26]. Як ми вже бачили, деякі фрази виглядають точно так, як фрази Прологу, а інші дещо інакше. Зокрема, запитання Прологу:

?- *A*1, *A*2, ..., ..., ..., *An*

означає точно те саме, що й неповна хорнівська фраза:

:- *A*1, *A*2, ..., ..., ..., *An*.

Система виведення Прологу базується на засобах аналізу хорнівських фраз. Конкретна стратегія, яку використовує ця мова, є формою аналізу постійного входу. Коли застосовується дана стратегія, вибір, що з чим аналізувати, проводиться за таким принципом.

Починаємо з цільового твердження та аналізуємо його разом з однією з аксіом. Отриману фразу знову аналізуємо з однією з аксіом і т. д. Ми завжди аналізуємо лише щойно отриману фразу разом з однією з первинних хорнівських фраз. У Пролозі щойно отриману фразу можна розглядати як послідовність цілей, що мають бути досягнуті. Все починається із запитання і закінчується (бажано) пустою фразою. На кожному кроці побудови виведення ми знаходимо фразу, початок якої збігається з однією з цілей, зв'язуємо потрібні змінні, обираємо мету, яка збігається, і додаємо зміст зв'язаної фрази до набору цілей, що мають бути досягнуті.

Наприклад, від:

:- *мати* (*іван, X*), *мати* (*X, Y*).

та

мати (*U, V*):-*батько* (*U, V*), *жінка* (*V*).

переходимо до

:- *батько* (*іван, X*), *жінка* (*X*), *мати* (*X, Y*).

Насправді стратегія доведення у Пролозі ще більш обмежена, ніж просто аналіз лінійного входу. В цьому прикладі ми вирішили звести перший літерал у цільовому пункті, але могли вибрати й інший.

Варто також наголосити, як Пролог вишукує альтернативні фрази для досягнення тієї самої мети. В основному Пролог використовує стратегію пошуку "в глибину" частіше, ніж "в ширину". Тобто, він оперує лише з однією альтернативою одночасно, вважаючи цей вибір вірним, доки не буде доведено зворотне. Для кожної мети він вибирає фрази в затвердженій послідовності і звертається до наступних лише після доведення невірності попередніх. Альтернативною до цієї стратегії є стратегія пошуку "в ширину", до якої Пролог звертається пізніше. Вона має перевагу в тому, що забезпечує перебір усіх варіантів, у той час як пошук "в глибину" може легко зупинитись у глухому куті. З іншого боку пошук "в глибину" дозволяє перевіряти варіанти до кінця і є простішим в оперуванні, до того ж займає менше ресурсів комп'ютера.

І, нарешті, кілька слів про те, в яких випадках у Пролозі відповідність двох фраз може відрізнятись від класичного аналізу. Більшість варіантів Прологу дозволить вам досягнути такої мети:

рівність (*X*, *X*).
? — рівність (*f*(*Y*), *Y*).

тобто дасть змогу умові відповідати складовій частині самої себе, тобто матиме місце рекурсивний виклик. У цьому прикладі $f(Y)$ зрівнюється з Y, що стоїть всередині $f(f(Y))$, яке перетворюється на $f(f(f(Y)))$ і т. д. Пролог дозволить вам надрукувати і запустити таку програму, але навряд чи ви отримаєте відповідь. Згідно з правилами уніфікації такі випадки не повинні траплятись. Ось чим відрізняється поведінка Прологу. В деяких версіях з'явиться попередження, що такого робити не можна, але оскільки такі випадки трапляються рідко, то більшість розробників програми випускали цю можливість.

Програмування на Пролозі має декларативну основу: ми повідомляємо про істинні твердження (аксіоми) і просимо перевірити істинність інших тверджень (теорем). Ідея того, що програмування має бути саме таким, виникла давно і привела до розробки практичної можливості логічного програмування — створення мов логічного програмування. Переваги логічного програмування полягають у тому, що програми легше прочитати, і вони не міститимуть описів того, як потрібно виконувати дії, скоріше там буде описано, як виглядатиме рішення. Тим більше, якщо програма схожа на інструкцію по досягненню мети, то в ній легше виявити дефекти. Ми будемо бачити, що робить програма, а не як вона це робить.

7.8. Основні ідеї Прологу

Спочатку коротко підсумуємо викладені роздуми. Отже, програма на Пролозі може складатись з набору фактів разом з набором умов, яким має задовольняти розв'язок, і комп'ютер має сам вивести рішення з даних фактів. Такий підхід до програмування називається логічним програмуванням. Пролог опирається на формальну логіку так само, як FORTRAN,

BASIC базуються на арифметиці та простій алгебрі. Для виконання завдань Пролог використовує техніку, розроблену для доведення теорем у логіці.

Пролог є дуже різнобічною мовою. Зауважимо, що Пролог можна застосовувати для програмування всіх видів алгоритмів, не тільки для тих, для яких він був спеціально створений. Використання Прологу не пов'язане з якимось одним класом алгоритмів, логікою програми чи форматом файлів. Тому Пролог не менш потужний, ніж Pascal, C або C++, а у багатьох аспектах навіть потужніший. Доцільність його застосування, очевидно, має визначатися конкретною ситуацією.

Перші повідомлення про Пролог з'явились на початку 70-х років. Він належить до класу логічних мов програмування, основні ідеї розробки яких запропонували Р. Ковальські і П. Хейс. Перший інтерпретатор Прологу був розроблений у Марселі (Франція) під керівництвом А. Колмерое в 1973 р. Наступна версія, виконана Д. Уореном — Единбурзька реалізація Прологу на машині DEC-10 — перевела Пролог і разом з ним логічне програмування з площини теоретичних досліджень у площину практичного програмування, зробила його корисним інструментом для вирішення різних задач штучного інтелекту.

Для Прологу характерним є і той факт, що програміст повинен мислити в термінах цілей. Що під цим розуміється? На відміну від традиційних мов програмування, Пролог вимагає від програміста змінити форму мислення по написанню програм. Прологівська програма являє собою набір визначень ситуацій і формулювань задач замість того, щоб детально описувати варіанти рішень цих задач. Основою Прологу є обмежений, але на диво потужний і гнучкий набір програмних механізмів, який включає: зіставлення зразків, задання структур даних типу дерева та автоматичне повернення. Назва "Пролог" утворилась як скорочення від "програмування в термінах логіки". Отже, "Пролог" можна віднести до мов програмування, які будуються на описовому або декларативному підході до програмування.

Можна виокремити два рівні характеристики прологівської програми: декларативний і процедурний. Перший з них визначає, яким повинен бути результат роботи програми, за допомогою відношень. Другий — як цей результат був отриманий і які з відношень реально оброблялись. Пролог-системи значну частину процедурних деталей виконують самостійно, без втручання програміста. Останній, таким чином, може більше уваги приділяти декларативному аспекту програми, не відволікаючись на організацію процесу обчислень, якщо його не хвилює питання ефективності цих обчислень.

Спочатку Пролог належав до теоретичних мов програмування і в основному використовувався як інструментарій у наукових дослідженнях. На це впливало і те, що довгий час учені із США не сприймали його переваг для вирішення задач штучного інтелекту. Дж. Малпас пояснює цей факт тим, що, по-перше, серед учених США сильними залишались лісповські традиції (мова Ліспа створена в Массачусетському технологічному інституті) і, по-друге, — попереднє знайомство з мовою логічного типу Мікро-

пленнер було невдалим. Остання реалізовувалася дуже неефективно. Та зі створенням швидких інтерпретаторів і компіляторів Пролог посів почесне місце не тільки серед найвживаніших мов вирішення задач штучного інтелекту, а й серед мов, які використовуються спеціалістами в галузях реляційних баз даних, програмної інженерії, задання знань, експертних системах і багатьох інших.

Резюмуючи сказане, можна відмітити такі переваги Прологу:

- Пролог має чітку математичну основу, дуже близьку до людського мислення.
- Використання єдиної мови специфікацій (числення предикатів) для опису вимог до програм і опису самої програми на Пролозі дозволяє поєднувати процес написання програми з її верифікацією.
- Застосування відношення як базового поняття мови дає змогу зручно працювати з реляційними базами даних.
- Паралельний принцип організації обчислень дозволяє просто і природно реалізовувати Пролог-програму на паралельному комплексі.
- Пролог підтримує обчислення, які базуються на пошуку через зворотний ланцюжок міркувань. Це забезпечує ефективність використання Прологу при побудові експертних систем.
- Пролог може зберігати за допомогою "логічних змінних" проміжні результати обчислень для подальшого використання. Це дозволяє природно вирішувати проблему організації логічного виведення.

Поряд із зазначеними перевагами спеціалісти відзначають такі вади Прологу.

- Складність розуміння процесу виконання програми на Пролозі, пов'язаної з "невидимим" порядком побудови виведення результату програмою.
- Погані засоби для вияву екстралогічних властивостей (оператори динамічного приєднання і вилучення тверджень).
- Відсутність досконалих засобів для розробки і відлагодження великих програм.
- Недостатні засоби підтримки модульного принципу програмування.

У комерційних цілях Пролог часто використовують у створенні експертних систем, інтелектуальних баз даних і програм, з якими можна спілкуватись звичайною мовою [175].

Пролог має багато спільного з Ліспом [267]— мовою, яку традиційно застосовують у дослідженнях зі штучного інтелекту. Обидві мови роблять досить простим виконання складних обчислень над складними даними. Крім того, вони досить потужні, щоб елегантно запрограмувати складні алгоритми. І Пролог, і Ліспі використовують динамічне виділення пам'яті, тому нема потреби декларувати змінні перед тим, як їх застосовувати. Ці мови також дозволяють програмам перевіряти та модифікувати самих себе, тому програма може "вчитись" на тій інформації, яку вона отримує під час своєї роботи.

Головна різниця полягає в тому, що Пролог має процедуру, яка здатна до "міркування" — машину логічного виведення, вбудовану в нього, тоді як Ліспо цього не має. Тому програми, які виконують якісь логічні роздуми, легше писати на Пролозі, ніж на Ліспі. Якщо вбудована машина логічного виведення не підходить для конкретної задачі, програміст дуже часто може використовувати частину цього механізму, переписавши все інше. В Ліспі, навпаки, якщо потрібна машина логічного виведення, програміст сам має її забезпечити.

Чи Пролог об'єктно-орієнтований? Не зовсім. Пролог — інше, нове і різнопланове вирішення проблеми, яку об'єктно-орієнтовані мови повинні були розв'язати. Пролог дозволяє говорити про властивості та зв'язки прямо, замість того, щоб добиратись до них через механізм наслідування.

7.9. Як працює Пролог

Потужність Прологу безпосередньо пов'язана з використанням інтерпретації логіки за допомогою процедур, тобто представленні знань за допомогою визначення процедур, після чого міркування стає простим процесом виклику необхідної процедури. Щоб зрозуміти, яким чином це відбувається, розглянемо такі два набори інформації:

(1) *Для будь-якого X, якщо X в Чернігівській_ області, тоді X в Україні*

(2) *Остер в Чернігівській_ області*.

Такі набори називають **базою знань**. Будемо називати елемент (1) **правилом**, оскільки він дає нам можливість логічно вивести з однієї порції інформації іншу, а елемент (2) — **фактом**, оскільки він не залежить від іншої інформації. Зверніть увагу, що правило містить "якщо", а факт — ні. Факти і правила — два типи речень. Факт не обов'язково має бути істиною з реального життя. Якщо ви скажете, що місто Остер є на Полтавщині, Пролог вам повірить.

Припустимо, ми хочемо знати, чи знаходиться місто Остер в Україні. Зрозуміло, що (1) і (2) можна зв'язати і відповісти на це запитання. Але як це зробити на комп'ютері? Треба виразити (1) і (2) як визначення процедур:

(1′) *Щоб довести, що X в Україні, треба довести, що X в Чернігівській області.*

(2′) *Щоб довести, що Остер в Чернігівській_ області, нічого не треба робити.*

Поставимо запитання у вигляді такої команди:

Довести, що Остер в Україні.

Це викликає процедуру (1′), яка, в свою чергу, викликає процедуру (2′). Остання повертає відповідь "так".

Пролог має власні позначення для представлення знань. Наша проста база знань може бути представлена у Пролозі таким чином:

в_україні (*X*):- в_чернігівській_області (*X*)
в_чернігівській_області (остер)

Тут **в_україні** та **в_чернігівській_області** — предикати, які визначають окремі особливості. Предикат може мати будь-яку фіксовану кількість аргументів (параметрів). Одномісний предикат показує властивість одного поняття, тоді як двомісні описують взаємозв'язок між двома поняттями.

Кількість аргументів, які має предикат, називається арністю (від таких термінів як унарність, бінарність, тернарність та інших подібних). Два різних предикати можуть мати однакову назву, якщо вони різної арності, але бажано намагатись уникати такої практики, оскільки вона призводить до непорозумінь.

На сьогоднішній день є багато різних реалізацій Прологу. Прикладом можуть служити Турбо Пролог, Мікро Пролог, Arity Prolog і ALS Prolog на IBM PC та Quintus Prolog на робочих станціях Sun, SWI Prolog, LPA Prolog, Cogent (Amzi) Prolog та Expert Systems Limited's Public Domain Prolog-2.

Протягом багатьох років стандартом де-факто для Прологу була мова, описана Клоксіном (Clocksin) та Меллішом (Mellish) в їх популярному підручнику "Програмування на Пролозі" ("Programming in Prolog"). Це, практично, мова, яку вживав ще на DEC-10 Уорен та його колеги наприкінці 70-х років, яку часто називають единбурзькою, або Edinburg Prolog, або DEC-10 Prolog. Більшість комерційних версій Прологу розробляються так, щоб бути сумісними з цією версією.

У 1995 р. Міжнародна організація стандартизації (ISO) опублікувала міжнародний стандарт для мови Пролог (Scowen 1995). ISO Пролог дещо схожий на Edinburg Prolog, але з дещо розширеними можливостями.

Нагадаємо також, що такі версії, як Turbo Prolog (PDC Prolog), Colmerauer's Prolog-II та Prolog III, Turbo Prolog мають можливість декларувати типи даних. Як результат — програми можуть виконуватись швидше, але це дуже заважає їм перевіряти та модифікувати себе. Colmerauer's Prolog-II та Prolog-III — мови обмеженого логічного програмування, тобто вони дозволяють накладати обмеження на значення змінних перед тим, як присвоювати їм якісь значення. Це робить доступним використання нової техніки.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Сформулюйте задачу автоматичного доведення теорем.
2. Дайте визначення логічної моделі.
3. Дайте визначення предиката.
4. Які основні елементи використовуються в численні предикатів?
5. Що таке квантори узагальнення та існування?
6. Дайте визначення атомарної формули.
7. Дайте визначення диз'юнкта.
8. Що таке фраза Хорна?
9. Поясніть зв'язок фраз Хорна з імплікаціями.
10. Дайте визначення пренексної нормальної форми.
11. Яким чином можна усунути квантори узагальнення та існування?

12. Що таке константи і функції Сколема?
13. Перелічіть основні дії, необхідні для приведення логічних формул до стандартної фразової форми.
14. Сформулюйте правило резолюцій. Наведіть власні приклади застосування цього правила.
15. Що означає виведення пустого диз'юнкта? Яким чином він може бути отриманий?
16. Опишіть алгоритм перевірки тверджень на основі методу резолюцій.
17. Що означає поняття повноти для методу резолюцій?
18. З чим пов'язана необхідність модифікації загального методу резолюцій?
19. Охарактеризуйте лінійну і вхідну резолюції. Покажіть, що вони не усувають бектрекінгового перебору.
20. Чи є вхідна резолюція повною? Наведіть клас диз'юнктів, для яких вона є повною.
21. Охарактеризуйте поняття логічного програмування.
22. Назвіть найпоширенішу мову логічного програмування.
23. Поясніть взаємозв'язок прологівських фактів і правил з фразами Хорна.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Розберіть вбудований прологівський механізм логічного виведення. Поясніть його зв'язок з бектрекінговими алгоритмами.
2. Наведіть приклади застосування логічного програмування для вирішення конкретних практичних задач.
3. Порівняйте Пролог з процедурними та об'єктно-орієнтованими мовами програмування.

ЗАДАЧІ І ВПРАВИ

1. Доведіть тотожності, які встановлюють зв'язок імплікацій з фразами Хорна:

   а) $(x \to y) = \bar{x} \vee y$;

   б) $((x \,\&\, y \,\&\, z) \to u) = \bar{x} \vee \bar{y} \vee \bar{z} \vee u$.

2. Дана база знань:

   Вовк їсть м'ясо.
   Заєць їсть моркву.
   Якщо хтось їсть м'ясо, то він є хижаком.
   Запишіть ці знання у вигляді фраз Хорна та перевірте методом резолюцій, чи є хижаками вовк і заєць.
3. Маємо базу знань: *будь-яка людина може успадкувати майно від іншої людини тільки, якщо вона є її родичем або якщо стосовно неї було складено заповіт. Іванов успадкував майно від Петрова, але Петров не складав заповіту.*
   Запишіть ці знання у вигляді фраз Хорна та доведіть методом резолюцій, що *Джиннер є родичем Джонса.*

4. Дана база знань: *Сніг буває тільки тоді, якщо температура є нижчою 5 °С. У липні температура ніколи не падає нижче 5 °С.*
   Запишіть ці знання у вигляді фраз Хорна і доведіть методом резолюцій, що *в липні ніколи не буває снігу.*
5. *Усі студенти Києво-Могилянської академії знають англійську мову. Серед програмістів, які поїхали на роботу до Лондона, ніхто не знав англійської мови.*
   Запишіть ці знання у вигляді фраз Хорна і доведіть методом резолюцій, що *серед програмістів, які поїхали на роботу до Лондона, не було жодного студента Києво-Могилянської академії.*
6. *Усі жінки красиві. У всіх відділах компанії "Рожева мрія" є жінки.*
   Запишіть ці знання у вигляді фраз Хорна та доведіть методом резолюцій, що *у кожному відділі компанії "Рожева мрія" працює хоча б один красивий співробітник.*

Розділ 8
ПРОДУКЦІЙНІ МОДЕЛІ

— Що буде, якщо щось буде?
— Буде черга, а потім знову нічого не буде.
Анекдот епохи розвинутого соціалізму

8.1. Характеристика продукційних моделей

Продукційною називається ***модель*** знань, в якій база знань складається із сукупності ***продукцій***, тобто правил **"Якщо А, тоді В".**

Термін "продукція" був введений американським математиком Е. Постом. Як і логічні, продукційні моделі насамперед концентруються на дедуктивному виведенні, але є менш формалізованими і тому більш наочними, гнучкими і зручними. На сучасному етапі найбільш вживаним формалізмом для задання знань в експертних системах є саме продукційні моделі.

Наведемо кілька типових прикладів продукцій.

Приклад 8.1. Уявіть інформаційну експертну систему для зоологічного парку. Ця система повинна на основі заданого опису деякої тварини відповідати на запитання, що це за тварина. Продукції такої системи можуть мати вигляд:

Якщо тварина є хижаком, має жовто-коричневий колір та темні смуги, то це — тигр.

Якщо тварина є птахом, вона не літає, вона плаває, має чорно-білий колір, то це — пінгвін.

Наведені продукції описують **імплікації** (з істинності A випливає істинність B).

Приклад 8.2. Нехай ідеться про систему керування деяким технологічним процесом. Продукції можуть мати такий вигляд: *Якщо температура перевищує 200 градусів, ввімкнути систему охолодження.* Такі продукції задають **умовні операції** (якщо справедлива умова A, виконати дію B).

Приклад 8.3. Типовими продукціями є **правила підстановки** (замінити A на B). Прикладом можуть послугувати правила підстановки **формальних граматик.**

База знань, організована як сукупність продукцій, разом з механізмом керування продукціями, утворює ***продукційну систему.***

Дамо формалізованіше визначення продукції [129]:

Продукційною моделлю називається подання знань у вигляді сукупності ***продукцій.*** *Продукція визначається як вираз такого вигляду:*

$$(i)\ Q;\ P;\ A \Rightarrow B;\ N,$$

де (i) *— ім'я продукції, за допомогою якого вона виокремлюється серед усієї множини продукцій;* Q *— сфера застосування продукції; предметна область, до якої вона належить;* P *— умова застосування продукції;* $A \Rightarrow B$ *— ядро продукції; інтерпретація залежить від конкретної ситуації;* N *— постумова продукції, постулює процедури, які необхідно виконати після реалізації ядра.*

Як приклад можна навести таку продукцію:

(*БС*-29) *Банківська сфера; клієнт має рахунок у банку на достатню суму; якщо клієнт бажає зняти певну суму грошей, то він повинен написати ордер; внести зміни до бази даних.*

Дуже перспективним видається поєднання продукційних моделей з фреймовими і мережними. Наприклад, самі знання можна описувати на основі семантичних мереж, а операції над ними задавати як продукції, що зумовлюють заміну одного фрагмента мережі на інший. З іншого боку, самі продукції можна включати до семантичних мереж і фреймів. Методи класу можна розглядати як продукції — особливо у випадку, якщо вони є демонами, що запускаються при виконанні певних умов. Як типові продукції можна розглядати зв'язки функціональної мережі, які задають алгоритми обчислення одних величин через інші.

Популярність продукційних моделей зумовлена такими їх позитивними рисами [129]:

1. Більшість людських знань може бути записана у вигляді продукцій.
2. Модульність; продукції, за рідким винятком, є незалежними, і внесення або вилучення окремих продукцій, як правило, не приводить до змін в інших продукціях.
3. У разі необхідності продукційні системи можуть реалізувати будь-які алгоритми.
4. Продукції можуть бути порівняно легко розподілені за сферами застосування.

5. Дуже перспективним є об'єднання продукційних систем і мережних завдань.
6. Продукції можуть працювати паралельно та асинхронно, тому їх зручно реалізовувати на основі багатопроцесорних комплексів.

До основних проблем, пов'язаних з використанням продукційних систем, можна віднести складність перевірки несуперечливості та коректності функціонування.

8.2. Продукції та мережі виведення

Системи продукцій часто буває зручно зображати у вигляді графів, які дістали назву ***мереж виведення***.

Мережі виведення будуються так. Вершини графу відповідають умовам і діям, що входять до лівих і правих частин продукцій. Якщо в продукційній системі є продукція $A \to B$, то від вершини A до вершини B йде орієнтована дуга. Якщо ж є продукція A, $B \to C$, то дуги (AC) та (BC) зв'язані відношенням I (кон'юнкції).

Так, для продукційної системи

$A \to C$
$B \to C$
$B, F \to L$
$F \to Q$
$D, C \to G$

мережа виведення має такий вигляд:

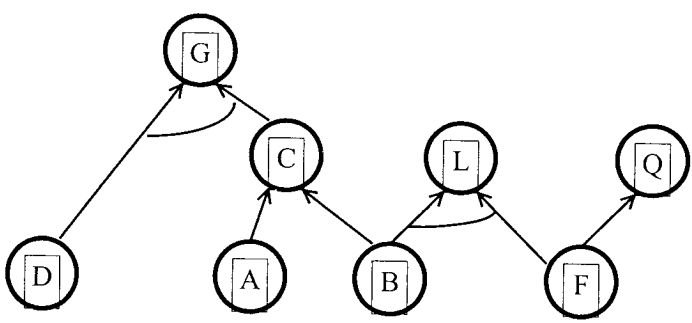


Рис. 8.1. Приклад мережі виведення

Детальніші правила побудови мереж виведення можна знайти в [175].

На основі аналізу мереж виведення стає наочнішим і очевиднішим те, що відповідь на запит користувача можна отримати шляхом зведення задачі до підзадач. Так, щоб вирішити задачу G, необхідно вирішити і задачу D, і задачу C; вирішення задачі C, в свою чергу, потребує вирішення або A, або B. Таким чином, мережа виведення для кожної задачі є структурою, близькою до традиційних ***І-АБО-графів***.

8.3. Типова схема роботи експертної системи на базі продукцій

На рис. 8.2 показана узагальнена схема типової експертної системи, побудованої на основі продукційних правил:

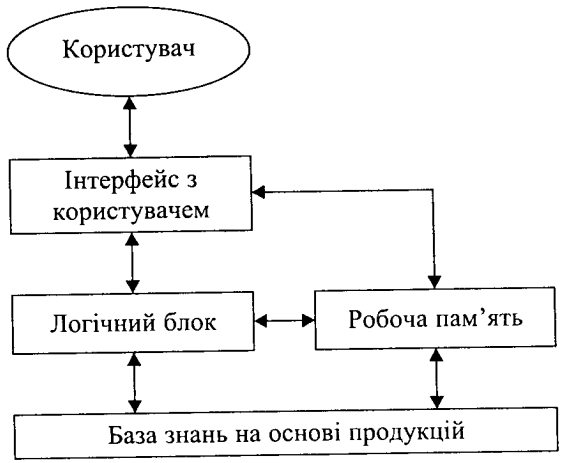


Рис. 8.2. Типова схема роботи продукційної системи

До робочої пам'яті заносяться факти, що задаються користувачем, а також його запити. Логічний блок зіставляє цю інформацію з правилами, що зберігаються в базі знань, і застосовує ті правила, які можуть бути зіставлені із вмістом робочої пам'яті. Наприклад, якщо в базі знань є продукції $A \Rightarrow B$ та $B \Rightarrow C$, то вони можуть бути виконані, коли користувач ввів дані про істинність факту A.

8.4. Пряме та зворотне виведення

Існують дві основні стратегії логічного виведення в продукційних системах:

- ***пряме;***
- ***зворотне.***

Пряме виведення на основі наявних правил передбачає аналіз: 1) усіх наслідків з фактів, 2) наслідків з наслідків і т. д. Процес продовжується, поки не буде встановлено істинність або хибність запиту користувача.

Наприклад, нехай є продукції $A \Rightarrow B$ та $B \Rightarrow C$; користувач оголосив про істинність факту A і спитав про істинність C. Ланцюжок прямого виведення матиме вигляд: з A виводиться B, з B виводиться C, отже, C — істинне.

Пряме виведення не вважається ефективним через те, що може перебиратися чимало безперспективних продукцій, і, відповідно, породжується дуже багато зайвих проміжних результатів. Експертні системи на основі

прямого виведення використовуються, як правило, при плануванні і прогнозуванні.

При ***зворотному виведенні*** логічний блок починає роботу із запиту користувача, тобто з твердження, яке перевіряється. Розглядається одне з правил, на основі яких можна вивести це твердження, після чого перевіряється істинність лівої частини цього правила. Процес повторюється, доки не дійде до фактів, які вважаються істинними.

У нашому прикладі ланцюжок зворотного виведення матиме вигляд: "*C* виводиться з *B*, *B* виводиться з *A*, *A* істинне, отже, і *C* істинне".

Як і у випадку прямого виведення, зворотне виведення може вимагати повторень у разі невдалої спроби, і тому є стратегією перебору.

Зворотне виведення тісно пов'язано з резолюцією, зокрема лінійною, і з прологівським механізмом виконання програм.

8.5. Типові дисципліни виконання продукцій

Передусім ядра продукцій прийнято поділяти на ***детерміновані*** і ***недетерміновані***. У детермінованих продукціях при виконанні лівої частини завжди виконується і права. У недетермінованих продукціях права частина при виконанні лівої виконується необов'язково. Можливі, наприклад, такі інтерпретації недетермінованих продукцій: *"Якщо A, то можливе B"*; або *"Якщо A, то B з певною вірогідністю"*.

Існують різні режими керування виконанням готових продукцій, тобто продукцій, ліві частини яких справедливі, і, відповідно, ці продукції можуть бути виконані негайно.

Можна виокремити такі режими:

- ***режим негайного виконання***;
- ***режим формування конфліктного набору***.

У режимі негайного виконання, якщо знайдена перша-ліпша продукція, саме вона виконується невідкладно.

У режимі формування конфліктного набору знайдені готові продукції не виконуються відразу, а включаються до ***конфліктного набору*** — списку продукцій, готових до виконання. Лише після завершення формування конфліктного набору відбувається вирішення конфлікту, тобто вибір зі списку готових продукцій якоїсь однієї. Інша назва конфліктного набору ***"фронт готових продукцій"***.

Розглянемо деякі проблеми, пов'язані з конфліктними наборами.

Нехай маємо продукції:

1) $A \Rightarrow B$

2) $A \Rightarrow C$,

і *A* істинне.

Тоді обидві продукції готові до виконання і утворюють конфліктний набір.

Якщо ядра продукцій інтерпретуються як імплікації, проблема є менш гострою. Продукційна система є чисто декларативною. Після застосування 1),

тобто виведення B, A залишається істинним, і ми завжди можемо використати продукцію 2). Таким чином, порядок застосування продукційних правил може впливати лише на час роботи системи, але не на істинність висновків.

Зовсім інша ситуація виникає, якщо ядра продукцій інтерпретуються як "виконати таку-то дію". Тоді після виконання 1) ситуація може змінитися: A може перестати бути істинним, і тоді продукція 2) може стати недосяжною. Тому, якщо насправді треба було вибирати C, вибір B стає помилкою, яку не завжди можна виправити.

Існують різні стратегії вирішення конфліктів, а також обмеження при включенні готових продукцій до конфліктного набору.

Розрізняють також ***централізоване*** і ***децентралізоване керування*** продукціями.

За централізованого керування продукційна система має центр керування, який вирішує, коли має спрацювати та чи інша продукція.

За децентралізованого керування кожна продукція може самостійно прийняти рішення про свій запуск. Один з типових механізмів, який застосовується при цьому, є механізм ***"класної дошки"***. Таку назву має спеціальна область пам'яті, доступна для різних продукцій. На ній продукції, що працюють паралельно, знаходять інформацію, що може ініціювати їх запуск. Туди ж вони пишуть інформацію, що може виявитись корисною для інших продукцій.

Зрозуміло, що "класних дошок" може бути кілька; можуть бути введені певні ієрархії "класних дошок" і т. п. Огляд різних архітектур "класних дошок" можна знайти в [305].

8.6. Основні стратегії вирішення конфліктів у продукційних системах

Очевидно, що найзагальнішим способом керування системою продукцій є комплексне планування її роботи, тобто прийняття рішень на основі аналізу всіх можливих дій та їх наслідків з погляду мети, що стоїть перед системою. Але зрозуміло, що вказаний принцип рідко може бути застосований у повному обсязі, насамперед через експоненційне зростання можливих варіантів розвитку подій.

Існують стратегії керування вирішенням конфліктів, мета яких — намагатися уникнути експоненційного вибуху. Подібні стратегії здебільшого можуть бути охарактеризовані як евристичні. В [129] описані деякі стратегії керування виконанням продукцій.

Принцип "стосу книг". Ґрунтується на ідеї, що найкориснішою є та продукція, яка використовується найчастіше (так само, як у стосі книг ті, які найчастіше користуються попитом, як правило, опиняються зверху). Відповідно, з фронту готових продукцій вибирається продукція з максимальною частотою використання. У [129] відмічено, що керування за принципом "стосу книг" доцільно застосовувати, якщо продукції відносно неза-

лежні одна від одної, наприклад, коли кожна з них являє собою правило **"ситуація А ⇒ дія В"**.

Принцип найдовшої умови. З фронту готових продукцій вибирається та, для якої стала справедливою найдовша умова виконання. Ідеологічною основою цього принципу є таке міркування, що часткові правила, застосовані до вузького класу ситуацій, є важливішими, ніж загальні правила для широкого класу ситуацій. Наприклад, нехай є дві продукції:

(1) *Якщо А — птах, то А літає.*

(2) *Якщо А — птах і А — пінгвін, то А не літає.*

Якщо відомо, що *А — пінгвін*, то за принципом найдовшої умови слід вибирати (2).

У [129] відмічається, що принцип найдовшої умови доцільно застосувати, якщо продукції прив'язані до типових ситуацій, зв'язаних відношенням **"часткове — загальне"**. До цього можна додати, що цей принцип заслуговує на особливу увагу при аналізі винятків. Так, у нашому прикладі правило (2) є винятком з правила (1); якби ми були впевнені, що (1) не має винятків, ми могли б застосувати це загальне правило у будь-якій ситуації. Легко бачити очевидні аналогії принципу найдовшої умови з принципом блокування наслідування, описаним у п. 5.6.

Принцип метапродукцій. Ґрунтується на введенні до системи знань ***метапродукцій***, тобто правил використання продукцій. Типовими метапродукціями можуть бути правила, які визначають, що слід робити, якщо до фронту готових продукцій ввійшли або не ввійшли ті чи інші конкретні продукції.

Принцип "класної дошки". Конфлікти вирішуються на основі обміну інформацією з використанням "класної дошки". Застосування "класної дошки" часто комбінується з застосуванням метапродукцій.

Принцип вибору за пріоритетом. Кожній продукції надається пріоритет, що відображає її важливість. Ці пріоритети можуть бути різними для кожного типу ситуацій, а також статичними і динамічними. Динамічні пріоритети змінюються з часом і можуть відображати, наприклад, ефективність використання продукцій у минулому або час перебування продукції у фронті готових продукцій. Аналіз часу знаходження продукції у фронті готових продукцій цікавий тим, що на його основі можна дійти діаметрально протилежних висновків. З одного боку, система керування продукціями може з часом підвищувати динамічний пріоритет, щоб будь-яка продукція колись була обов'язково виконана. З іншого боку, система керування може дійти висновку, що, якщо продукція досі не виконувалась, то вона взагалі не потрібна, і, відповідно, не підвищити її пріоритет, а, навпаки, знизити.

Керування за іменами. Базується на заданні для імен продукцій формальної граматики або іншої процедури, що забезпечує звуження фронту готових продукцій і вибір з нього чергової продукції для виконання.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте неформальні визначення продукції, продукційної моделі та продукційної системи.
2. Наведіть можливі інтерпретації продукцій.
3. Дайте формальне визначення продукції.
4. Опишіть поняття "мережа виведення". Наведіть власний приклад.
5. Опишіть типову схему роботи продукційної системи.
6. Які ви знаєте стратегії логічного виведення в продукційних системах?
7. Що таке пряме виведення? Наведіть приклад.
8. Що таке зворотне виведення? Наведіть приклад.
9. Яка стратегія логічного виведення (пряма чи зворотна) вважається ефективнішою? Поясніть, чому.
10. У чому полягає різниця між детермінованими і недетермінованими продукціями?
11. Що означає централізоване і децентралізоване керування продукціями?
12. Опишіть поняття "класної дошки". Поясніть, як вона використовується при децентралізованому керуванні продукціями.
13. Дайте визначення конфліктного набору.
14. Чи може порядок виконання продукцій у продукційних системах впливати на результат їх функціонування? Поясніть, чому.
15. Опишіть основні стратегії вирішення конфліктів у продукційних системах.
16. Охарактеризуйте принцип найдовшої умови. Яким чином він пов'язаний з обробкою винятків?

ТЕМА ДЛЯ ОБГОВОРЕННЯ

Охарактеризуйте місце продукційних моделей серед інших моделей задання знань.

ЧАСТИНА
III

ФОРМАЛІЗАЦІЯ НЕДОСТОВІРНИХ І НЕЧІТКИХ ЗНАНЬ

Розділ 9
МОДАЛЬНІ ЛОГІКИ

Я знаю, що я нічого не знаю.
Сократ

9.1. Алетичні та епістемічні логіки

Часто буває необхідним включати до бази знань твердження, подібні до таких: *"На вулиці може йти сніг"* або *"Вася знає, що квадрат гіпотенузи дорівнює сумі квадратів катетів"*, або *"Маша гадає, що Дніпро впадає в Азовське море"*, або *"Петро вважає, що Степан не заперечуватиме, що він убив Івана"* тощо.

Подібні твердження називаються ***модальними***, оскільки вони оперують із твердженнями, які дають оцінку іншим твердженням. Логічні системи, що оперують з подібними твердженнями, носять назву ***модальних логік***.

Розглянемо два основних типи модальних логік: логіки, які оперують твердженнями про можливість або необхідність того чи іншого факту ***(логіки можливого, алетичні логіки)*** та логіки, які визначають ставлення суб'єкта до тих чи інших фактів, в першу чергу його знання та віру (логіки знання та віри, ***епістемічні логіки***).

Подібно до традиційних кванторів існування та узагальнення, модальні логіки залучають до розгляду інші квантори. Так, логіки можливого оперують з модальними кванторами необхідності та можливості [167]:

$\Box F$ — твердження F є необхідним;

$\Diamond F$ — твердження F є можливим.

Основним співвідношенням, яке зв'язує квантори можливості та необхідності, є:

$\Box F \equiv \neg \Diamond \neg F$

(якщо висловлювання F є необхідним, його заперечення є неможливим).

9.2. Тризначна логіка Лукасевича

Найпростішою і однією з найвідоміших модальних логік є ***тризначна логіка Лукасевича***. У цій логіці будь-яке твердження може набувати одне з трьох можливих значень істинності:

0 — неможливе твердження;

1 — можливе, але не необхідне твердження;

2 — необхідне твердження.

У логіці Лукасевича має місце **закон виключеного четвертого: будь-яке твердження є або неможливим, або можливим, але не необхідним або необхідним**.

Таблиці істинності для заперечення, кон'юнкції та диз'юнкції в логіці Лукасевича визначаються таким чином:

$\sim F$

F	~F
0	2
1	1
2	0

$F \wedge G$

F\G	0	1	2
0	0	0	0
1	0	1	1
2	0	1	2

$F \vee G$

F\G	0	1	2
0	0	1	2
1	1	1	2
2	2	2	2

9.3. Логіка знання

Найбільш дослідженою та формалізованою епістемічною логікою є логіка знання, яка оперує з твердженнями про знання того чи іншого суб'єкта (людини або штучно створеної інтелектуальної системи).

Позначимо модальне твердження *"суб'єкт X знає твердження A"* формулою $K_X(A)$. У [320] наведено такі класичні аксіоми логіки знання:

Аксіома modus ponens: $(K_X(A) \wedge K_X(A \Rightarrow B)) \Rightarrow K_X(B)$.

Ця аксіома стверджує, що, якщо суб'єкт знає твердження A і знає, що з нього логічно випливає твердження B, то він знає і твердження B. Інколи ця аксіома формулюється в еквівалентній дистрибутивній формі:

$K_X(A \Rightarrow B) \Rightarrow [(K_X(A) \Rightarrow K_X(B)]$

(цей запис називається ***аксіомою дистрибутивності***).

Аксіома знання: $K_X(false) \Rightarrow false$.

Ця аксіома по суті являє собою лінгвістичну домовленість про те, що знати хибні твердження неможливо. З іншого боку, це означає, що якщо деякий інший суб'єкт Y заявляє, що *X знає деяке твердження*, то Y обов'язково повинен вважати це твердження істинним.

Аксіома позитивної інтроспекції: $K_X(A) \Rightarrow K_X(K_X(A))$.

Ця аксіома стверджує, що, якщо дехто знає певне твердження, то він знає і про те, що він знає це твердження.

Аксіома негативної інтроспекції. $\neg K_X(A) \Rightarrow K_X(\neg K_X(A))$.

Ця аксіома стверджує, що, якщо хтось не знає деякого твердження, то він знає про те, що він його не знає.

Аксіома епістемічної необхідності. З логічної звідності A випливає $K_X(A)$.

Аксіома логічної всемогутності. З тверджень "з A виводиться B" і з $K_X(A)$ випливає $K_X(B)$.

Остання аксіома стверджує, що якщо суб'єкт знає про деякий факт, то він знає і про всі логічні наслідки з цього факту.

Цікавим є таке запитання: чи обов'язково всі перелічені аксіоми повинні виконуватися в будь-якій реальній ситуації? Очевидно, аксіоми епістемічної необхідності та логічної всемогутності є надто сильними. Вони, як правило, не виконуються при моделюванні знань реального учня (таке моделювання є дуже важливим при створенні навчальних комп'ютерних систем). Учень далеко не завжди виводить усі логічні наслідки з доступних йому знань (так само, як будь-яка людина дуже рідко здійснює дедуктивні побудови до кінця).

Аксіома негативної інтроспекції також може не мати місця. Якщо реальний учень не знає про деякий факт, з цього зовсім не випливає, що він знає про те, що він його не знає. Більше того, при проектуванні систем штучного інтелекту відмова від негативної інтроспекції може стати одним з потужних засобів цілевизначення: якщо система отримує інформацію про те, що вона чогось не знає, вона може поставити собі за мету отримати відповідні знання і почати докладати для цього зусиль.

Крім того, необхідно враховувати і той факт, що знання можуть бути неявними, зокрема вони далеко не завжди формулюються у вигляді явних тверджень. Проводяться дослідження, спрямовані на моделювання неявних, в тому числі неповних, знань, а також на розробку процедур для ефективного маніпулювання такими знаннями. Слід згадати, наприклад, епістемічну логіку Левеск'є, описану в [10].

9.4. Семантика можливих світів

Ідеологія семантики можливих світів, основи якої описані в [167], базується на такому філософському міркуванні. Можна уявити собі, що, крім того світу, в якому ми живемо, існують паралельні світи, в яких події могли б розвиватися по-іншому. Або: ті чи інші історичні події, які відбувалися в минулому, могли б не мати місця, якби історичний розвиток пішов іншим шляхом.

Семантика можливих світів очевидним чином пов'язана з логікою можливого. Так, на семантиці можливих світів базується ***чотиризначна логіка можливого***, для якої характерні чотири можливі значення істинності:

3 — ***необхідно істинні*** твердження, істинність яких визначається фізичними та іншими законами, і які є істинні в усіх можливих світах, в яких діють ці закони;

2 — ***нейтрально істинні*** твердження, які є істинними, але могли б виявитися хибними, або є хибними в деякому іншому світі;

1 — ***нейтрально хибні*** твердження, які є хибними, але могли б виявитися істинними;

0 — ***необхідно хибні*** (або ***неможливі***) твердження, які є хибними в усіх можливих світах.

Менш очевидним є те, що на основі семантики можливих світів можна розвивати і епістемічні логіки. Але, якщо ми кажемо про **знання** деякого суб'єкта A, ми можемо розглядати **можливий світ**, пов'язаний з суб'єктом A, і в цьому можливому світі істинними виявляються ті твердження, які A вважає істинними.

9.5. Основи теорії можливостей

Теорія можливостей [108], яка інтенсивно розвивається в останні десятиліття, має на меті розглянути з єдиних позицій модальні, недостовірні та нечіткі знання.

У рамках теорії можливостей з кожним твердженням L пов'язуються дві міри: ***міра можливості*** $\Pi(L)$ та ***міра необхідності*** $H(L)$. Ці міри, з одного боку, носять модальний характер, а з іншого — є узагальненням мір упевненості, які залучаються до розгляду при неточному логічному виведенні.

Основною аксіомою теорії можливостей є аксіома монотонності, яка формулюється таким чином: якщо $L \Rightarrow M$, то $\Pi(M) \geq \Pi(L)$; $H(M) \geq H(L)$. Інакше кажучи, міра можливості або необхідності висновку не може бути меншою, ніж відповідна міра передумови.

У [108] наводяться такі основні властивості мір можливості та необхідності:

$\Pi(L \vee M) = \max(\Pi(L), \Pi(M))$.

$H(L \wedge M) = \min(H(L), H(M))$.

$\Pi(L) = 1 - H(\neg L)$.

$\Pi(L) \geq H(L)$ (якщо твердження є необхідно істинним, воно повинно бути можливо істинним).

Якщо $H(L) > 0$, то $\Pi(L) = 1$.

Якщо $\Pi(L) < 1$, то $H(L) = 0$.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке модальне твердження?
2. Охарактеризуйте поняття модальної логіки.

3. Що таке алетичні та модальні логіки?
4. Охарактеризуйте тризначну логіку Лукасевича.
5. Наведіть для логіки Лукасевича таблиці істинності кон'юнкції, диз'юнкції та заперечення.
6. Наведіть основні аксіоми логіки знання. Чи завжди вони повинні виконуватися?
7. Охарактеризуйте семантику можливих світів.
8. Наведіть власні приклади необхідно істинних, нейтрально істинних, необхідно хибних і нейтрально хибних тверджень.
9. Охарактеризуйте в загальних рисах теорію можливостей.
10. Наведіть аксіому монотонності для мір можливості та необхідності.

ТЕМА ДЛЯ ОБГОВОРЕННЯ

Розгляньте типові ситуації, в яких справджуються або не справджуються ті чи інші аксіоми логіки знання.

Розділ 10

ЛОГІЧНЕ ВИВЕДЕННЯ ЗА НЕДОСТОВІРНИХ ЗНАНЬ

— Лікарю, а що ви мені написали в діагнозі: ЧЙЗ?
— Чорт Його Знає!

Анекдот

10.1. Поняття про неточне логічне виведення

Проблема недостовірних знань була в загальних рисах охарактеризована в розд. 4. Часто буває так, що експерт не зовсім упевнений в тому чи іншому факті, але, незважаючи на це, інформація залишається цінною і повинна бути включена до бази знань.

Згідно з [129], висловлення називається ***неточним***, якщо його істинність або хибність не можуть бути встановлені однозначно, тобто твердження не є ні абсолютно достовірним, ні абсолютно хибним.

Неточним виведенням *називається логічне виведення в умовах неточності (недостовірності) знань.*

Неточне виведення традиційно розглядається як самостійний напрям, хоча неточне твердження можна було б інтерпретувати як частковий випадок модального твердження.

Неточне виведення слід відрізняти від роботи з нечіткими знаннями, хоча останні також формально підпадають під сформульоване вище визначення.

10.2. Деякі визначення з теорії ймовірностей

Багато методик недостовірного логічного виведення тісно пов'язано з апаратом теорії ймовірностей. Тому для розуміння цього розділу потрібно навести основні необхідні факти з теорії ймовірностей. Для ґрунтовнішого ознайомлення з основами теорії ймовірностей читачеві рекомендується звернутися до відомих підручників, наприклад [43, 85, 143, 277].

Ймовірність події A (позначається $P(A)$) слід розуміти як міру достовірності деякої події. Завжди виконується властивість $0 \le P(A) \le 1$. При цьому ***неможлива подія*** (подія, яка ніколи не може відбутися) має ймовірність 0, а ***достовірна подія*** — ймовірність 1. Дамо формалізованіші визначення.

Розглянемо ***множину елементарних наслідків*** Ω. Елементарні наслідки розглядаються як можливі наслідки деякого експерименту, при цьому ці наслідки вважаються **рівноможливими та взаємовиключними**. Тоді будь-яка подія A розглядається як підмножина множини елементарних наслідків. Подія A відбувається, якщо має місце будь-який елементарний наслідок, що входить до множини A. Якщо елементарний наслідок входить до множини A, кажуть, що він є ***сприятливим*** для події A.

Ймовірність події A *визначається як відношення кількості сприятливих наслідків до загальної кількості елементарних наслідків.*

На основі даного визначення в ряді випадків можна безпосередньо розрахувати ймовірності. Наведемо кілька прикладів.

Приклад 10.1. Підкидається гральний кубик, на кожній грані якого міститься певна кількість очок від 1 до 6. Яка ймовірність того, що випаде парна кількість очок?

Оскільки в результаті експерименту може випасти будь-яка кількість очок від 1 до 6, множина елементарних наслідків $\Omega = \{1, 2, 3, 4, 5, 6\}$.

Сприятливими для події A є наслідки 2, 4, 6, отже, $A = \{2, 4, 6\}$.

Маємо $P(A) = 3/6 = 0.5$.

Приклад 10.2. Підкидаються два гральних кубики. Яка ймовірність того, що сума очок, які випадуть на обох гранях, дорівнюватиме 5?

Експеримент полягає в підкиданні двох кубиків, і елементарний наслідок є парою, кожний елемент якої задає кількість очок, що випала на відповідному кубику. Таким чином,

$\Omega = \{11, 12, 13, 14, 15, 16, 21, \ldots, 61, 62, 63, 65, 66\}$;

кількість елементарних наслідків дорівнює 36.

$A = \{14, 41, 23, 32\}$.

Тоді $P(A) = 4/36 = 1/9 \approx 0.11$.

Подією, ***протилежною*** до події A (позначається $\overline{A}$), називається та, яка полягає в тому, що подія A не відбувається. На основі класичного визначення ймовірностей легко переконатися в тому, що

$$P(\overline{A}) = 1 - P(A)$$

(читачеві рекомендується довести це самостійно).

Ймовірність події, по суті, є частотою появи цієї події. Це підкреслюється в такому визначенні.

Частотне визначення ймовірності. Нехай проводиться серія однотипних експериментів і нехай $K_A(n)$ — кількість випадків, в яких відбулася подія A, якщо було проведено n експериментів. Тоді

$$P(A) = \lim_{n\to\infty} \frac{K_A(n)}{n}.$$

Наприклад, якщо проводиться 1000 експериментів, а ймовірність події A дорівнює 0.3, то подія A настане приблизно в 300 випадках. При цьому зі зростанням кількості експериментів частина випадків, в яких відбувається дана подія, все більше наближається до ймовірності.

Введемо такі позначення:

$P(A \cup B)$ — ймовірність того, що настане або подія A, або подія B, або і A, і B;

$P(A \cap B)$ — ймовірність того, що настануть і A, і B.

Тоді

$$P(A \cup B) = P(A) + P(B) - P(A \cap B).$$

Зокрема, якщо A і B — несумісні (взаємовиключні) події, маємо

$$P(A \cup B) = P(A) + P(B).$$

Дуже важливим є поняття ***умовної ймовірності***. Умовною ймовірністю події A за умови B (позначається $P(A\backslash B)$ називається ймовірність появи A за умови, що подія B уже відбулася.

Справедливе співвідношення

$$P(A \cap B) = P(B) \cdot P(A\backslash B).$$

Події A і B називаються ***незалежними***, якщо поява A жодним чином не залежить від появи B. Інакше кажучи, A і B незалежні, якщо $P(A\backslash B) = P(A)$.

Кажуть, що події $H_1, ..., H_n$ утворюють ***повну групу подій***, якщо вони є взаємовиключними і сума їх ймовірностей дорівнює 1 (це означає, що з цих подій настане одна і тільки одна).

Якщо $H_1, ..., H_n$ — повна група подій, то для будь-якої події A справедливі співвідношення

$$P(A) = \sum_{i=1}^{n} P(H_i) \cdot P(A\backslash H_i)$$

(формула повної ймовірності) та

$$P(H_i\backslash A) = \frac{P(H_i) \cdot P(A\backslash H_i)}{\sum_{i=1}^{n} P(H_i) \cdot P(A\backslash H_i)}.$$

Остання формула називається ***формулою Байєса*** і має дуже велике значення, оскільки дозволяє обчислювати ***апостеріорні*** ймовірності $P(H_i\backslash A)$ "гіпотез" H_i за умови, що подія A відбулася через апріорні ймовірності $P(A\backslash H_i)$. Більшість методик неточного логічного виведення так чи інакше пов'язані з формулою повної ймовірності та формулою Байєса.

При неточному логічному виведенні з кожним твердженням (фактом або правилом виведення) пов'язується число, яке характеризує міру його надійності. Ця характеристика називається ***коефіцієнтом упевненості***, або ***мірою достовірності***. Розрахунок коефіцієнтів упевненості тісно пов'язаний з ймовірнісними методами, хоча ненадійність інформації далеко не завжди носить імовірнісний характер (див. п. 10.3).

Коефіцієнти упевненості часто визначаються на основі експертних оцінок. Проте в ряді випадків коефіцієнт упевненості можна отримати шляхом статистичних досліджень. Якщо ж здійснюється логічне виведення, то коефіцієнт упевненості нерідко можна підрахувати, якщо відомі міри достовірності умови та правила виведення.

Чим надійнішою є інформація, тим вищі відповідні коефіцієнти упевненості. Часто використовується шкала, за якої коефіцієнт **1** відповідає достовірно істинній події, **0** — достовірно хибній, а **0.5** — повній невизначеності.

Але згадана шкала не є єдино можливою. Так, експертні системи MYCIN та EMYCIN, що є розвитком MYCIN (ці системи призначені для виявлення мікроорганізмів у крові) використовують шкалу, в якій достовірній істинності відповідає значення **1**, достовірній хибності — значення **–1**, а повній невизначеності — **0**.

10.3. "Об'єктивна" та "суб'єктивна" невизначеність

Перелічимо деякі з можливих типів невизначеності.

"Об'єктивна" невизначеність. Цей тип пов'язаний з принципово випадковим характером процесів, які прогнозуються. Наприклад, передбачається результат перебігу певної хвороби, а статистичний аналіз свідчить, що досі ефективність лікування становила 70 %. У такому разі робиться відповідний висновок: пацієнт буде вилікований з імовірністю 70 %. Важливо розуміти, що **"об'єктивна" невизначеність, як правило, пов'язана з впливом певних невідомих або неконтрольованих факторів і за своєю суттю являє собою ймовірність відповідної події.**

Можна виокремити дві основні властивості "об'єктивної" невизначеності:

- подія, для якої визначається коефіцієнт упевненості, ще не відбулася, але обов'язково настане момент, коли вона або відбудеться, або не відбудеться; після цього міра достовірності стане дорівнювати відповідно 1 або 0;
- якби було можливим провести серію експериментів за однакових умов, то частина експериментів, в яких подія відбулася б, приблизно дорівнювала б α; тут α — міра достовірності (фактично, ймовірність події).

"Суб'єктивна" невизначеність. Ми говоримо про цей тип невизначеності у разі, якщо подія, про яку йдеться, або вже відбулася, або не відбулася, і істинна міра достовірності дорівнює відповідно або 1, або 0. Але,

якщо ця істинна міра достовірності експертові невідома (він не знає, чи відбулася подія, чи ні) або відома нечітко, то він дає власну оцінку міри надійності, і ця власна оцінка неминуче є суб'єктивною. Зрозуміло, що доцільність застосування ймовірнісних методів у такому разі є досить сумнівною.

Варто розрізняти два типи "суб'єктивної" невизначеності.

"Суб'єктивна" невизначеність при авторитетності джерела знань. Авторитетність джерела знань означає, що експерт є людиною серйозною і відповідальною та його власній оцінці можна довіряти. Тому власна оцінка істинності твердження приймається як робоча і використовується при логічному виведенні. Авторитетними, як правило, є серйозні наукові праці, енциклопедії тощо.

"Суб'єктивна" невизначеність при неавторитетності джерела знань. У даному разі, незважаючи на ступінь упевненості "експерта", його оцінці довіряти не можна. Тут можна розглядати різноманітні випадки, наприклад:

- твердження є об'єктивно істинним, але "експерт" у ньому невпевнений і ступінь його кваліфікації не дозволяє йому адекватно сформулювати міру невпевненості;
- твердження є об'єктивно хибним, але "експерт" вважає його істинним і формулює як достовірне;
- твердження є об'єктивно хибним, "експерт" усвідомлює його хибність, але повідомляє про істинність твердження з метою ввести в оману.

Часто доводиться мати справу з ***невизначеністю комбінованого типу***, тобто з невизначеністю, що поєднує риси як "об'єктивної", так і "суб'єктивної" невизначеності. Один з найпростіших випадків подібного поєднання полягає в такому. Розглядається твердження *"Завтра відбудеться подія A"*. Існує істинна, "об'єктивна" ймовірність цієї події. Але експертові ця істинна оцінка ймовірності може бути або невідомою, або відомою неточно. Тоді він дає власну оцінку вказаної ймовірності, і ця власна оцінка має суб'єктивний характер.

Слід також звернути увагу на таке явище, як ***конфлікт між різними джерелами знань***. Різні джерела знань можуть суперечити одне одному; наприклад, те, що людина прочитала в книзі, суперечить її знанням, або різні наукові школи приходять до протилежних висновків стосовно одного й того самого явища.

Принцип індиферентності є одним з найпростіших правил, яке допомагає приймати рішення в умовах невизначеності. Він формулюється так: якщо є n гіпотез і немає жодних свідчень на користь будь-якої з цих гіпотез, то міра достовірності кожної гіпотези приймається за $1/n$.

Бездумне застосування принципу індиферентності може привести як до нерозумних висновків, так і до логічних помилок. Розглянемо простий приклад. Нехай є дві гіпотези: *"Під даним деревом є скарб"* і *"Під даним деревом немає скарбу"*. Згідно з принципом індиферентності міру достовірності кожної з цих гіпотез слід прийняти за 50 %. Але 50 шансів зі ста — це дуже серйозна підстава для того, щоб почати шукати скарб під деревом. Зрозуміло, що такий висновок суперечить здоровому глузду.

10.4. Загальні принципи неточного виведення

У [129] виокремлені два типи механізмів роботи з неточними твердженнями.

Неточне виведення "приєднаного" типу характеризується тим, що з кожним твердженням x пов'язується міра його достовірності $\gamma(x)$. Логічне виведення здійснюється за принципами, характерними для точних знань, але при цьому висновкам також приписується певна міра достовірності. При цьому необхідно задати:

- *функцію* $\gamma(x) = f(\gamma(x_1), \ldots \gamma(x_n))$, яка задає міру неточності складного твердження x, якщо задані міри неточності його складових частин $x_1, \ldots, x_n$. Наприклад, є два висловлення: x_1 = *"завтра буде дощ"* з мірою достовірності $\gamma(x_1) = 0.3$ і x_2 = *"завтра буде сніг"* з мірою достовірності $\gamma(x_2) = 0.6$. З цих двох тверджень можна утворити ряд складених тверджень, наприклад: $x = x_1 \vee x_2$ (*"завтра буде або дощ, або сніг"*). Функція f розрахунків неточності для складених тверджень може задаватись по-різному, наприклад, типовим є використання функцій $\gamma(x) = \max(\gamma(x_1), \ldots, \gamma(x_n))$ для диз'юнкції та $\gamma(x) = \min(\gamma(x_1), \ldots, \gamma(x_n))$ — для кон'юнкції. Тоді у нашому випадку $\gamma(x) = \max(\gamma(x_1), \gamma(x_2)) = \max(0.3, 0.6) = 0.6$;
- *функцію* $\gamma(y) = g(\gamma(x), \gamma(r))$; ця функція задає міру неточності висновку y, якщо задані міри неточності умови x та правила виведення r. Наприклад, маємо правило r: *якщо завтра будуть опади, людина бере парасольку*. Нехай міра достовірності $\gamma(r)$ цього правила дорівнює 0.8, змістовно це можна інтерпретувати так: якщо очікуються опади, люди беруть парасольку у 80 випадках зі 100. Нехай міра достовірності умови (*"завтра будуть опади"*) $\gamma(x) = 0.6$. Для розрахунку міри достовірності висновку (*"певна людина візьме парасольку"*) можна вводити різні функції; типовим є використання добутку: $g(\gamma(x), \gamma(r)) = \gamma(x) \cdot \gamma(r)$. Тоді у нашому випадку $\gamma(y) = 0.6 \cdot 0.8 = 0.48$;
- *функцію комбінування свідоцтв* $\gamma(y) = h(\gamma_1(y), \ldots, \gamma_n(y))$, де $\gamma_i(y) = g(\gamma(x_i), \gamma(r_i))$. Це означає: якщо існує кілька свідоцтв на користь (або проти) певного твердження, і кожне з цих свідоцтв приводить до певної міри істинності висновку, то на підставі цих мір потрібно побудувати деяку узагальнену міру. Формальніше, якщо для твердження y існує n правил виведення r_i типу "Якщо x_i, то y", і кожному з цих правил виведення приписані міри достовірності $\gamma(r_i)$, і кожна умова має свою міру достовірності $\gamma(x_i)$, то ми отримуємо n мір достовірності для y: $\gamma_i(y) = g(\gamma(x_i), \gamma(r_i))$. Функція комбінування свідоцтв дозволяє отримати одну об'єднану міру достовірності. Комбінування свідоцтв є центральною проблемою неточного виведення (детальніше про цю проблему йтиметься у п. 10.6).

Для кожної конкретної методики неточного логічного виведення при недостовірних знаннях потрібно визначити конкретний вигляд функцій f, g та h.

Таких методик запропоновано досить багато; чимало з них створювалося для використання в конкретних експертних системах. Загальний огляд відомих методик неточного логічного виведення можна знайти в [129].

Як зазначалося раніше, в основі більшої частини цих методик лежить апарат теорії ймовірностей. Усі вони мають багато спільних рис. Необхідно зауважити, що жодну з цих методик не можна вважати загальною та універсальною. Багато проблем виникає навіть у разі "об'єктивної" невизначеності, для якої застосування ймовірнісних методів дає найкращі результати (деякі з цих проблем ми детально розглянемо нижче).

У разі "суб'єктивної" невизначеності ситуація виявляється ще гіршою. Тому переважна більшість методів неточного логічного виведення не має належної теоретичної бази і носить відверто евристичний характер.

Теорія неточного логічного виведення на сучасному етапі інтенсивно розвивається, і цілком вірогідною є поява нових, досконаліших методик.

Механізми неточного виведення другого типу передбачають наявність спеціальних схем виведення, орієнтованих на схему представлення неточності. Надалі ми розглядатимемо лише виведення "приєднаного" типу, характерне для продукційних систем.

10.5. Точкові та інтервальні міри неточності

Міри неточності тверджень можуть бути ***точковими*** та ***інтервальними***. У разі точкового оцінювання з кожним твердженням пов'язується єдине число, яке задає міру його достовірності. Натомість за інтервального оцінювання міра достовірності задається певним інтервалом.

Інтервальні міри достовірності в цілому є надійнішими. Наприклад, дехто каже: *"Міра достовірності того, що команда А виграє баскетбольний матч у команди В, дорівнює 0.5"*. Вважаємо, що 0 відповідає достовірній хибності, а 1 — достовірній істинності. Можливі як мінімум дві інтерпретації такої відповіді.

У першому випадку "експерт" може не розумітися на баскетболі. У такому разі його відповідь фактично означає: *"Не маю жодного уявлення, але згідно з принципом індиферентності — п'ятдесят на п'ятдесят"*. Ясно, що з такої "оцінки" немає ніякої практичної користі.

Зовсім інша ситуація виникає, якщо експерт детально проаналізував ситуацію і дійшов висновку, що команди абсолютно рівні за силою. Відповідь *"0.5"* у такому разі означає: *"Команди мають рівні шанси на перемогу. Результат залежить від випадку, тобто є об'єктивно невизначеним. Якби провести не один матч, а цілу серію, то у половині матчів виграла б команда А, а у половині — команда В"*.

Застосування інтервальних оцінок дозволило б розділити ці дві ситуації. У першому випадку інтервал невизначеності дорівнював би [0, 1], а в другому — був би значно вужчим, наприклад [0.45, 0.55].

Звичайно, це стосується лише випадків, коли люди об'єктивно оцінюють ситуацію та адекватно висловлюють міру своєї невпевненості. Ясно,

що і у першому випадку ніщо не заважає "експертові" оцінити інтервал невизначеності, наприклад, як [0.9, 1.0], але це вже зовсім інша тема.

10.6. Проблема комбінування свідоцтв

Як уже зазначалося, проблема комбінування свідоцтв є центральною проблемою неточного логічного виведення. Вона пов'язана з тим, що існує кілька свідоцтв, які говорять або на користь певного висновку, або проти нього. Як за таких умов оцінити міру достовірності висновку?

Приклад. Вважатимемо, що значення **1** відповідає достовірній істинності, а **0** — достовірній хибності.

Нехай маємо два правила:

Правило 1. *Якщо очікується дощ, Іванов візьме з собою парасольку.*

Правило 2. *Якщо у Іванова буде багато речей, він не візьме з собою парасольку.*

Нехай дощ очікується з імовірністю 0.8, тоді за правилом 1 можна дійти висновку, що Іванов візьме парасольку, з мірою достовірності 0.8.

Нехай про те, що у Іванова буде багато речей, відомо з мірою достовірності 0.95. Тоді за правилом 2 міра достовірності того, що Іванов не візьме парасольку, оцінюється як 0.95, а міра достовірності протилежного прогнозу (*"Іванов візьме парасольку"*) — лише як 0.05.

Отже, ми маємо суперечливі свідоцтва, які дають майже протилежні прогнози. Як їх комбінувати? У найпростішому випадку можна взяти середнє арифметичне від обох свідоцтв і оцінити міру достовірності як $(0.8 + 0.05)/2 = 0.425$. Але можна брати і складніші функції комбінування свідоцтв. Зокрема, ми можемо враховувати міру надійності джерел інформації, а також те, чого Іванов більше не любить: мокнути під дощем чи носити з собою багато речей.

Якщо необхідно комбінувати різні свідоцтва, зменшується роль точкових оцінок мір достовірності і зростає значення інтервальних.

10.7. Приклади застосування мір достовірності

Може виникнути запитання: а що дає введення міри достовірності? Яка, наприклад, різниця, як ми оцінимо міру достовірності деякої події: як 0.3 чи як 0.7?

Можна навести як мінімум три ситуації, в яких більш-менш адекватна оцінка міри достовірності має велике практичне значення.

1. **"Об'єктивна" невизначеність; статистичний характер явищ, що досліджуються.** Нехай ми збираємося провести серію експериментів і оцінюємо успішність окремого експерименту з певною мірою достовірності. Тоді, якщо ця оцінка адекватна, ми можемо відразу спрогнозувати процент успіхів у серії експериментів. Наприклад, якщо міра достовірності успіху дорівнює 0.85, а проводиться 1000 експериментів, то ми можемо сказати, що приблизно 850 з них завершаться успішно.

2. **"Об'єктивна" невизначеність; чітка структурованість явищ, що досліджуються.** Нехай ми прогнозуємо деяке явище, яке залежить від певної кількості відносно контрольованих факторів (подібна ситуація виникає, наприклад, при прогнозі погоди, прогнозі соціально-економічних явищ і т. п.). Тоді, якщо ми знатимемо межі зміни кожного фактора, ми можемо більш-менш точно спрогнозувати і явище, яке нас цікавить.
3. **Прийняття рішень в умовах ризику і невизначеності.** Невизначеність при цьому може носити як об'єктивний, так і суб'єктивний характер.

Розглянемо типовий приклад, який пояснює ситуацію. Гравець на кінних перегонах (з точки зору теорії — особа, яка приймає рішення) може поставити 100 гривень на певного коня. Якщо кінь приходить до фінішу першим, гравець повертає свої 100 гривень і отримує додатковий виграш у 100 гривень. Якщо ж ні — гравець втрачає свої 100 гривень.

Яке ж рішення повинен прийняти гравець? Воно насамперед визначається мірою достовірності події. Якщо міра достовірності виграшу (об'єктивна чи суб'єктивна) оцінюється як 0.7, то гравець оцінює свої шанси на виграш як значні і має всі підстави зробити ставку. Якщо ж міра достовірності дорівнює 0.3, приймається протилежне рішення.

Точніше кажучи, такі міркування є повністю справедливими, якщо суб'єктивна оцінка гравця є адекватною, він має достатній капітал, і ситуація повторювана, тобто гравець може зробити повторну ставку ("повторити експеримент") достатню кількість разів. Нехай, наприклад, суб'єктивна міра достовірності виграшу (власна оцінка гравця) дорівнює 0.7. Якщо ця суб'єктивна оцінка адекватна, її можна вважати об'єктивною, і вона фактично дорівнює ймовірності виграшу. Тоді можна провести такий розрахунок. Якщо проводиться 1000 забігів, гравець виграє у середньому у 700 випадках і програє у середньому в 300 випадках. Тоді його сумарний виграш у середньому становитиме $(700 - 300) \cdot 1000 = 400\,000$ гривень.

Якщо ж ставку можна зробити лише один раз, тоді навіть за умови адекватності міри достовірності висновок стає менш однозначним. Якщо, наприклад, капітал гравця становить усього 125 гривень, а ввечері йому конче потрібно мати при собі 100 гривень, ставку не слід робити навіть за дуже високих шансів на виграш. Навпаки, якщо ввечері необхідно мати 200 гривень, є сенс ризикнути і зробити ставку навіть за дуже низьких шансів.

10.8. Деякі формалізації мір ризику за неточного логічного виведення

З попереднього параграфа випливає, що практичним критерієм вірності оцінки мір достовірності з точки зору ***особи, що приймає рішення***, повинен стати виграш у разі, якщо ці міри оцінюються вірно, та програш — у разі, якщо вони оцінюються невірно. Необхідна якась кількісна оцінка

функції виграшу, яка визначає виграш при прийнятті вірного рішення та програш — при прийнятті невірних рішень. Можна, крім того, сказати, що задана тим чи іншим чином функція виграшу є також мірою ризику, який зумовлений невірними оцінками.

Одна з класичних мір ризику пов'язана з прийняттям рішень про те, відбудеться чи не відбудеться деяка подія. Можливі два типи рішень:

β_0 — приймається рішення про те, що подія відбудеться;

β_1 — приймається рішення про те, що подія не відбудеться;

Можливі дві ситуації:

α_0 — подія насправді відбувається;

α_1 — подія насправді не відбувається.

Розглядається ***матриця виграшів*** (або ***матриця ризиків***):

$$\begin{array}{c} \\ \beta_0 \\ \beta_1 \end{array} \begin{array}{c} \begin{array}{cc} \alpha_0 & \alpha_1 \end{array} \\ \begin{pmatrix} c_{00} & c_{01} \\ c_{10} & c_{11} \end{pmatrix} \end{array}$$

Тут c_{ij} — виграш у разі, якщо приймається рішення β_i, а фактично має місце ситуація α_j.

Коефіцієнти c_{ij} можуть бути додатними, нульовими або від'ємними. Додатному коефіцієнту відповідає фактичний виграш, а від'ємному — фактичний програш. Слід розрізняти абсолютні та відносні виграші та програші. Так, виграш за невірного рішення може бути додатним, але меншим порівняно з тим, яким він міг би бути при вірному рішенні, і тоді йтиметься про відносний програш.

Слід зазначити, що коефіцієнти c_{ij} також можуть бути відомими неточно.

Очевидно, в переважній більшості випадків виконується властивість

$$c_{00} \geq c_{10};\ c_{01} \leq c_{11}$$

(правильне рішення повинно збільшити виграш порівняно з неправильним).

Спробуйте самостійно відповісти на такі запитання:

1. Чи обов'язково виконується рівність

$$c_{01} = c_{10}\ ?$$

Якщо ні, наведіть відповідні приклади.

2. Чи є обов'язковими співвідношення

$$c_{00} \geq c_{01};\ c_{10} \leq c_{11}\ ?$$

Далі, якщо міра об'єктивної невизначеності (ймовірність) події дорівнює λ, то ймовірність того, що подія не відбудеться, дорівнює $1 - \lambda$. Тоді очікувані виграші R_0 та R_1 при прийнятті рішень відповідно β_0 та β_1 можна оцінити як

$$R_0 = \lambda\, c_{00} + (1 - \lambda)\, c_{01};$$
$$R_1 = \lambda\, c_{10} + (1 - \lambda)\, c_{11}.$$

Тоді слід прийняти те рішення β_i, для якого відповідний виграш R_i набуває більшого значення. Існують інші критерії прийняття рішень в умовах невизначеності; огляд деяких з них можна знайти в [116, 194].

Інша можлива міра ризику пов'язана з оцінкою міри достовірності деякої події A (ця оцінка може носити суб'єктивний характер). Дану міру ризику можна задати функцією $g_A(\rho, \tau)$ — виграш у разі, якщо міра достовірності події A оцінюється як τ у той час, коли вона дорівнює ρ.

Видається доцільним висунути щодо введеної таким чином функції виграшу такі вимоги:

1) *Домінування вірних рішень:* для будь-яких ρ та τ виконується співвідношення

$$g_A(\rho, \tau) \le g_A(\rho, \rho).$$

2) *Монотонність:* для будь-яких ρ, τ_1 та τ_2 справедливим є твердження: якщо $\mu(\rho, \tau_1) \le \mu(\rho, \tau_2)$, то $g_A(\rho, \tau_1) \ge g_A(\rho, \tau_2)$. Тут $\mu(\rho, \tau)$ — міра близькості (відстань) між ρ та τ. Змістовно це означає, що чим точніше ми оцінили істинне значення міри достовірності, тим на більший виграш ми можемо розраховувати.

Ясно, що з властивості 2) випливає властивість 1); зворотне невірне.

В цілому вимоги 1) та 2) є досить розумними та реалістичними, хоча можна навести багато прикладів, коли вони обидві не виконуються. Спробуйте знайти кілька таких прикладів самостійно.

10.9. Деякі проблеми виведення

Нехай ми маємо продукційне правило $A \Rightarrow B$ *(якщо A, то B)*, при цьому коефіцієнт упевненості цього правила дорівнює γ. З погляду теорії ймовірностей цей коефіцієнт упевненості можна проінтерпретувати як $P(B|A)$ — умовну ймовірність B за умови A.

Нехай коефіцієнт упевненості твердження A дорівнює α. Тут ми розглядаємо коефіцієнти упевненості окремих тверджень як їх ймовірності.

Чому β дорівнює $= P(B)$ — коефіцієнт упевненості висновку B? Зразу ж необхідно сказати, **що ми не можемо обчислити $P(B)$ точно** — для цього не вистачає інформації. Натомість ми можемо обчислити **інтервал**, до якого потрапить ця ймовірність [175].

Очевидно, події A та $\overline{A}$ складають повну групу подій (див. п. 10.2). Тоді відповідно до формули повної ймовірності маємо:

$$\beta = P(B) = P(A)\,P(B|A) + P(\overline{A})\,P(B|\overline{A}) = \alpha\gamma + (1 - \alpha) \cdot P(B|\overline{A}).$$

У цій формулі фігурує невідоме значення $P(B|\overline{A})$, і саме тому точне обчислення β не є можливим. Але, оскільки $0 \le P(B|\overline{A}) \le 1$, маємо

$$\alpha\gamma \le P(B) \le \alpha\gamma + (1 - \alpha),$$

$$\beta \in [\alpha\gamma,\ \alpha\gamma + (1 - \alpha)].$$

Отже, інтервал невизначеності для висновку B є тим меншим, чим більшим є коефіцієнт упевненості умови A. Якщо $\alpha = 1$, β визначається точно.

Якщо ж $\alpha = 0$, інтервал невизначеності для β становить [0, 1], а це еквівалентно повній відсутності будь-якої корисної інформації.

Ми бачимо, що навіть у найпростіших випадках пряме застосування теоретико-ймовірнісних співвідношень спричиняє проблеми. Ситуація ще більше ускладнюється, якщо невизначеність носить "суб'єктивний" характер. Тому необхідно мати наближені, але простіші методики обчислення коефіцієнтів упевненості, які у більшості випадків давали б прийнятний результат.

10.10. Схема EMYCIN

Однією з найпростіших і найдавніших схем неточного логічного виведення була схема MYCIN, розроблена для однойменної експертної системи, що визначала наявність мікроорганізмів у крові. Згодом вона була модифікована, і ця модифікація дістала назву EMYCIN. Схема EMYCIN добре себе зарекомендувала та широко використовується і сьогодні.

У схемі EMYCIN з кожним твердженням A пов'язується коефіцієнт впевненості $\rho(A)$, який змінюється від -1 до 1. Значення -1 відповідає достовірній хибності, 0 — повній невизначеності, 1 — достовірній істинності. Хоча механізм EMYCIN тісно пов'язаний з ймовірнісними методами, зрозуміло, що введені таким чином коефіцієнти упевненості не є ймовірностями.

Якщо ми маємо правило $A \Rightarrow B$, коефіцієнт упевненості якого дорівнює $\rho(A \Rightarrow B)$, то **коефіцієнт упевненості висновку** дорівнює:

$$\rho(B) = \rho(A) \cdot \rho(A \Rightarrow B).$$

Коефіцієнт упевненості **заперечення** обчислюється за формулою:

$$\rho(\overline{A}) = -\rho(A).$$

Коефіцієнт упевненості **кон'юнкції** тверджень A та B обчислюється як мінімум коефіцієнтів упевненості окремих складових:

$$\rho(A \wedge B) = \min(\rho(A), \rho(B)).$$

Коефіцієнт упевненості **диз'юнкції** тверджень A та B можна обчислювати як максимум:

$$\rho(A \vee B) = \max(\rho(A), \rho(B)).$$

Але частіше правило $(A \vee B) \Rightarrow C$ замінюють двома правилами:

$$A \Rightarrow C,$$
$$B \Rightarrow C,$$

і далі застосовують формули комбінування свідоцтв, описані нижче.

Комбінування свідоцтв. Нехай ми маємо два правила зі спільною правою частиною:

$$A \Rightarrow C,$$
$$B \Rightarrow C.$$

З використанням раніше наведених формул ми можемо розрахувати два коефіцієнти упевненості для висновку C. Нехай за першим правилом

ми отримали коефіцієнт упевненості τ_1, а за другим — коефіцієнт упевненості τ_2. Як поєднати ці два коефіцієнти упевненості та отримати комбінований коефіцієнт τ?

Схема EMYCIN розрізняє три випадки:

Випадок 1: $\tau_1 \geq 0$, $\tau_2 \geq 0$. Це означає, що обидва правила свідчать в одному напрямі — на користь висновку, і, таким чином, підсилюють одне одного. Тоді комбінований коефіцієнт розраховується за формулою:

$$\tau = \tau_1 + \tau_2 - \tau_1\tau_2.$$

Випадок 2: $\tau_1 \leq 0$, $\tau_2 \leq 0$. Обидва правила свідчать проти висновку. Комбінований коефіцієнт розраховується за формулою

$$\tau = \tau_1 + \tau_2 + \tau_1\tau_2.$$

Випадок 3: коефіцієнти різного знака. Маємо суперечність: одне правило свідчить на користь висновку, а інше — проти нього. Тоді комбінований коефіцієнт обчислюється за формулою:

$$\tau = (\tau_1 + \tau_2) / (1 - \min(|\tau_1|, |\tau_2|))$$

(результат визначається вагомішим свідченням, але його вплив дещо послаблюється).

Сьогодні не існує абсолютно досконалого і загальноприйнятого механізму логічного виведення при недостовірних даних. Є кілька схем неточного логічного виведення, кожна з яких має свої переваги та недоліки [129].

Схема PROSPECTOR, як і MYCIN, була однією з перших. Методи логічного виведення, характерні для цієї схеми, повністю грунтуються на байєсівському підході.

Байєсівські методи дістали значного розвитку у ***схемі Піерла***, яка базується на механізмі ***байєсівських мереж***. Детальний опис логічного виведення на основі байєсівських мереж можна знайти в [320].

Теорія Демпстера-Шефера [121, 223] ставила за мету звільнитися від теоретико-ймовірнісних догматів і, таким чином, зробити неточне логічне виведення придатнішим для використання в умовах "суб'єктивної" невизначеності.

У [129] описуються також схема INFERNO, числення інцидентів та ін.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте поняття неточного логічного виведення.
2. Дайте класичне і частотне визначення ймовірності.
3. Що таке повна група подій?
4. Наведіть формулу повної ймовірності та формулу Байєса.
5. Охарактеризуйте поняття "коефіцієнт упевненості".
6. Що таке "об'єктивна" та "суб'єктивна" невизначеності? Який з цих типів невизначеностей має ймовірнісний характер?
7. Охарактеризуйте принцип індиферентності.
8. У чому полягає проблема комбінування свідоцтв?
9. Що таке інтервальні міри невизначеності?

10. Як пов'язані умовні ймовірності та правила виведення?
11. Чому застосування точних байєсівських методів, зокрема формули повної ймовірності, не завжди дозволяє отримати точний коефіцієнт упевненості?
12. Опишіть схему логічного виведення EMYCIN.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Наведіть власний приклад прийняття рішень в умовах невизначеності. Формалізуйте відповідні міри ризику.
2. Наведіть власний приклад розрахунку коефіцієнтів упевненості за схемою EMYCIN.

ЗАДАЧІ І ВПРАВИ

1. Нехай дано неточне правило *"Якщо Петров складе іспит, він влаштує святкову вечерю"* з коефіцієнтом упевненості 0.8.
   Нехай коефіцієнт упевненості того, що *Петров складе іспит*, становить 0.7. Розрахуйте інтервал, в який потрапить коефіцієнт упевненості висновку *"Петров влаштує святкову вечерю"*, на основі правила повної ймовірності.
2. Дані неточні правила:
   Якщо завтра буде дощ, Іванов візьме з собою парасольку (коефіцієнт упевненості — 0.7).
   Якщо у Іванова буде багато речей, він не візьме з собою парасольку (коефіцієнт упевненості — 0.3).
   Нехай про те, що *завтра буде дощ*, відомо з коефіцієнтом упевненості 0.8, а про те, що *у Іванова буде багато речей* — з коефіцієнтом упевненості 0.9. Розрахуйте коефіцієнт упевненості у тому, що *Іванов візьме з собою парасольку*, за правилами системи EMYCIN з використанням формул комбінування свідоцтв. (Не забувайте, що в EMYCIN достовірній істинності відповідає значення 1, а достовірній хибності — значення – 1.)
3. Дані неточні правила:
   Якщо у пацієнта грип, ліки будуть ефективними (коефіцієнт упевненості — 0.9).
   Якщо пацієнт буде дотримуватися режиму, ліки будуть ефективними (коефіцієнт упевненості — 0.7).
   Нехай про те, що *у пацієнта грип*, відомо з коефіцієнтом упевненості 0.8, а про те, що *пацієнт буде дотримуватися режиму* — з коефіцієнтом упевненості 0.6. Розрахуйте коефіцієнт упевненості у тому, що *ліки будуть ефективними*, за правилами системи EMYCIN з використанням формул комбінування свідоцтв. (Не забувайте, що в EMYCIN достовірній істинності відповідає значення 1, а достовірній хибності — значення – 1).

Розділ 11

ЛОГІЧНЕ ВИВЕДЕННЯ ЗА НЕЧІТКОЇ ІНФОРМАЦІЇ

Де початок того кінця, яким закінчується початок?

Афоризм

11.1. Інтуїтивне поняття нечіткості

Багатьом твердженням природної мови притаманна нечіткість, яка полягає в тому, що поняття, які фігурують у цих твердженнях, неможливо окреслити точно. Як можна, наприклад, визначити поняття *"маленьке число"* або *"цікава книга"*? Де закінчуються *"маленькі числа"* і починаються *"великі числа"*? Яку кількість камінців можна вважати *"купою"*? Легко уявити собі псевдоіндуктивні логічні висновки типу:

Один камінець не утворює купу.
Додавання одного камінця до "не купи" не перетворює її на купу.
Отже, будь-яка кількість камінців не утворює купу.

Для формалізації таких понять було запропоновано застосовувати апарат ***нечітких множин*** *(fuzzy sets),* точніше, нечітких підмножин. Інша назва нечітких множин — ***розмиті множини***. Нечітким множинам та їх можливим застосуванням присвячено багато літератури, наприклад, [2, 108, 150, 178, 225].

11.2. Функція належності як основна характеристика нечіткої множини

Якщо поняття не можна окреслити точно, то можна говорити про те, що даний об'єкт охоплюється цим поняттям тією чи іншою мірою. Наприклад, можливо сказати, що людина, зріст якої 135 см, *зовсім невисока*, людина зі зростом 170 см *більш-менш висока*, зі зростом 185 см — *досить висока* і т. п. Інакше кажучи, для кожного зросту можна задати **ступінь** того, наскільки такий зріст відповідає поняттю "висока людина".

Відомо, що зі звичайними, чіткими множинами тісно пов'язане поняття ***характеристичної функції***.

Характеристичною функцією $\chi_A(x)$ *називається функція, значення якої вказують, чи належить елемент* x *множині* A: $\chi_A(x) = 1$, *якщо* $x \in A$ *і* $\chi_A(x) = 0$, *якщо* $x \notin A$.

Професор Л. Заде у 1965 р. запропонував розширити поняття характеристичної функції. Він ввів поняття функції нечіткої належності $\mu_A(x)$ елемента до множини, дозволивши цій функції набувати будь-якого значення від 0 до 1. Чим більшим є значення функції належності, тим більшою мірою елемент належить до множини. Якщо $\mu_A(x) = 1$, елемент x чітко

належить множині A, якщо ж $\mu_A(x) = 0$, то елемент x чітко не належить цій множині. При $0 < \mu_A(x) < 1$ елемент належить множині нечітко. Чим вагомішим є значення функції належності, тим більшою мірою елемент належить до множини.

Наведемо формальне визначення.

Нехай E — довільна непуста множина. ***Нечіткою підмножиною A*** *множини E називається множина пар*

$$A = \{(x, \mu_A(x))\},$$

де $x \in E$, $\mu_A(x) \in [0, 1]$.

Функція $\mu_A : E \to [0, 1]$ називається ***функцією належності*** нечіткої підмножини A, а E — ***базовою множиною***, або ***базовою шкалою***. Значення $\mu_A(x)$, визначене для кожного $x \in E$, називається ***ступенем належності*** елемента x нечіткій множині E. Як правило, до A не включаються елементи з нульовим ступенем належності, хоч це і не є обов'язковим. ***Носієм*** нечіткої підмножини A називається чітка підмножина A^* множини E, що містить ті елементи з E, для яких $\mu_A(x) > 0$.

Таким чином, нечітка підмножина характеризується своєю функцією належності, яка визначає, якою мірою елемент базової множини належить нечіткій підмножині. Носієм нечіткої підмножини завжди є чітка звичайна підмножина базової множини. Тому правильніше говорити "нечітка підмножина", а не "нечітка множина". Втім ми часто вживатимемо термін "нечітка множина", якщо це не викликатиме непорозумінь.

Приклад 11.1. Формалізуємо поняття *"високий зріст"*. Для цього слід визначити функцію належності до відповідної нечіткої множини.

Очевидно, базовою множиною E є множина невід'ємних дійсних чисел. Функцію належності можна задати аналітичною формулою (рис. 11.1):

$$\mu_A(x) = \begin{cases} 0, & x < 1.5; \\ 2x - 3, & 1.5 \le x \le 2; \\ 1, & x > 2. \end{cases}$$

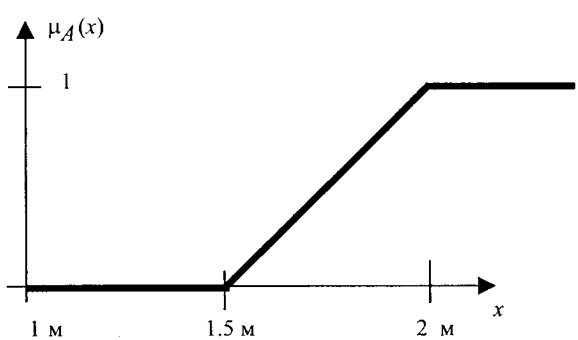


Рис. 11.1. Функція належності поняття "високий зріст"

Тоді відповіддю на запитання *"Чи є високою людина зі зростом 185 см?"* може бути відповідь *"Ступінь належності 0.7"*, що можна проінтерпретувати як *"досить висока"*.

Але цю саму нечітку множину можна задати і по-іншому, якщо явно перелічити деякі ступені належності, наприклад:

$A = \{(1.5, 0), (1.6, 0.2), (1.7, 0.4), (1.8, 0.6), (1.9, 0.8), (2, 1)\}$.

Необхідно звернути увагу на такі важливі моменти:

- **функція належності не може бути визначена однозначно, оскільки вона є дуже суб'єктивною;** кожна людина може запропонувати свої значення цієї функції;
- **самі ступені належності можна розглядати як розмиті.** Уявимо собі, що дехто оцінив зріст 185 см як високий зі ступенем належності 0.8. Якщо у нього спитати, чому 0.8, а не 0.85 чи 0.82, він не зможе чітко відповісти на це запитання. Тому замість конкретних числових значень ступенів належності можна говорити про **інтервали оцінок ступенів належності**. Якщо ступінь належності кожного елемента оцінюється інтервалом, говорять про ***нечіткі множини другого роду*** [225] з функцією належності $\mu_A : E \to [0, 1]^{[0, 1]}$. Такі міркування можна продовжити і далі. Втім на практиці найчастіше користуються описаними раніше нечіткими множинами першого роду (з конкретними числовими значеннями ступенів належності);
- **незважаючи на те, що функція належності є суб'єктивною оцінкою, її не можна вибирати довільно.** У нашому випадку функція належності, яка описує поняття *"високий зріст"*, повинна бути **неспадною**: неможливо, щоб хтось оцінював зріст 165 см більшим ступенем належності, ніж зріст 195 см. Деякі міркування з цього приводу можна знайти в [199].

11.3. Основні операції над нечіткими множинами

Над нечіткими множинами може бути визначено досить багато різноманітних операцій. Розглянемо деякі з них.

Рівність. Дві нечіткі підмножини базової множини E називаються рівними між собою, якщо для будь-якого $x \in E$ $\mu_A(x) = \mu_B(x)$. Позначатимемо це як $A = B$.

Включення. Будемо казати, що A є підмножиною B (позначається $A \subseteq B$), якщо для будь-якого $x \in E$ $\mu_A(x) \le \mu_B(x)$.

Перетин. Перетином двох нечітких множин A і B називається нечітка множина $C = A \cap B$ з функцією належності

$$\mu_C(x) = \min(\mu_A(x), \mu_B(x)).$$

Об'єднання. Об'єднанням двох нечітких множин A і B називається нечітка множина $C = A \cup B$ з функцією належності

$$\mu_C(x) = \max(\mu_A(x), \mu_B(x)).$$

Доповнення. Нечітку множину B називатимемо доповненням нечіткої множини A, якщо

$$\mu_B(x) = 1 - \mu_A(x).$$

Для нечітких множин виконуються всі основні властивості звичайних чітких множин, крім двох: об'єднання $A \cup \overline{A}$ не обов'язково дорівнює базовій множині; перетин $A \cap \overline{A}$ не обов'язково дорівнює порожній множині.

11.4. Нечітке логічне виведення

Одну з найтиповіших задач логічного виведення за умов нечіткості можна сформулювати так:

Дано (нечітке) продукційне правило *"Якщо A, то B"*.
Спостерігається *A'* (*A* в певній мірі).
Яким повинно бути *B?*

Формальніше, задані носії U і V та їх нечіткі підмножини A і B. Задано або елемент $u \in U$, або множина $A' \subseteq U$. Потрібно визначити елемент $v \in V$, що являє собою висновок системи, який визначає результат застосування нечіткого продукційного правила.

Приклад 11.2. Дане нечітке продукційне правило:
Якщо студент багато працює в бібліотеці, він отримає високу оцінку.
Як множину U ми розглядаємо множину чисел, що визначають кількість годин на тиждень, які студент може проводити в бібліотеці. Можна взяти U як діапазон чисел від 0 до 30. Для простоти обмежимося невеликою кількістю можливих значень: $U = \{0, 3, 6, 9, 12, 18, 21, 27\}$.

Аналогічно, якщо оцінки (рейтинги) виставляються за 100-бальною шкалою, за V можна взяти діапазон чисел від 59 до 100. Знову ж таки, обмежимося невеликим набором можливих значень: $V = \{59, 72, 84, 91, 96, 100\}$.

Задамо функції належності для нечітких множин *A ("багато працює в бібліотеці")* та *B ("високий рейтинг")* таким чином:
$A = \{(3, 0); (6, 0.1); (9, 0.4); (12, 0.6); (18, 0.8); (21, 1); (27, 1)\}$;
$B = \{(59, 0); (72, 0.2); (84, 0.4); (91, 0.7); (96, 0.9); (100, 1)\}$.

Нехай явним чином задана кількість годин, які студент працює в бібліотеці, або ступінь належності, що визначає, чи багато він працює в бібліотеці (це практично еквівалентно, оскільки, знаючи конкретну кількість годин, ми завжди можемо визначити відповідний ступінь належності). Нехай цей ступінь належності дорівнює α. Тоді для отримання висновку можна застосувати ***метод простої підстановки нечіткого значення***, відповідно до якого v вибирається з умови:

$$\mu_B(v) = \alpha.$$

Нехай у нашому прикладі дано, що *студент працює в бібліотеці 9 годин*. Ступінь належності дорівнює 0.4, і система повинна дійти висновку, що такий *студент повинен отримати оцінку 84 ("добре")*. Якби в нашій спрощеній множині V не виявилося значення точно з таким ступенем належності, слід було б провести інтерполяцію або взяти найближче значення.

11.5. Метод центру тяжіння композиції максимум-мінімум

Як повинно змінитися нечітке логічне виведення, якщо A' задається не як конкретне значення, а як нечітка множина (наприклад, якщо у вищенаведеному прикладі відомо, що *студент проводить в бібліотеці середню кількість часу*)? Для цього необхідно залучити до розгляду ***відношення нечіткої імплікації***.

Нечітким відношенням *між множинами A та B називається нечітка підмножина їх декартового добутку. Інакше кажучи, якщо*

$$A = \{(u_i, \mu_A(u_i)), u_i \in U\}, B = \{(v_j, \mu_B(v_j)), v_j \in V\},$$

то відношення ARB визначається як множина пар

$$ARB = \{((u_i, v_j), \mu_R(u_i, v_j)), u_i \in U, v_j \in V\}.$$

Відношення нечіткої імплікації $A \to B$ можна вводити по-різному. Часто використовується формула ***min-імплікації***:

$$\mu_{A\to B}^{(\min)}(u_i, v_j) = \min(\mu_A(u_i), \mu_B(v_j)).$$

Для задання імплікації застосовуються й інші формули:

- ***нечітке розширення класичної імплікації:***

$$\mu_{A\to B}^{(Kl)}(u_i, v_j) = \max(\mu_B(v_j), 1-\mu_A(u_i));$$

- ***нечітка імплікація Лукасевича:***

$$\mu_{A\to B}^{(Luc)}(u_i, v_j) = \min(1, 1-\mu_A(u_i)+\mu_B(v_j)).$$

Тепер ми можемо отримати множину B' — нечітку множину висновків, які відповідають множині A'. Ця множина є результатом ***композиції max-min*** множини A' і нечіткої імплікації:

$$B' = A' \bullet (A \to B),$$

де $\bullet$ — знак max-min композиції, що обчислюється за формулою

$$\mu_{B'}(v_j) = \max_{u_i} \min(\mu_{A'}(u_i), \mu_{A\to B}(u_i, v_j)).$$

Але отримати одну лише множину B' недостатньо, треба ще знайти конкретну числову відповідь (кажуть, що треба провести ***дефадзифікацію***). Найчастіше за числову відповідь береться ***центр тяжіння*** знайденої нечіткої множини, який обчислюється за формулою:

$$v^* = \frac{\sum v_j \mu_{B'}(v_j)}{\sum \mu_{B'}(v_j)}.$$

Увесь описаний метод нечіткого логічного виведення часто називають ***методом центру тяжіння композиції максимум-мінімум***.

Повернемося до прикладу 11.2.

Ми задали функції належності для нечітких множин A *("багато працює в бібліотеці")* та B *("високий рейтинг")* таким чином:

A = {(3, 0); (6, 0.1); (9, 0.4); (12, 0.6); (18, 0.8); (21, 1); (27, 1)};

B = {(59, 0); (72, 0.2); (84, 0.4); (91, 0.7); (96, 0.9); (100, 1)}.

Нехай дано, що *студент працює в бібліотеці середню кількість часу.* Задамо нечітку множину A' *(середня кількість часу)*:

A' = {(3, 0); (6, 0.2); (9, 0.7); (12, 1); (18, 0.6); (21, 0.2); (27, 0)}.

Обчислимо відношення min-імплікації нечітких множин A та B:

		59	72	84	91	96	100
	$A \backslash B$	0	0.2	0.4	0.7	0.9	1
3	0	0	0	0	0	0	0
6	0.1	0	0.1	0.1	0.1	0.1	0.1
9	0.4	0	0.2	0.4	0.4	0.4	0.4
12	0.6	0	0.2	0.4	0.6	0.6	0.6
18	0.8	0	0.2	0.4	0.7	0.8	0.8
21	1	0	0.2	0.4	0.7	0.9	1
27	1	0	0.2	0.4	0.7	0.9	1

Знайдемо max-min-композицію множини A' та щойно знайденого відношення $A \to B$. Результатом буде нечітка множина B':

		59	72	84	91	96	100
	$A \backslash B$	0	0.2	0.4	0.7	0.9	1
3	0	0	0	0	0	0	0
6	0.2	0	0.1	0.1	0.1	0.1	0.1
9	0.7	0	0.2	0.4	0.4	0.4	0.4
12	1	0	0.2	0.4	0.6	0.6	0.6
18	0.6	0	0.2	0.4	0.6	0.6	0.6
21	0.2	0	0.2	0.2	0.2	0.2	0.2
27	0	0	0	0	0	0	0
	B'	0	0.2	0.4	0.6	0.6	0.6

Нарешті, проведемо дефадзифікацію отриманої множини B':

$\nu^* = (72 \cdot 0.2 + 84 \cdot 0.4 + 91 \cdot 0.6 + 96 \cdot 0.6 + 100 \cdot 0.6) / (0.2 + 0.4 + 0.6 + 0.6 + 0.6) = 220.2 / 2.4 = 91.75 \approx 92$.

Отже, на основі проведеного розрахунку із застосуванням нечіткого композиційного правила виведення можна дійти висновку, що ***студент повинен отримати оцінку 92 бали.***

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте поняття нечіткості.
2. Дайте визначення характеристичної функції.
3. Дайте визначення нечіткої функції належності.

4. Чому нечітку функцію належності можна розглядати як узагальнення характеристичної функції?
5. Дайте визначення нечіткої множини. Наведіть кілька власних прикладів.
6. Дайте визначення рівності та включення нечітких множин.
7. Дайте визначення доповнення нечіткої множини.
8. Дайте визначення об'єднання та перетину нечітких множин.
9. Охарактеризуйте задачі, пов'язані з нечітким логічним виведенням.
10. Опишіть метод простої підстановки нечіткого значення, який може бути застосований у разі, якщо в умові явно задається ступінь належності.
11. Опишіть метод центру тяжіння композиції максимум-мінімум.
12. Що таке нечітке відношення?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Порівняйте міри нечіткості та ймовірності. Що між ними спільного і чим вони відрізняються?
2. Яке з відомих відношень нечіткої імплікації дозволяє отримувати кращі результати?

ЗАДАЧІ І ВПРАВИ

1. Повторіть розрахунки, наведені в пп. 11.4—11.6, але із застосуванням імплікації Лукасевича та нечіткого розширення класичної імплікації.
2. Дано нечітке продукційне правило:
   Якщо тварина їсть багато м'яса, вона виросте великою.
   Нехай дано, що *тварина їсть середню кількість м'яса*. Підрахуйте, якою вона виросте, за правилом центру тяжіння композиції максимум-мінімум. Функції належності візьміть на свій розсуд.
3. Покажіть, що при $\mu_B(v) \geq \mu_A(u)$:
   а) $\mu_{A\to B}^{(\min)}(u,v) = \mu_A(u)$;
   б) $\mu_{A\to B}^{(Luc)}(u,v) = 1$;
   в) $\mu_{A\to B}^{(Kl)}(u,v) \geq 0.5$.

ЧАСТИНА IV МОДЕЛІ І МЕТОДИ ПРИЙНЯТТЯ РІШЕНЬ

Розділ 12

ОСНОВНІ ПІДХОДИ ДО ПЛАНУВАННЯ ЦІЛЕСПРЯМОВАНИХ ДІЙ

Розвиток має бути
планомірним і пропорційним.
З матеріалів з'їздів КПРС

12.1. Планування цілеспрямованих дій і прийняття рішень

Як ми вже зазначали, функціонування будь-якої інтелектуальної системи полягає в тому, що вона сприймає зовнішню ситуацію і певним чином реагує на неї. Проаналізувавши ситуацію, вона повинна **прийняти рішення про вибір певної дії**, виходячи з власної мети. Але таке рішення — це заключний етап процесу планування. ***Планування цілеспрямованих дій*** полягає в аналізі всіх можливих дій системи та наслідків цих дій. Аналізуючи можливі наслідки, система оцінює один з них як найсприятливіший для себе і вибирає ту дію, яка, на думку системи, повинна привести до очікуваного результату.

Широкий клас завдань у рамках прийняття рішень може бути сформульовано у вигляді класичної оптимізаційної задачі: знайти рішення, за яким деяка цільова функція досягає свого максимуму при заданих обмеженнях. Так, керівник фірми бажає максимізувати свій дохід, не порушуючи при цьому закони. Або потрібно якнайшвидше посадити космічний корабель на Марсі, але так, щоб сила удару була не надто великою.

Подібні задачі є предметом науки, яка називається ***дослідженням операцій*** [116, 151, 164]. Формальніше оптимізаційну задачу можна сформулювати в такому вигляді:

знайти $x = (x_1, \ldots, x_n)$, *за якого функція* $f(x)$ *досягає максимуму і задовольняються обмеження* $g_i(x) \geq 0$.

Функція $f(x)$ називається ***цільовою функцією***, а функції $g_i(x)$ — ***обмеженнями*** оптимізаційної задачі.

Приклад 12.1. Мандрівник збирається в дорогу. Він може взяти в рюкзак певну кількість предметів різних типів. Нехай є n типів предметів; наявна кількість i-го предмета становить r_i. Кожний предмет має власну

цінність c_i та вагу q_i. Потрібно зібрати рюкзак таким чином, щоб сумарна цінність узятих предметів була максимальною, але щоб сумарна вага не перевищувала заданої межі u.

Формалізуємо цю постановку. Позначимо через x_i кількість предметів i-го типу, що беруться в дорогу. Тоді очевидним чином отримуємо математичну постановку задачі:

Знайти $x = (x_1, \ldots, x_n)$, *для якого* $\sum_{i=1}^{n} c_i x_i \to \max$ *та задовольняються обмеження* $\sum_{i=1}^{n} q_i x_i \le u$; x_i — *цілі*; $x_i \ge 0$; $x_i \le r_i$.

Будь-який елемент x, який задовольняє обмеженням $g_i(x) \ge 0$, називається ***допустимим рішенням*** задачі. Якщо умова максимізації не висувається, йдеться про ***задачу пошуку допустимих рішень***. Якщо обмежень немає, йдеться про ***безумовну оптимізацію***.

Умова максимізації цільової функції ніяк не звужує загальності постановки. Дійсно, якщо треба вирішити задачу *мінімізації* функції $g(x)$, ми завжди можемо поміняти знак цієї функції і вирішувати задачу *максимізації* функції $h(x) = -g(x)$.

Залежно від вигляду цільової функції та функцій-обмежень розрізняють неоднакові часткові випадки оптимізаційних задач, для кожного з яких існують свої методи вирішення. Так, оптимізаційна задача з лінійною цільовою функцією та лінійними обмеженнями називається ***задачею лінійного програмування***. ***Нелінійне програмування*** є складнішим розділом дослідження операцій. Якщо всі x_i цілі, йдеться про ***дискретне програмування***. Зокрема, саме до задач дискретного програмування належить згадана задача про рюкзак.

Серед багатьох ідей, які застосовуються для знаходження максимумів функцій і функціоналів, можна, мабуть, виокремити такі: лінійне програмування [164], принцип максимуму Понтрягіна [214] і теорію локальних екстремумів [189].

Поряд з класичними ідеями оптимізації в останні десятиріччя почали розвиватись ідеї іншої природи — послідовного аналізу варіантів, їх відбракування, послідовного звуження множини можливих розв'язків.

На важливість розгляду оптимізаційних задач вказував ще Д. Гілберт. Так, розглядаючи свою знамениту програму, він плекав надію, що в XX столітті математики опанують методи розв'язку оптимізаційних задач. Це дійсно проблема, оскільки обчислювальні характеристики більшості оптимізаційних задач відносять їх до класу *NP*-повних задач. Отже, для їх розв'язків бажано використовувати і наближені евристичні алгоритми.

Останні значні успіхи було досягнуто в тому розділі теоретичної інформатики, який дістав назву "Конструювання і аналіз комбінаторних алгоритмів". Ефективні комбінаторні алгоритми знаходять застосування в багатьох областях символьної обробки інформації та в дискретній оптимізації.

12.2. Повний перебір як найочевидніший метод вирішення оптимізаційної задачі

Існує очевидний і досить універсальний метод вирішення оптимізаційних задач, який може бути застосований, якщо множина припустимих рішень M обмежена. Це — метод ***повного перебору***, який полягає у переборі всіх можливих варіантів. Метод дає гарантований розв'язок, якщо множина M скінченна (ситуація, характерна для дискретного програмування), та існує ефективний алгоритм породження будь-якого елемента з M та обчислення на цьому елементі цільової функції.

Таким чином, ми маємо типову загальноінтелектуальну процедуру. Якщо інтелектуальна система потрапляє в нову ситуацію і намагається планувати подальші дії, вона може спробувати звести задачу планування цілеспрямованих дій до оптимізаційної задачі (для цього, очевидно, достатньо визначити множину припустимих рішень M і цільову функцію $f(x)$). Якщо це вдається, повний перебір варіантів у більшості випадків дозволить отримати оптимальне рішення.

Але повний перебір має очевидний недолік: для більшості практичних ситуацій кількість варіантів, які доводиться перебирати, надто велика, і реалізувати метод за прийнятний час не вбачається можливим. Тому розвиваються методи і алгоритми, які дозволяють скоротити такий перебір.

У той самий час далеко не завжди слід шукати саме оптимальне рішення. Часто достатньо обмежитися рішенням субоптимальним (наближеним до оптимального) або просто припустимим. У таких випадках достатньо виробити якесь правило, яке б виключало застосування повного перебору і для більшості вхідних даних давало б прийнятний результат. У цьому полягає суть ***евристичних алгоритмів***, про які йдеться в розд. 21.

Втім далеко не будь-яка інтелектуальна задача припускає очевидне зведення до оптимізаційної, оскільки часто не вдається записати в явному вигляді цільову функцію або обмеження.

12.3. Евристичний пошук

Багато практичних задач може бути вирішено на основі ***евристичного пошуку***, яким прийнято називати процедуру систематизованого перебору на основі послідовного аналізу можливих варіантів та оцінки їх наслідків. Ґрунтуючись на основі [168], можна навести таку загальну схему евристичного пошуку:

1. *Вибрати певну дію з області можливих дій.*
2. *Здійснити дію; це приведе до зміни ситуації.*
3. *Оцінити нову ситуацію.*
4. *Якщо досягнуто успіху — кінець; якщо ні — повернутися на крок* 1 *і розпочати спочатку.*

Здійснити дію можна реально (і тоді йдеться про стратегію спроб і помилок) або уявно (і тоді можна говорити про попереднє планування, яке повинно передувати дійсному прийняттю рішення).

Одним з важливих варіантів евристичного пошуку є ***метод послідовних поліпшень***. Основну його ідею можна сформулювати так: якщо ми маємо певне наближення до рішення x^k, ми застосовуємо деяку процедуру для переходу до іншого наближення x^{k+1}, яке буде кращим за попереднє. На методі послідовних поліпшень ґрунтується більшість основних методів дослідження операцій.

12.4. Експоненційна складність евристичного пошуку

Розглянемо **задачу про виконуваність булевого виразу**, яка формулюється так: для будь-якого булевого виразу від n змінних знайти хоча б один набір значень змінних $(x_1, \ldots, x_n)$, за якого цей вираз приймає значення 1.

Найпростіша схема евристичного пошуку для цієї задачі може мати такий вигляд: спочатку надаємо одне з двох можливих значень (0 або 1) першій змінній x_1, потім другій і т. д. Коли будуть розставлені значення всіх змінних, ми можемо визначити значення булевого виразу. Якщо він дорівнює 1, задача вирішена. Якщо ні — потрібно повернутися назад і змінити значення деяких змінних.

Можна інтерпретувати цю задачу як задачу пошуку на певному ***дереві перебору***. Кожна вершина цього дерева відповідає певному набору встановлених значень $x_1, \ldots, x_k$. Ми можемо встановити змінну x_{k+1} в 0 або 1, тобто маємо вибір з двох дій. Тоді кожній з цих дій відповідає одна з двох дуг, які йдуть від цієї вершини. Вершина $x_1, \ldots, x_k$ має двох синів $x_1, \ldots, x_k$ 0 і $x_1, \ldots, x_k$ 1.

При $n = 3$ дерево можливостей матиме вигляд, показаний на рис. 12.1 (вершини помічені наборами значень змінних, а дуги — рішеннями, які приймаються на черговому кроці). Вершини, що відповідають рішенням задачі для виразу $(x_1 x_2) \equiv x_3$, зафарбовані.

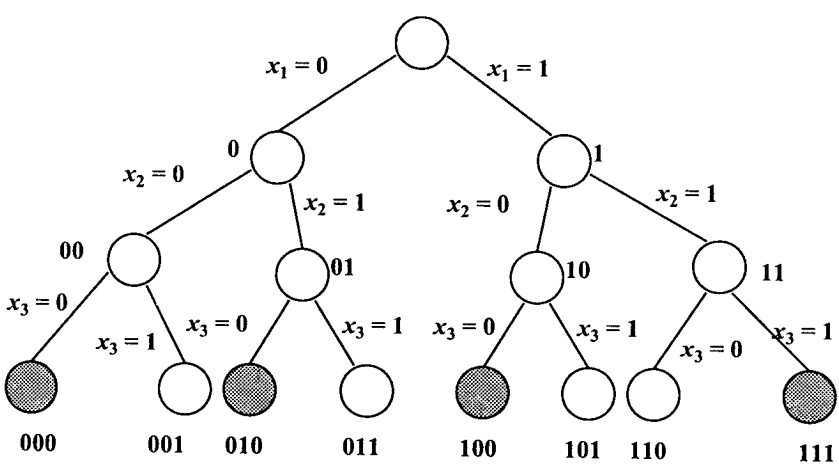


Рис. 12.1. Дерево перебору для задачі про виконуваність булевого виразу

З рисунка видно, що **вирішення задачі зводиться до перебору листів дерева** з метою виявлення, які з них відповідають наборам, що перетворюють заданий булевий вираз на 1. Для виразу $(x_1\ x_2) \equiv x_3$ такими наборами будуть 000, 010, 100, 111.

Якщо $n = 3$, дерево має $2^3 = 8$ листів. У загальному ж випадку кількість можливих варіантів дорівнює 2^n. Цей вираз експоненційно залежить від n, і перебірний алгоритм має експоненційний характер.

Слід звернути увагу на те, що не будь-який перебірний алгоритм є експоненційним. Як приклад можна навести загальновідомий алгоритм пошуку найбільшого елемента в масиві. Незважаючи на його перебірний характер, він є не експоненційним, а лінійним.

Зі зростанням розмірності будь-який поліноміальний алгоритм стає ефективнішим, ніж будь-який експоненційний. Для лінійного алгоритму зростання швидкодії комп'ютера в 10 разів дозволяє за той же час розв'язати задачу, розмір якої в 10 разів перевищує попередню. Для експоненційного алгоритму з основою 2 цей же розмір можна збільшити лише на 3 одиниці.

12.5. Пошук у глибину і пошук у ширину

Серед різноманітних методів, за допомогою яких можна організовувати пошук потрібної вершини на дереві перебору, а відтак і шляху до неї, прийнято виокремлювати дві основні стратегії:

- ***пошук у ширину***;
- ***пошук у глибину*** (інші назви — ***перебір з поверненнями***; ***бектрекінг***).

Процедура пошуку в ширину передбачає аналіз на кожному кроці синів усіх вершин, що були проаналізовані до цього (в інтерпретації прийняття рішень це означає **паралельну** перевірку всіх можливих альтернатив).

Процедура пошуку в глибину передбачає першочерговий аналіз нащадків тих вершин, що були проаналізовані останніми. Це означає, що всі альтернативи аналізуються **послідовно**, одна за одною; аналіз деякої альтернативи завершується лише тоді, коли вдається остаточно встановити, приводить вона до успіху чи ні. Якщо ж альтернатива зумовлює невдачу, відбувається **повернення** і розгляд інших альтернатив.

На рис. 12.2 показана послідовність перегляду вершин при пошуку в ширину, а на рис. 12.3 — при пошуку в глибину.

Можна навести ряд прикладів, коли перебір у глибину дозволяє виграти час порівняно з перебором в ширину і навпаки. Як правило, пошук у глибину дозволяє зекономити **пам'ять**, оскільки при його реалізації немає необхідності запам'ятовувати все дерево, достатньо зберігати в пам'яті лише вершини, що мають відношення до поточної альтернативи.

На практиці процедура пошуку в глибину набула значнішого поширення. Можна стверджувати, що перебір з поверненням став класичною загальноінтелектуальною процедурою, що лягла в основу сучасних методик

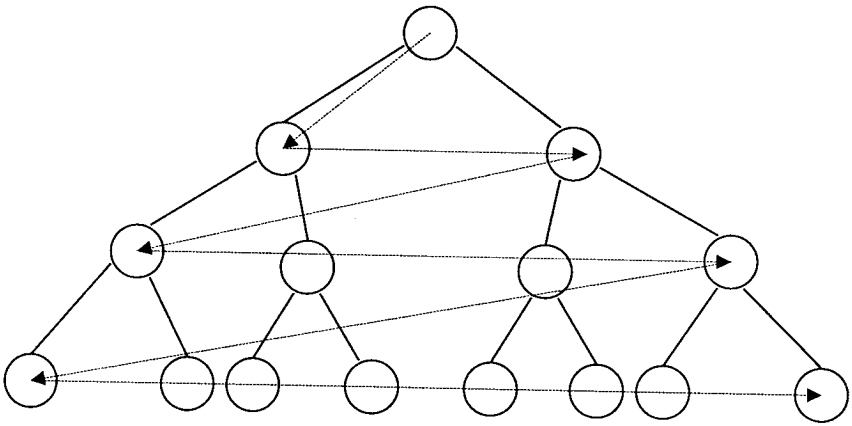


Рис. 12.2. Процедура пошуку в ширину

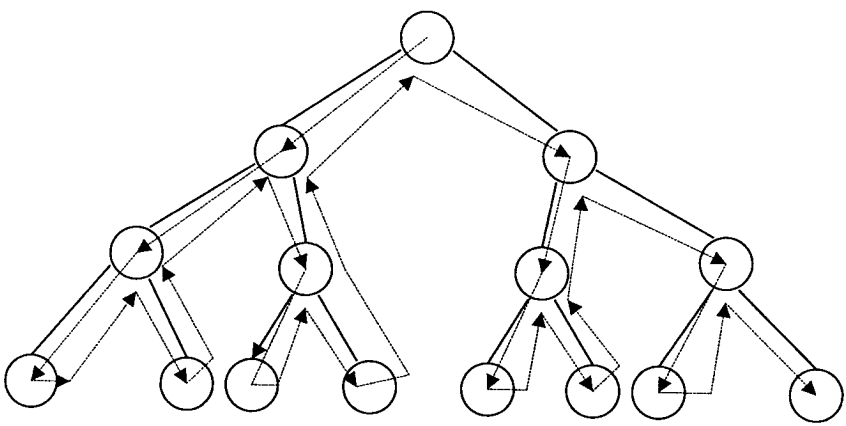


Рис. 12.3. Процедура пошуку в глибину

планування цілеспрямованих дій, програмування ігор, автоматизованого доведення теорем тощо.

12.6. Простір задач і простір станів

Методики планування цілеспрямованих дій прийнято розподіляти на два класи [129]: планування в просторі станів і планування в просторі задач.

При ***плануванні в просторі станів*** заданим вважається деякий набір станів (ситуацій). Відомі дії, які може здійснювати система та які визначають перехід з одного стану до іншого.

Графом станів задачі *називається орієнтований граф, вершини якого відповідають можливим станам предметної області, а дуги — методам переходу від стану до стану.*

Дуги можуть мати мітки, які інтерпретуються як вартість або довжина відповідного переходу. **Тоді вирішення задачі являє собою пошук шляху від початкового стану до цільового; при цьому типовою є вимога оптимізації даного рішення, тобто пошуку найкоротшого шляху.** Відповідно, після того як зведення задачі до формальної моделі проведено, можна використовувати відомі алгоритми пошуку шляхів на графах (алгоритми Мура, Дейкстри, гілок і границь і т. п.).

Класичним прикладом планування в просторі станів можна вважати задачу пошуку шляху від одного міста до іншого. Можна навести й інші, менш очевидні приклади. Детальніше планування в просторі станів описується в розд. 14.

Планування в просторі задач передбачає декомпозицію початкової задачі на підзадачі, поки не буде досягнуто зведення до задач, для яких є готові алгоритми вирішення. Таку декомпозицію зручно уявляти собі у вигляді ***І/АБО-графа***. Пошуку в просторі задач присвячено розд. 15.

Існують методики комбінування методів пошуку в просторі станів і в просторі задач; деякі підходи до цього описані в [113].

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте роль планування цілеспрямованих дій і прийняття рішень у загальній схемі функціонування інтелектуальної системи.
2. Яким чином можна звести задачу прийняття рішень до оптимізаційної задачі? Наведіть власний приклад.
3. Дайте визначення оптимального та припустимого вирішення оптимізаційної задачі.
4. Охарактеризуйте поняття "повний перебір".
5. Охарактеризуйте поняття евристичного пошуку.
6. Дайте визначення часової та ємнісної складності задачі.
7. Наведіть відому вам класифікацію алгоритмів за часовою складністю.
8. Чому евристичний пошук може мати експоненційну часову складність? Наведіть приклад.
9. Чи будь-який перебірний алгоритм є експоненційним?
10. Опишіть процедури пошуку в глибину та пошуку в ширину.
11. Охарактеризуйте планування в просторі задач і просторі станів.
12. Що таке граф станів задачі?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Наведіть власні приклади обмежувальних правил, які істотно скорочували б перебір і дозволяли б знаходити для тих чи інших конкретних задач прийнятні, хоча й неоптимальні рішення за прийнятний час.
2. Наведіть власний приклад задачі, яку можна вирішувати в просторі станів. Намалюйте граф станів цієї задачі.
3. Порівняйте пошук у глибину з пошуком у ширину. Наведіть приклади, коли одна зі стратегій має незаперечні переваги над іншою.

Розділ 13

АНАЛІЗ СКЛАДНОСТІ АЛГОРИТМІВ РОЗВ'ЯЗКУ ІНТЕЛЕКТУАЛЬНИХ ЗАДАЧ

Сложно... но можно.
З реклами

13.1. Автоматний спосіб задання алгоритму

Одним з основних обчислювальних пристроїв, які використовуються при побудові інтелектуальних систем, є комп'ютер. У свою чергу, в основі обчислювальної частини традиційного комп'ютера (електронно-обчислювального пристрою) повинні бути перетворювачі, що сприймають деякі вхідні послідовності нулів та одиниць і перетворюють їх на вихідні послідовності, тобто здійснюють інформаційні перетворення, в тому числі арифметичні операції. Такі перетворювачі називають ***логічними схемами***.

Логічні схеми можна розділити на два класи. Перший з них — ***комбінаційні схеми***. Значення вихідних змінних комбінаційної схеми визначаються лише значенням вхідних змінних і не залежать від поточного стану схеми. Інший клас логічних схем — схеми з пам'яттю.

Будь-яка комбінаційна схема реалізує ту чи іншу булеву функцію від своїх вхідних змінних. Булевою називається функція, аргументи і результат якої можуть приймати одне з двох значень: 0 або 1. Змінна, що може приймати тільки значення 0 або 1, називається булевою змінною.

На відміну від комбінаційної схеми, скінченний автомат є перетворювачем, вихід якого залежить не тільки від вхідних і вихідних сигналів, але й від поточного стану автомата, причому кількість вхідних і вихідних змінних, кількість можливих значень цих змінних, а також кількість можливих станів автомата скінченна. Поточний стан автомата зберігається в його пам'яті.

Скінченні автомати мають досить обмежені обчислювальні можливості. Ширші вони в автоматів з нескінченною пам'яттю магазинного типу, але й ці автомати є недостатньо універсальними. Найуніверсальнішою моделлю комп'ютерних обчислень є машина Тьюринга.

Автомати можна розглядати як машини, що мають пам'ять у вигляді стрічки та пристрій для читання інформації з неї. Стрічка розділена на квадрати, в яких розміщуються визначені символи. Автомат, проглядаючи ці квадрати по черзі по одному, виконує обчислення і розв'язує задачі [173].

Взагалі говоритимемо, що автомати — це пристрої скінченного розміру, яким притаманна така властивість: визначений вихідний сигнал є функцією обробки вхідного сигналу.

Формально найпростіший з автоматів — скінченний автомат — можна описати так:

$$FA = <Q, A, b, q_0, F>,$$

де $Q = (q_1, q_2, ..., q_n)$ — скінченна множина станів керування; $A = (a_1, a_2, ..., a_m)$ — вхідний алфавіт; $b: Q \times A \to Q$ — функція переходів; q_0 — початковий стан; $F \subseteq Q$ — множина заключних станів керування.

Приклад 1. Нехай нам потрібно побудувати скінченний автомат-розпізнавач. Він мусить допускати тільки послідовності символів 0 і 1, при цьому в послідовності розпізнавання кількість одиниць повинна бути парною, а нулів — непарною.

Згідно з умовою задачі A складатиметься з символів 0, 1, #, де # позначатиме довільний символ, відмінний від 0 і 1. Тобто, $A = <0, 1, \#>$.

Множину станів Q виберемо так: $Q = <q_0, q_1, q_2, q_3, q_4, q_5>$ де q_0 — початковий стан; q_1 — стан помилки; q_2 — стан, який визначає, що в послідовності кількість нулів непарна, а одиниць — парна (нуль вважатимемо числом парним); q_3 — стан, який визначає, що в послідовності кількість нулів і одиниць непарна; q_4 — стан, який визначає, що в послідовності кількість нулів і одиниць парна; q_5 — стан, який визначає, що в послідовності кількість нулів парна, а одиниць — непарна. Згідно з умовою задачі $F = <q_2>$.

Функція переходів визначається так:

$q_0\# \to q_1$	$q_00 \to q_2$	$q_30 \to q_5$
$q_1\# \to q_1$	$q_01 \to q_5$	$q_31 \to q_2$
$q_2\# \to q_1$	$q_10 \to q_1$	$q_40 \to q_2$
$q_3\# \to q_1$	$q_11 \to q_1$	$q_41 \to q_5$
$q_4\# \to q_1$	$q_20 \to q_4$	$q_50 \to q_3$
$q_5\# \to q_1$	$q_21 \to q_3$	$q_51 \to q_4$

Автомати зручно задавати табличним або графічним способом. Один з варіантів табличного задання нашого автомата наводиться на рис. 13.1.

	#	0	1	
q_0	q_1	q_2	q_5	
q_1	q_1	q_1	q_1	
q_2	q_1	q_4	q_3	←
q_3	q_1	q_5	q_2	
q_4	q_1	q_2	q_5	
q_5	q_1	q_3	q_4	

Рис. 13.1. Табличне задання автомата

Рядки таблиці позначаються станами множини Q, а стовпці — символами вхідного алфавіту. Символ ← позначає заключні допустимі стани.

На рис. 13.2. демонструється графічний спосіб опису функціонування автомата, де вершини графа q_i та q_j з'єднуються спрямованою дугою, позначеною символом a, якщо автомат має команду: $q_i a \to q_j$. Заключні допустимі вершини позначаються подвійним колом.

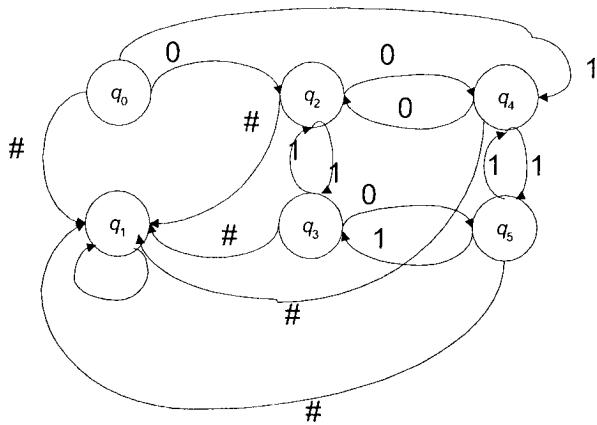


Рис. 13.2. Графічне задання автомата

Теорія автоматів може бути використана і в нетрадиційних сферах, наприклад, для вирішення проблеми безсмертя [100].

Цю задачу можна сформулювати так. Нехай є машина, що здатна замінювати свої деталі в міру того, як вони починають виходити з ладу. Кожна деталь машини характеризується деяким середнім часом безвідмовної роботи. Тоді розв'язок проблеми безсмертя в постановці Е. Мура полягає в створенні машини, що мала б більший середній час безвідмовної роботи, ніж кожна з її деталей.

Відповідь виглядає так: якщо деяка машина складається не більше ніж з n деталей, і вірогідність безвідмовної роботи кожної деталі не перевищує деякої константи $k > 0$, тоді вірогідність одночасної відмови всіх деталей у певний момент часу дорівнює, в найгіршому разі, $1 - (1 - k)^n$. Тому прийде час, коли всі деталі відмовлять одночасно.

Машина Тьюринга є спеціальним типом автомата. Вона функціонально складається зі стрічки, розбитої на квадратні комірки, та пристрою читання-запису. Останній може читати символ з комірки і записувати один символ у комірку, рухаючись вздовж стрічки в різні сторони. Є також програма роботи — набір четвірок або п'ятірок, які визначають операції або обчислення машини як функції від множини початкових символів на стрічці (входів). У кінці роботи отримується вихідний набір, який є сукупністю символів, що залишились на стрічці після того, як були виконані всі операції, що задавались програмою.

Суттєвий факт, який вдалося довести за допомогою машини Тьюринга,— це те, що існує машина Тьюринга, яка може при визначених вхідних значеннях обчислити довільну, обчислювану за Тьюрингом, функцію. Така машина називається універсальною машиною Тьюринга.

Програму опису роботи машини Тьюринга M_1 можна розглядати як деяку послідовність символів. Немає жодних принципових перешкод для того, щоб розглядати цю програму як вхідні дані для деякої іншої машини M_y, що може моделювати машину M_1.

Універсальну машину Тьюринга можна неформально визначити як машину:

$$<S^{(y)}, Q^{(y)}, \Pi^{(y)}>,$$

що може сприймати програму П для обчислення будь-якої функції, яку, в принципі, можна обчислити за допомогою спеціалізованої машини $<S^{(*)}, Q^{(*)}, \Pi^{(*)}>$. У подальшому вона працює як машина $<S^{(*)}, Q^{(*)}, \Pi^{(*)}>$. Можна довести, що таку універсальну машину можливо побудувати. Якщо у нас є така машина, то згідно з тезою Тьюринга для неї можна написати програму обчислення будь-якої функції, обчислюваної в інтуїтивному розумінні, тобто запрограмувати будь-який інтуїтивний алгоритм.

13.2. Клас функцій, обчислюваних за Тьюрингом

Визначимо, яким вимогам повинна задовольняти функція для того, щоб її можна було обчислити за Тьюрингом. Дамо деякі попередні визначення [173, 179].

Базовими (вихідними) функціями будемо називати такі функції: *нуль-функція* — $Z(x) = 0$ за будь-якого x; *додавання одиниці* — $N(x) = x + 1$; *проектуючі функції* — $U_n^i(x_1, \ldots, x_n) = x_i$ при всіх $x_1, \ldots, x_n$.

Нові функції можна отримувати за допомогою таких правил:

1. **Підстановка** — якщо $f(x_1, \ldots, x_n) = g(h_1(x_1, \ldots, x_n), \ldots, h_m(x_1, \ldots, x_n))$, то кажуть, що функція сконструйована з функцій $g, h_1, \ldots, h_m$ за допомогою правила підстановки.

2. **Рекурсія** — розглядаються два випадки. Якщо $n \neq 0$ і $f(x_1, \ldots, x_n, 0) = g(x_1, \ldots, x_n)$; $f(x_1, \ldots, x_n, y+1) = h(x_1, \ldots, x_n, y, f(x_1, \ldots, x_n, y))$, то кажуть, що функція сконструйована за допомогою рекурсії.

При $n = 0$ визначення рекурсії таке: $f(0) = k$, де k — ціле невід'ємне число і $f(y+1) = h(y, f(y))$.

3. **μ-оператор**: нехай функція $g(x_1, \ldots, x_n, y)$ така, що за будь-яких $x_1, \ldots, x_n$ існує хоча б одне значення y, при яких $g(x_1, \ldots, x_n, y) = 0$. Позначимо через $f(x_1, \ldots, x_n) = \mu(g(x_1, \ldots, x_n, y) = 0)$ найменше значення y, за якого $g(x_1, \ldots, x_n) = 0$. Тоді говоритимемо, що f сконструйована за допомогою μ-оператора.

Функція f називається *примітивно рекурсивною*, якщо її можна сконструювати з базових функцій за допомогою скінченного числа застосувань підстановок і рекурсій.

Функція f називається *рекурсивною*, якщо її можна сконструювати з базових функцій за допомогою скінченного числа застосувань підстановок, рекурсій і μ-операторів.

Будь-яка примітивно рекурсивна функція є водночас і рекурсивною. Клас примітивно рекурсивних функцій дуже широкий. Навряд чи вам доводилося мати справу з функціями, що є рекурсивними, але не є примітивно рекурсивними. Проте такі функції існують.

Частково рекурсивною функцією називатимемо функцію, що визначена не при всіх значеннях аргумента. Відома теорема [173]:

Теорема 13.1. *Клас функцій, частково обчислюваних за Тьюрингом, збігається з класом частково рекурсивних функцій.*

Машини Тьюринга дозволяють обчислювати рекурсивні функції, і тільки їх. Але залишається відкритим ключове питання: чи не можна запропонувати інші формалізації поняття алгоритму і відповідні обчислювальні пристрої, що дозволяли б обчислювати нерекурсивні функції? А. Чорч висунув гіпотезу, яка дістала назву тези Чорча.

Теза Чорча: *клас частково рекурсивних функцій збігається з класом частково обчислюваних функцій.*

Теза Чорча еквівалентна тезі Тьюринга. Якщо ми визнаємо тезу Чорча, ми повинні визнати те, що, який би пристрій для здійснення обчислень ми не сконструювали, він не зможе робити більше, ніж машина Тьюринга. Він може здійснювати обчислення швидше, програмування для нього може бути легшим, але його обчислювальні можливості не вийдуть за рамки рекурсивних функцій.

Тезу Чорча, як і рівнозначну їй тезу Тьюринга, довести неможливо, оскільки до її формулювання входить інтуїтивне поняття "обчислювана функція", яке неможливо точно сформулювати.

Отже, ми дійшли висновку, що можемо дати точне визначення універсального обчислювального пристрою.

Універсальним обчислювальним пристроєм *називатимемо обчислювальний пристрій, на якому можна промоделювати роботу машини Тьюринга.*

Якщо є точне визначення поняття "алгоритм", ми можемо визначити, чи існує алгоритм для вирішення тієї чи іншої задачі.

13.3. Моделі РАМ і РАСП

Можна зазначити, що машина Тьюринга є дуже незручною для програмування через такі вади:

1. *Немає довільного доступу до пам'яті* — якщо, наприклад, машина вказує на комірку з номером k, а потрібно перейти до $k + n$, то машина повинна послідовно переглянути комірки $k + 1$, $k + 2$ і т. д.;

2. *Неструктурованість записів на стрічці* — заздалегідь невідомо, де закінчується одне число і починається інше; доводиться використовувати спеціальні розподільники.

3. *Дуже обмежений набір команд* — відсутні, наприклад, основні арифметичні операції.

У [15] наведені дві моделі обчислювальних машин, які більш наближені до практичної реалізації та які позбавлені зазначених вад. Це — РАМ і РАСП.

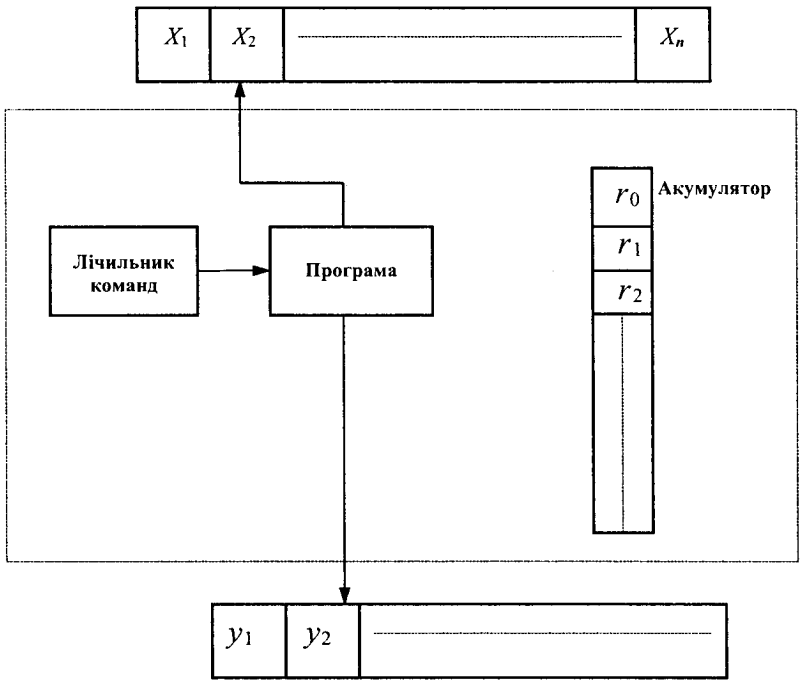


Рис. 13.3. Структура РАМ

РАМ є одноадресною машиною з довільним доступом до пам'яті. Вона складається з вхідної стрічки (з якої вона може тільки читати), вихідної стрічки (на яку вона може тільки записувати) і пам'яті. Вхідна і вихідна стрічки є послідовностями клітинок; у будь-якій з них може бути записане ціле число. Пам'ять є послідовністю регістрів; у будь-якому з них можна зберігати довільне ціле число. Структура РАМ наводиться на рис. 13.3.

Набір команд включає одноадресні команди основних арифметичних операцій, читання та запису, переходу до потрібної команди та зупинки. Регістр r_0 має спеціальне призначення і називається акумулятором. Дію акумулятора охарактеризуємо описом дії команди ADD *a*. Вона полягає у додаванні до акумулятора вмісту регістру з номером *a*; результат зберігається в акумуляторі.

Модель РАСП відрізняється від РАМ тим, що програма перебуває в регістрах пам'яті і може змінювати сама себе. Легко побачити аналогії між РАСП і типовим фон-нейманівським процесором.

13.4. Складність алгоритмів

Якщо ми маємо певну проблему, то завжди намагаємося знайти алгоритм її вирішення. Але одну й ту саму задачу можна розв'язувати за допомогою різних підходів, різних алгоритмів. Як вирішити питання, котрий з алгоритмів кращий, ефективніший? Для цього потрібен спеціальний математичний апарат аналізу алгоритмів, їх складності [15, 16, 142, 304]. Під ***складністю алгоритму*** розумітимемо часову оцінку роботи алгоритму або ж ємність необхідної пам'яті для виконання алгоритму за допомогою деякої абстрактної обчислювальної машини. Визначення цих характеристик називатимемо *апріорним аналізом алгоритму*, що ігнорує всі факти, котрі залежать від комп'ютера (обчислення проводяться на РАМ-машині), мови запису алгоритму (мови програмування), і концентрує свої зусилля на визначенні *типу* функції, яка характеризує або час виконання алгоритму, або розмір потрібної пам'яті для виконання алгоритму.

Проводячи повний аналіз часу обчислення або потрібної пам'яті, ми повинні обов'язково реалізовувати дві фази: апріорний аналіз і тестування. Тестування дозволяє за допомогою дійсно отриманих статистик про час виконання алгоритму або необхідну пам'ять на різних даних визначити найтиповіші оцінки, найкращі оцінки, найгірші оцінки для різних випадків застосування алгоритму.

Алгоритми можуть бути порівняні і за допомогою інших критеріїв. Але визначення порядку зростання часової оцінки залежно від зростання розміру вхідних даних є проблемою, вирішення якої, з точки зору комп'ютерного застосування, і в майбутньому буде найактуальнішим.

Із задачею будемо асоціювати *ціле* число, що характеризує розмір вхідних даних задачі. Наприклад, якщо ми розглядаємо задачу впорядкування, тоді таке ціле характеризує кількість елементів впорядкування. Якщо розглядаємо задачу, пов'язану з обробкою матриці (обчислити визначник матриці), тоді такою величиною буде розмірність матриці і т. п.

Час, необхідний алгоритму для вирішення проблеми, який є деякою функцією від розміру вхідних даних (розміру проблеми, задачі), *називають часовою складністю алгоритму*.

Гранична поведінка росту складності, залежно від розміру, називається *асимптотичною часовою складністю*. Аналогічно можна визначити складність алгоритму відносно пам'яті і асимптотичну складність алгоритму відносно пам'яті.

Можна було б думати, що для сучасних комп'ютерів з величезною пам'яттю і швидкістю такі характеристики алгоритмів неактуальні. Але це не так. Якщо часова оцінка алгоритму велика і розмір задачі досить великий, тоді такий алгоритм навіть на сучасних суперкомп'ютерах у реальний час не може бути виконаний. Тому визначення типу часової оцінки алгоритму є досить важливою проблемою.

Існує кілька спеціальних математичних позначень, які використовуються при аналізі алгоритму [15, 304]. Розглянемо запис $f(n) = O(g(n))$.

Він читається так: функція f має порядок O від g від n, тоді і тільки тоді, коли існують дві додатні константи c і n_0, такі що $|f(n)| \le c\,|g(n)|$ для всіх $n \ge n_0$.

Припустимо, ми визначили час обчислень $f(n)$ для деякого алгоритму: $f(n) = O(g(n))$. Коли ми говоримо, що алгоритм має час обчислень $O(g(n))$, то маємо на увазі те, що коли алгоритм буде оброблений комп'ютером на даних деякого типу, які мають розмір n, результуючий час буде завжди меншим за добуток деякої константи та $|g(n)|$.

Коли визначатимемо порядок $f(n)$, завжди будемо намагатися отримати найменше $g(n)$, таке що $f(n) = O(g(n))$.

Враховуючи той факт, що нас цікавить гранична поведінка часової складності алгоритму і те, що часова відмінність виконання базових операцій алгоритму (додавання, віднімання, множення, ділення, присвоєння) незначна, ми можемо взяти деяке одне, середнє, константне часове значення для характеристики виконання всіх базових операцій алгоритму.

Не важко довести теорему [304]:

Теорема 13.2. *Якщо* $A(n) = a_m n^m + \ldots + a_1 n + a_0$ *є поліномом ступеня* m, *тоді* $A(n) = O(n^m)$.

Теорема 13.2 наводить ідею знаходження часових характеристик для складних алгоритмів, що мають модульну структуру. Нехай алгоритм має k блоків, часові оцінки яких будуть відповідно мати вигляд $c_1 n^{m_1}, c_2 n^{m_2}, \ldots, c_k n^{m_k}$, тоді часова оцінка всього алгоритму буде $c_1 n^{m_1} + c_2 n^{m_2} + \ldots + c_k n^{m_k}$ і згідно з теоремою 13.2 є $O(n^m)$, де $m = \max\{m_i\}$, $i = 1, \ldots, k$.

Розглянемо випадок, коли два алгоритми розв'язують одну задачу, вхід якої має довжину n. У першого час обчислень — $O(n)$, у другого — $O(n^2)$. Постає запитання, якому з алгоритмів віддати перевагу? Очевидно, що для досить великих значень n час виконання другого алгоритму буде значно більшим від часу виконання першого алгоритму. Все залежить від значень константи c. Наприклад, якщо дійсний час виконання цих алгоритмів є відповідно $2n$ і n^2, тоді перший алгоритм буде швидшим за другий для всіх $n > 2$. З іншого боку, якщо час виконання алгоритмів є $10^4 n$ і n^2, тоді другий алгоритм буде швидшим для всіх $n < 10^4$.

Тому ми не можемо говорити про перевагу одного алгоритму над іншим, поки не дізнаємося про характеристику константи і довжину входу.

Значення константи c на практиці залежить від багатьох факторів. При апріорному аналізі дослідження значень константи c не проводять, оскільки для великих $n > n_0$ вони перестають бути суттєвими.

Найбільш типовими часовими оцінками алгоритмів є $O(1)$ — *константна*, $O(\log n)$ — *логарифмічна*, $O(n)$ — *лінійна*, $O(n^m)$ — *поліноміальна* і $O(2^n)$ — *експоненційна*. Для досить великих n справедливе відношення:

$$O(1) < O(\log n) < O(n) < O(n \log n) < O(n^2) < O(n^3) < O(2^n).$$

Експоненційні алгоритми для досить великих значень довжини входу n мають величезний реальний час обчислень. Відмітимо також, що часові

оцінки $O(n)$ і $O(\log n)$ значно менші за інші. Для великих обсягів даних алгоритми зі складністю, більшою за $O(n \log n)$, стають непрактичними. Алгоритми експоненціального типу можуть використовуватись на практиці тільки для дуже малих значень n і досить малого значення константи c.

Цікавим також є таке питання: чи існують коректно визначені математичні задачі, для яких не існує алгоритму розв'язку? А. Тьюрингом було доведено, що такі задачі існують [173]. Прикладом нерозв'язної задачі може служити проблема зупинки: для конкретної програми і вхідних даних визначити, чи зупиниться вона колись. Тьюринг довів, що не існує алгоритму, який би коректно вирішував усі часткові випадки цієї задачі.

Повернемось до визначення позначень.

$f(n) = \Omega(g(n))$ тоді і тільки тоді, коли існують додатні константи c і n_0, такі що $|f(n)| \geq c\,|g(n)|$ при $n > n_0$.

Існують випадки алгоритмів, для яких матимуть місце $f(n) = O(g(n))$ і $f(n) = \Omega(g(n))$.

Для цього ми використовуватимемо нотацію:

$f(n) = \Theta(g(n))$ тоді і тільки тоді, коли існують додатні константи c_1, c_2 і n_0, такі що для всіх $n > n_0$, $c_1\,|g(n)| \leq c_2\,|g(n)|$.

Якщо $f(n) = \Theta(g(n))$, тоді $g(n)$ є одночасно і нижньою, і верхньою межею $f(n)$. У цьому разі кращий і гірший випадок залежить від значень констант, має часову оцінку, що лежить у межах від $\Omega(1)$ до $O(n)$.

Дамо формальну математичну нотацію:

$f(n) \sim O(g(n))$ (читається *"f від n є асимптотою до $g(n)$"*) тоді і тільки тоді, коли $\lim\limits_{n\to\infty} \dfrac{f(n)}{g(n)} \to 1$.

Уданому разі $f(n)$ і $g(n)$ відрізняються тільки константним фактором. Наприклад, якщо $f(n) = a_k n^k + a_{k-1} n^{k-1} + \ldots + a_0$, тоді $f(n) = O(n^k)$ і $f(n) \sim O(a_k n^k)$.

Виокремимо найвживаніші суми оцінок. Для більшості алгоритмів сумарні оцінки мають вигляд $\sum\limits_{g(n)\leq i\leq h(n)} f(i)$. У найпростіших випадках вони набувають вигляду поліномів Бернуллі:

$$\sum_{i=1}^{n} 1, \sum_{i=1}^{n} i, \sum_{i=1}^{n} i^2 \ldots$$

Зробимо оцінки цих сум:

$$\sum_{i=1}^{n} i = n(n+1)/2 = \Theta(n^2)$$

$$\sum_{i=1}^{n} i^2 = n(n+1)(2n+1)/6 = \Theta(n^3)$$

і в загальному випадку

$$\sum_{i=1}^{n} i^k = \frac{n^{k+1}}{k+1} + \frac{n^k}{2} + \text{lower order terms}$$

Тому ми можемо дійти висновку, що $\sum_{i=1}^{k} i^k = \Theta(n^{k+1})$,

або точніше: $\sum_{i=1}^{k} i^k \sim O\left(\frac{n^{k+1}}{k+1}\right)$.

Наведемо кілька прикладів побудови часових оцінок.

Розглянемо алгоритм, який знаходить максимальний елемент з n елементів масиву цілих чисел і описаний програмою 1 (хто не зможе прочитати нотацію паскалевської програми, подальші роздуми може пропустити).

Програма 1. **Знаходження максимального елемента послідовності чисел.**

```
program EX13_1;
const n = 100;
var max, i, m: integer;
A: array [1..n] of integer;
begin
    readln (m);
    if m ≥ 1 then
        begin
            readln (A [1]);
            max := A [1];
            for i:= 2 to m do
                begin
                    readln (A [i]);
                    if A [i] > max then max := A [i];
                end;
            writeln ('Максимальний елемент = ', max);
        end;
    else
    writeln ('Пуста послідовність');
end;
```

Часова оцінка цього алгоритму є одночасно і $O(m)$, і $\Omega(m)$, оскільки її визначає цикл for, який завжди потребує $m - 1$ ітерацій.

Розглянемо ще один алгоритм — алгоритм пошуку search елемента x у масиві A. Якщо знайдеться i, таке що $A[i] = x$, тоді перше таке i є результатом пошуку, в іншому разі результатом буде значення 0.

Програма 2. Ще один варіант знаходження максимального елемента

```
program EX13_2;
const n = 1000;
type int = 1..n;
ar: array [int] of real;
var x: real;
j, k, l: int;
a: ar;
```

```
function search (a:ar; m:int; x:real):int;
begin
    i := 0; j := 1;
    while ((j <= m) or (i <> 0)) do
    if a [j] = x then i := j else j := j + 1;
    search := i;
end;
begin
    readln (x, k);
    for j := 1 to k read (a [j]);
    l := search (a, k, x);
    if l > 0 then writeln ('x дорівнює елементу масиву з індексом i = ', l)
    else writeln ('x не дорівнює жодному елементу масиву, i = 0')
end;
```

Алгоритм пошуку, реалізований функцією **search**, має часову оцінку, яка лежить у межах від $\Omega(1)$ до $O(k)$. У цьому алгоритмі потрібне значення можна знайти за найліпших умов при першому ж порівнянні, а за найгірших — переглянувши всі елементи масиву.

13.5. Задачі класу P і NP

Терміни *задачі класу P і NP* з'явились на початку 70-х років [15, 97, 148, 304]. Вони стали символом великих труднощів, з якими борються розробники алгоритмів вирішення задач постійно зростаючого розміру. Назвемо *P*-задачею задачу, що може бути розв'язана на детермінованій машині Тьюринга за поліноміальний час $O(n^m)$, де n — розмір задачі. Інакше кажучи, для розв'язку *P*-задач існує поліноміальний алгоритм.

Під *детермінованою* машиною Тьюринга розумітимемо машину, описану в п. 13.1. Детермінованість означає, що за будь-якої ситуації дія машини є чітко визначеною. На відміну від цього, недетермінованою машиною Тьюринга називається машина, команди якої мають вигляд $S_k q_i \to S'Q'G$. Це означає, що, якщо машина перебуває в стані q_i і читає символ S_k, вона може записати будь-який символ з множини S', перейти до будь-якого стану з множини Q' і зсунутися ліворуч, праворуч або залишитися на місці. Можна сказати, що на кожному кроці породжується певна кількість нових машин, кожна з яких виконує ту чи іншу конкретну дію. Тепер можна дати визначення *NP*-задачі. *NP*-задачею називається задача, що може бути виконана за поліноміальний час на недетермінованій машині Тьюринга.

Чимало задач математики, теоретичного програмування, дискретної математики, дослідження операцій належать до класу *NP*-повних і список таких задач постійно зростає. Тому всім розробникам алгоритмів бажано розуміти сутність і значення цього поняття.

Для науковця характерним є прагнення розробляти ефективні алгоритми вирішення якомога ширших класів задач. Історичний досвід розвитку

математики, практика розв'язку багатьох проблем показали, що загальність методу та його ефективність перебувають у постійному антагонізмі: виграємо в одному — програємо в іншому. Але при вирішенні конкретних задач із застосуванням ЕОМ важливо знати, чи можна в ідеалі очікувати на створення достатньо загальних та ефективних методів, чи потрібно свідомо йти по шляху розбиття задач на підзадачі і, користуючись їх специфікою, розробляти для них ефективні алгоритми, що будуть давати розв'язок підзадачі в реальний час.

Ці проблеми зумовили потребу в аналізі складності задач, проблематика якого актуальна, оскільки такі задачі постійно виникають на практиці та їх вирішення на ЕОМ пов'язане з величезними затратами машинного часу і пам'яті.

Більшість дискретних і комбінаторних задач можна вирішити за допомогою перебору варіантів. Наприклад, задача про рюкзак (п. 12.2) типу $\sum_{i=1}^{n} a_i x_i = b,\ x_i = 0, 1$ вирішується перебором варіантів булевих n-мірних векторів. Такі задачі називаються перебірними. Число кроків перебору росте експоненційно залежно від розміру задачі. Для деяких задач з цього класу вдається побудувати алгоритми, що виключають повний перебір варіантів (знаходження паросполук у графі). Але число таких задач незначне.

Усе це зумовило постановку центральної теоретико-методологічної проблеми всієї дискретної математики — *чи можна виключити перебір при розв'язку дискретних задач?* Дана проблема має також велике пізнавальне значення. При пошуку ефективних точних методів розв'язку широкого класу дискретних задач потрібно враховувати можливість *відсутності подібних методів* і, відповідно, вміти вчасно обмежувати своє прагнення, визнавши, що існують *складновирішувані задачі*. Ця проблема залишається відкритою і зараз [15].

Феномен складновирішуваних задач відомий давно. Відразу ж після уточнення поняття алгоритму було виявлено значну кількість алгоритмічно невирішуваних проблем. Для доведення нерозв'язності коректно визначених математичних задач використовують апарат машин Тьюринга і правило звідності. Типовим прикладом нерозв'язної задачі може бути згадувана нами *проблема зупинки*. Як ми зазначали, Тьюринг довів, що не існує алгоритму, який би коректно вирішував усі часткові випадки цієї задачі. Можна знайти деякі евристичні способи знаходження окремих нескінченних циклів за програмою і вхідними даними, але ніколи всіх можливих варіантів ми не врахуємо. Можна було б запустити програму на якомусь обчислювальному пристрої і чекати поки вона не зупиниться. У разі, коли вона досягала при зупинці оператора закінчення програми, ми могли б дати відповідь на поставлене питання про зупинку. Але така схема не буде алгоритмом, бо немає гарантії, що сама вона зупиниться.

У більшості перебірних задач є скінченна множина варіантів, серед яких потрібно знайти розв'язок. Але зі збільшенням розміру входу задачі

число варіантів швидко зростає і задача стає складновирішуваною, інакше кажучи — практично нерозв'язною. Тому у випадку скінченної множини варіантів аналогом алгоритмічної нерозв'язності є перебір експоненціального числа варіантів, а аналогом розв'язності — існування алгоритму поліноміального типу.

Для визначення типу задачі вживають поняття звідності задач з математичної логіки. Задача $П_1$ зводиться до задачі $П_2$, якщо метод розв'язку задачі $П_2$ можна трансформувати в метод розв'язку задачі $П_1$. Звідність називається поліноміальною, якщо подібне перетворення можна зробити за поліноміальний час. Тому можна виокремити два основних класи задач: клас *NP* усіх перебірних задач і клас *P* перебірних задач, розв'язок яких можна отримати за поліноміальний час на машині Тьюринга. Зрозуміло, що $P \subseteq NP$. Головною проблемою теорії звідності є питання, чи збігаються класи *P* і *NP*. Так математично формулюється проблема елімінації (уникнення) перебору.

Як уже зазначалося, у класі *NP* є *NP*-повні (універсальні) задачі, якими називаються такі, що до них поліноміально зводиться довільна задача з класу *NP*. Вони використовуються як еталони складності.

Якщо б вдалось довести, що якась *NP*-повна задача належить до класу *P*, тоді було б доведено, що $P = NP$, і можна було б чекати на побудову ефективних алгоритмів для різних класів дискретних задач. Якщо ж класи *P* і *NP* відмінні, тоді потрібно розробляти ефективні алгоритми для вужчих класів задач. Більшість сучасних математиків вважають, що гіпотезу $P = NP$ неможливо ні довести, ні спростувати. Недоведеність відмінності між класами *P* і *NP* можна пояснити так. Ці класи визначаються за допомогою часу роботи обчислювального пристрою з потенційно необмеженою пам'яттю. Але добре відомо, що час виконання алгоритму на машині погано піддається опису та аналізу загальновідомими математичними засобами. Сьогодні питання, чи дійсно $P \subset NP$, залишається відкритим.

Важливим є також поняття *NP*-складної задачі. Такою називається задача, якщо до неї за поліноміальний час може бути зведена будь-яка *NP*-задача. Якщо *NP*-складна задача є *NP*-задачею, вона називається *NP*-повною. Сутність таких задач характеризує теорема Кука. Вона стверджує, що будь-яка *NP*-задача поліноміально зводиться до задачі про виконуваність булевого виразу. Нагадаємо, що ця задача формулюється так: для заданого булевого виразу знайти хоча б один набір значень змінних, який перетворює цей вираз на 1.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте автоматний спосіб задання алгоритмів.
2. Чим відрізняється комбінаційна схема від скінченного автомата?
3. Охарактеризуйте табличний і графічний способи задання автоматів.
4. Що таке машина Тьюринга?

5. Функції якого класу можуть бути обчислені за допомогою машини Тьюринга?
6. Охарактеризуйте поняття часової складності.
7. Наведіть відому вам класифікацію алгоритмів за часовою складністю.
8. Дайте визначення *P*- та *NP*-задач.
9. Що таке *NP*-повна задача? Наведіть приклад.

Розділ 14
ПЛАНУВАННЯ В ПРОСТОРІ СТАНІВ

То берег левый нужен им,
То берег правый...
З пісні

14.1. Основні поняття теорії графів

Як уже зазначалося в п. 12.7, прийняття рішень на основі планування в просторі станів передбачає побудову формалізованої моделі предметної області у вигляді ***графу станів***, вершини якого відповідають можливим ситуаціям (станам), а дуги — операціям, які дозволяють переходити від одного стану до іншого. Також очевидно, що вирішення багатьох інтелектуальних задач пов'язане з діями над графами: орієнтованими і неорієнтованими, циклічними і ациклічними, навантаженими і ненавантаженими. Часто такі маніпуляції вимагають знаходження вершини (вузла) або множини вершин, ребра або множини ребер, які задовольняють якійсь певній умові. Наприклад, такими умовами можуть бути: знайти множину ребер графу, значення ваг яких менше за задану величину; знайти остове дерево графу; знайти остове дерево мінімальної вартості; знайти компоненти двозв'язності графу і т. п.

Визначити такі множини можна лише шляхом систематичної перевірки всіх вершин або ребер заданого графу. Фактично задача зводиться до пошуку в графі. Тому важливою компонентою багатьох алгоритмів на графах є алгоритм пошуку обходу графу. Для будь-яких операцій з графом потрібно початкове його задання в пам'яті ЕОМ. Основними способами задання графу є: матриця інцидентності, матриця суміжності та списки суміжності.

Однією зі структур даних, що найчастіше використовується в програмуванні, є лінійні списки. Вони можуть бути задані за допомогою методів послідовного або зв'язного зберігання. При виборі способу зберігання в конкретному випадку слід враховувати, які операції і в якій кількості виконуватимуться над лінійними списками.

Лінійний список *— це скінченна послідовність однотипних елементів. Кількість елементів послідовності називають довжиною списку.*

Списки залежно від організації зв'язку між елементами бувають однозв'язними (лінійними), двозв'язними (лінійними), двозв'язними (циклічними) і т. п.

Лінійний список L, що складається з елементів $l_1, l_2, ..., l_n$ записуватимемо у вигляді $L = <l_1, l_2, ..., l_n>$ і зображатимемо графічно, як показано на рис. 14.1.

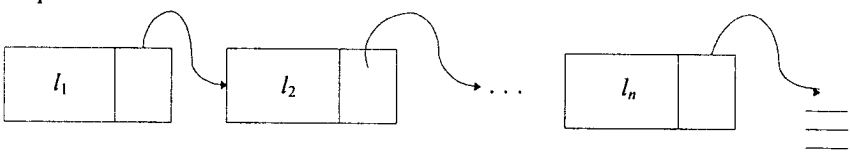


Рис. 14.1. Графічне зображення списку

Вперше поняття "список" було введено в роботі А. Ньюела, Г. Саймона, Дж. Шоу при розробці програми "Логік—Теоретик" і знайшло перше впровадження при реалізації операцій з пам'яттю в ЕОМ. При роботі зі списками найчастіше використовуються такі операції: знайти елемент в списку із заданою характеристикою, вилучити певний елемент зі списку, внести елемент у спеціально визначене місце списку, виконати певні дії над елементами списку. В більшості мов програмування, за винятком мов, подібних до Ліспу або Прологу, не існує стандартних механізмів для задання лінійних списків.

Можна виокремити два основні методи задання лінійних списків: послідовне і зв'язне зберігання. Перше в більшості випадків реалізується за допомогою масивів, а друге — за допомогою динамічних змінних. При виборі засобів зберігання списку слід враховувати, які операції і з якою послідовністю виконуватимуться над ними. При використанні списку як базової структури для побудови складніших структур даних, а також для економії пам'яті, використовується зв'язне зберігання лінійних списків з використанням динамічних змінних. Якщо ми організовуємо видалення та вставку елементів у список спеціальним чином, то отримуємо загальноприйняті в програмуванні структури даних типу стек, черга і т. п.

***Стеком** називається спеціально організований список, в якому занесення і видалення елементів можна проводити тільки через один початковий елемент, який називають вершиною стека* [15, 51, 248, 304].

Стек працює за принципом "останній зайшов — перший вийшов" (Last In First Out). Тому стек ще називають списком типу LIFO.

***Чергою** називають список, в якому всі вставки здійснюються в кінець списку (позначатимемо змінною rear), а всі видалення відбуваються з "голови" (початку) списку (позначатимемо змінною front).*

Ці операції виконуються за принципом "перший зайшов — перший вийшов" (First In First Out), тому чергу ще називають списком типу FIFO.

При вирішенні конкретних задач можуть виникати й інші різновиди зв'язного зберігання [15, 51, 248, 304].

Поняття ***граф*** вперше ввів до розгляду знаменитий математик Ейлер в 1736 р. [15, 304]. Граф G містить дві множини $G = (V, E)$, де V — множина вершин, E — множина ребер. V є скінченною непустою множиною вершин (іноді їх називають вузлами), традиційно пронумерованою 1, 2, ..., n. E — скінченна множина пар вершин. Кожна пара із E є ребром у G. Коли пари впорядковані, тоді графи називаються ***орієнтованими***, в іншому випадку — ***неорієнтованими***. Впорядковане ребро позначатимемо $< i, j >$, а невпорядковане — (i, j).

У більшості випадків застосувань з множиною ребер графу пов'язують множину дійсних чисел, які називають вагами. Такий граф називають ***зваженим***.

У неорієнтованому графі говоримо, що вершина i суміжна вершині j, коли існує ребро (i, j). Ступенем вершини назвемо число суміжних їй вершин. В орієнтованому графі виділяють півстепінь входу (число вхідних) та півстепінь виходу (число вихідних ребер з вершини).

Шляхом з вершини v_p до вершини v_q є послідовність вершин v_p, v_{i1}, v_{i2}, ..., v_{in}, v_q, така що (v_p, v_{i1}), (v_{i1}, v_{i2}), ..., (v_{in}, v_q) є ребрами в E (G). Довжиною шляху називають число ребер, які належать шляху. *Простим шляхом* називають шлях, в якому всі вершини, за винятком першої та останньої, є різними.

Циклом називають простий шлях, в якому перша та остання вершини збігаються. На рис. 14.2 маємо орієнтований і неорієнтований графи з 6 вершинами і 6 ребрами.

Неорієнтований граф називають *зв'язним*, коли для довільної пари вершин існує шлях між ними.

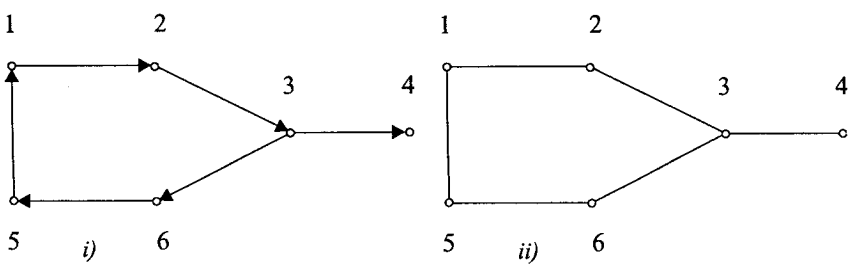


Рис. 14.2. Зображення графу: *і)* орієнтованого; *іі)* неорієнтованого

14.2. Способи задання графів

Існує два основних підходи до задання графів: послідовний і зв'язний.

Послідовна форма використовує квадратну таблицю, яку називають матрицею суміжності. Для неорієнтованого графу матриця суміжності Graf (1:n, 1:n) визначається так: Graf $(i, j) = 1$, якщо $(i, j) \in E$ і Graf $(i, j) = 0$ — в іншому випадку. У разі навантаженого графу Graf $(i, j) =$ *вартість*

ребра (i, j), якщо ребро існує, та Graf $(i, j) = +\infty$ у разі відсутності ребра (i, j) в $G = (V, E)$. Аналогічно задається й орієнтований граф, але з урахуванням орієнтації ребер.

На рис. 14.3 продемонстровано задання матрицями суміжності неорієнтованого і орієнтованого графів з рис. 14.2.

	1	2	3	4	5	6
1	0	1	0	0	1	0
2	1	0	1	0	0	0
3	0	1	0	1	0	1
4	0	0	1	0	0	0
5	1	0	0	0	0	1
6	0	0	1	0	1	0

	1	2	3	4	5	6
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	1	0	1
4	0	0	0	0	0	0
5	1	0	0	0	0	0
6	0	0	0	0	1	0

Рис. 14.3. Матриці суміжності графів

За такого задання нам потрібно зробити $O(n^2)$ операцій по заповненню таблиці. Тому всі алгоритми, які використовують послідовну форму задання графу, не можуть мати часових оцінок, кращих за $O(n^2)$.

Введемо позначення: якщо множина V складається з n вершин, тоді $|V| = n$, аналогічно $|E| = m$. У теорії графів для задання графу використовують *матрицю інцидентності*. Вона має розмірність $n \times m$, де $|V| = n$, $|E| = m$. Для орієнтованого графу стовпець, що відповідає дузі $\langle u, v \rangle \in E$, містить 1 в рядку, що відповідає вершині u, та 0 в рядку v і нулі в інших рядках. У разі неорієнтованого графу стовпець, що відповідає ребру (u, v), містить 1 в рядках, що відповідають u та v, і нулі в інших рядках. З погляду програмування такий спосіб задання графу потребує $n \times m$ комірок пам'яті, він незручний для організації доступу до інформації. Наприклад, для побудови відповіді на запитання типу *"існує дуга (u, v)?"* в найгіршому випадку потрібно переглянути всі m стовпців.

Тому існує інша форма задання графу, яку називають списком зв'язності. Граф G з n вершинами буде задаватися n списками — по одному для кожної вершини i. Список для вершини i містить вершини, суміжні з i. На рис. 14.4 показані списки суміжності для графів з рис. 14.2.

Списки суміжності можуть бути задані масивом VERTEX $(1{:}p)$, де $p = e$, якщо граф орієнтований, і $p = 2e$ в неорієнтованому графі, а $e = |E|$. Head (i), $1 \le i \le n$ дає початкові вершини околу суміжності для вершини i. Якщо ми визначимо Head $(n + 1) = p + 1$, тоді вершини списку суміжності для вершини i запам'ятовуються в VERTEX (j), де Head $(i) \le j <$ Head $(i + 1)$. Якщо список суміжності для вершини i пустий, тоді Head (i) = Head $(i + 1)$. На рис. 14.5 продемонстровано послідовне задання списків суміжностей з рис. 14.4.

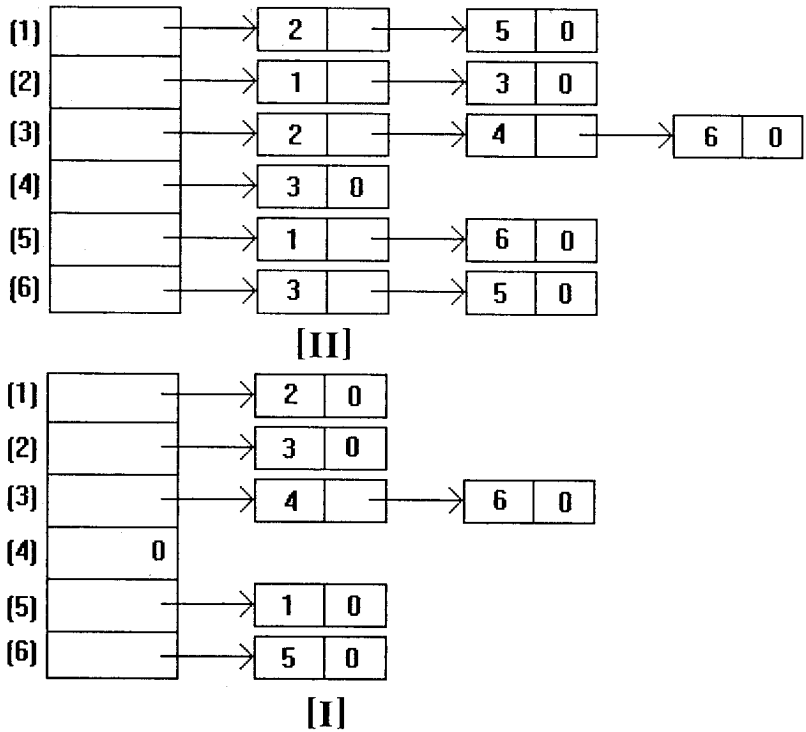


Рис. 14.4. Списки суміжності

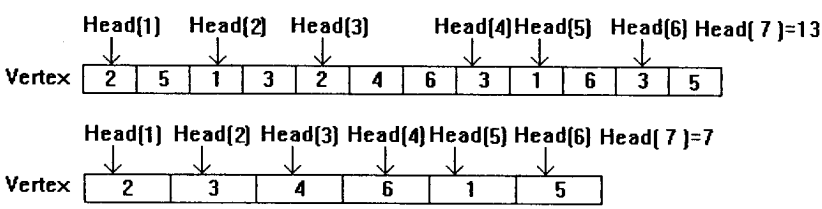


Рис. 14.5. Послідовне задання списків суміжностей із рис. 14.4

Зазначимо, що в класичному списку суміжності для неорієнтованих графів кожне ребро (u, v) задається подвійно: через вершину u в списку **початок** $[u]$ і через вершину v в списку **початок** $[v]$. Для багатьох алгоритмів на графах характерною є динамічна модифікація ребер, їх видалення і додавання. Тому в цих випадках у вузлах списків суміжності потрібно мати додаткові поля, що забезпечуватимуть ефективність реалізації наведених операцій. Найбільш використовуваними є такі поля: покажчик на попередній елемент, вузол **початок** $[v]$, що має серед своїх елементів вершину u, покажчик на вузол **початок** $[u]$.

14.3. Дерева

Одним з найвживаніших спеціальних типів графів є дерева. Орієнтований граф без циклів називається орієнтованим ациклічним графом.

***Дерево** — це орієнтований ациклічний граф, для якого виконуються такі умови:*

1) у графі знайдеться одна вершина, в яку не входить жодне ребро. Ця вершина називається коренем дерева;

2) у будь-яку вершину, крім кореня, входить тільки одне ребро;

3) з кореня можна знайти унікальний шлях до кожної вершини дерева.

Орієнтований граф, що складається з кількох дерев, називається *лісом*.

Нехай $F = (V, E)$ — граф, що є лісом. Якщо $(v, u) \in E$, тоді v називають "батьком" вузла u, а u — "сином" вузла v. Якщо є шлях з v в u, тоді v називають "предком" вузла u, а u — "нащадком" вузла v. Вузол v і його нащадки разом утворюють піддерево лісу F, і вузол v називають коренем цього піддерева.

Глибина вузла v в дереві — це довжина шляху із кореня в v. *Висота вузла v в дереві* — це довжина найдовшого шляху із v в деякий вузол. *Висотою дерева* називають висоту його кореня. *Рівень вузла v* в дереві дорівнює різниці висоти дерева і глибини вузла v.

Впорядкованим деревом називають те, в якому множина синів кожного вузла впорядкована. При зображенні впорядкованих дерев вважають, що множина синів кожного вузла впорядкована зліва направо.

Двійковим (бінарним) деревом називається таке впорядковане дерево, для якого виконуються умови:

1) кожний син довільного вузла ідентифікується або як *лівий син*, або як *правий син*;

2) кожний вузол має *не більше одного* лівого сина і *не більше одного* правого сина.

Піддерево T_l, коренем якого є лівий син вузла v (якщо таке існує), називається *лівим піддеревом вузла v*. Аналогічно визначається *праве піддерево вузла* T_r. Відмітимо, що всі вузли T_l знаходяться ліворуч усіх вузлів у T_r.

Максимальне число вузлів на рівні i бінарного дерева дорівнює $2^i - 1$. Також максимальне число вузлів у бінарному дереві глибини k дорівнює $2^k - 1$, де $k > 0$.

Для багатьох задач, пов'язаних з використанням дерев, виникає потреба переглянути всі вузли дерева в певному порядку. Розглянемо основні принципи проходження дерев на прикладі бінарних дерев. Алгоритми обходу мають рекурсивний характер:

I) пряме проходження:

1. відвідати корінь;
2. відвідати згідно з прямим проходженням ліве піддерево кореня;
3. відвідати згідно з прямим проходженням праве піддерево кореня;

II) обернене проходження:

1. відвідати згідно з оберненим проходженням ліве піддерево кореня;

2. відвідати корінь;

3. відвідати згідно з оберненим проходженням праве піддерево кореня;

III) кінцеве проходження:

1. відвідати згідно з кінцевим проходженням праве піддерево кореня;

2. відвідати згідно з кінцевим проходженням ліве піддерево кореня;

3. відвідати корінь.

Візьмемо до розгляду дерево, зображене на рис. 14.6.

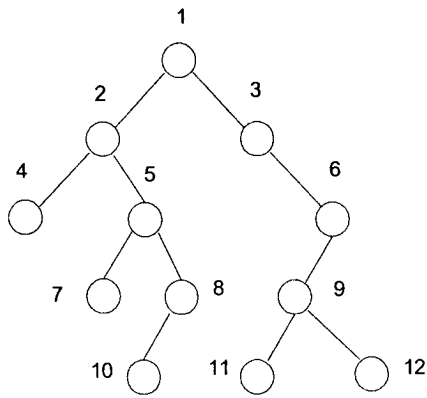


Рис. 14.6. Бінарне дерево

Тоді перегляд вузлів у прямому напрямку матиме таку послідовність: 1, 2, 4, 5, 7, 8, 10, 3, 6, 9, 11, 12; в оберненому проходженні: 4, 2, 7, 5, 10, 8, 1, 3, 11, 9, 12, 6; в кінцевому порядку: 12, 11, 9, 6, 3, 10, 8, 7, 5, 4, 2, 1.

Структура даних дерева найчастіше застосовується для реалізації різних довідників. Тому над деревами доводиться виконувати такі операції, як *"знайти вузол із певною властивістю"*, *"визначити батька або синів заданого вузла"*, *"вилучити вказаний вузол або частину дерева"*, *"додати новий вузол"* тощо.

14.4. Способи зберігання дерев

Для дерев прийняте послідовне і зв'язане зберігання [15, 51, 248, 304]. ***Послідовне зберігання*** створюється на основі одного з лінійних зображень дерева — у вигляді "рядка", зокрема воно може бути рівневим або дужковим.

Рівневе задання. З кожним вузлом дерева v зв'язується номер рівня цього дерева k — ціле додатнє число. Потім записуються всі його вузли з номерами рівнів відповідно до якогось порядку проходження дерева. Так, рівневим заданням дерева, зображеного на рис. 14.6, згідно з прямим порядком може бути 1, 1, 2, 2, 4, 3, 5, 3, 7, 4, 8, 4, 10, 5, 3, 2, 6, 3, 9, 4, 11, 5, 12, 5.

Дужкове задання. Дужковим заданням ($g3$) дерева B з одного вузла є запис цього вузла. Якщо B складається з кореня W і піддерев $B_1, B_2, \ldots, B_m$, тоді дужкове задання записується у вигляді $W\,(g_3\,(B_1), g_3\,(B_2), \ldots, g_3\,(B_m))$.

Дерево з рис. 14.6 у дужковому зображенні матиме вигляд:
1 (2 (4, 5 (7, 8 (10))), 3 (6 (9 (11, 12)))).

Форми зв'язного зберігання базуються на використанні динамічних змінних. Вони залежать від опису відношень між *предками* та *нащадками* (батьками та синами).

Найбільш використовуваними формами зв'язного зберігання є стандартна, обернена і розширена стандартна. У стандартній формі фіксуються зв'язки від батька до всіх синів. Отже, якщо дерево має степінь *n*, тоді в кожному вузлі ми повинні мати *n* вказівників на синів вузла. Якщо вузол має менше *n* синів, тоді непотрібні вказівники зв'язку обнуляються за допомогою вмонтованої функції nil. В оберненій формі зв'язного зберігання дерев вказуються зв'язки від синів до батьків. Розширена стандартна форма об'єднує стандартну та обернену форму зв'язного зберігання дерев. Для дерева з рис. 14.6 зв'язне зберігання в стандартній формі подане на рис. 14.7.

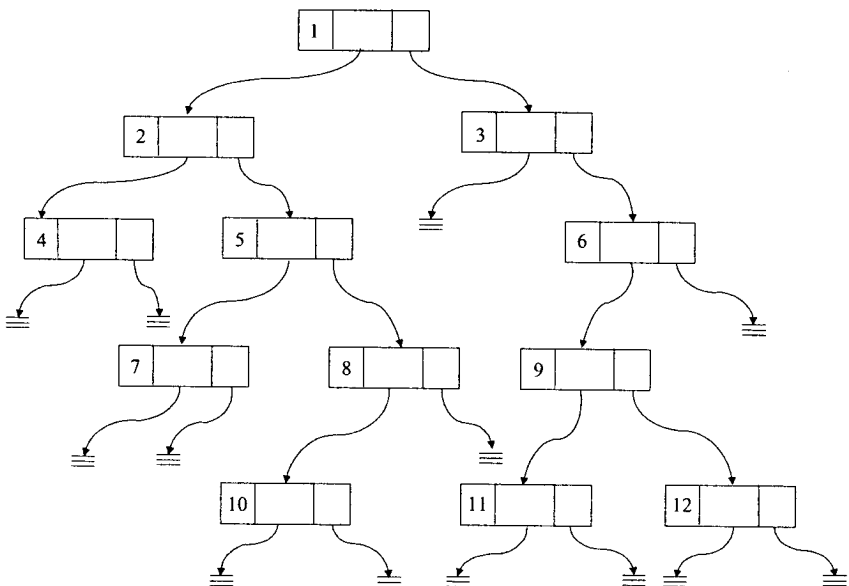


Рис. 14.7. Зберігання дерева у стандартній формі

В описаних формах задання дерева один вузол задання відповідає одному вузлу дерева, і кількість зв'язків вузла залежить від степеня дерева. Коли дерево неоднорідне і має багато вузлів, степінь яких значно менший від степеня дерева, тоді обробка незадіяних зв'язків спричиняє зайвий клопіт. Тому розглянемо задання вузла з фіксованим розміром.

Вузол матиме три поля — TAG, DATA і LINK. Коли TAG = 1, DATA містить покажчик на список, елементи якого відповідають синам відповідного вузла; в іншому разі DATA включає елемент-листок. Поле LINK ви-

користовується як поле зв'язку списку піддерев кожного елемента дерева. Так, дерево, зображене на рис. 14.8, відображатиметься в даному разі списковою структурою з рис. 14.9.

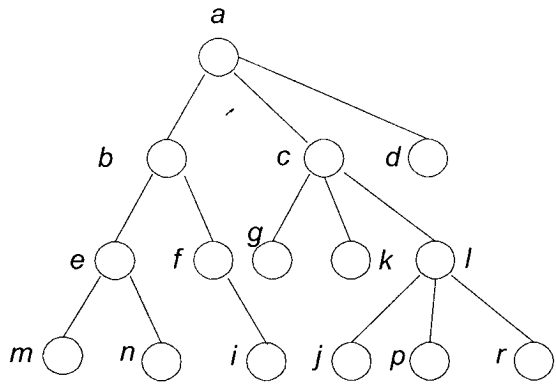


Рис. 14.8. Дерево

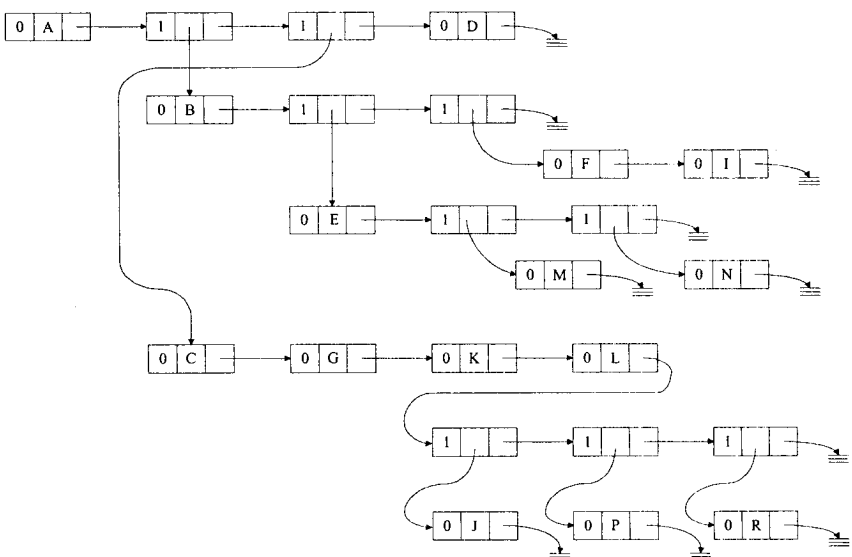


Рис. 14.9. Списокова структура задання дерева з рис. 14.8 у формі вузла фіксованого розміру

Бінарне дерево називається повним, якщо для деякого цілого числа k кожний вузол глибини, меншої за k, має як лівого, так і правого сина і кожний вузол глибини k є листом. Повне бінарне дерево висоти k має в точності 2^{k-1} вузлів.

У такому разі повне бінарне дерево з n вузлами можна зручно задати послідовною нумерацією вершин, починаючи з кореня, по рівнях зліва

направо за допомогою масиву Tree (i). Тоді для будь-якого вузла з індексом i, $1 \le i \le n$ справджуватиметься:

1) батько вузла i, Parent (i) визначатиметься $\lfloor i/2 \rfloor$, якщо $i \neq 1$. Коли $i = 1$, тоді цей вузол вже є коренем і тому батька він не має;

2) лівий син вузла i, Lson (i) визначатиметься формулою Lson (i) = $2 \times i$, коли $2 \times i \le n$. Коли ж $2 \times i > n$, тоді вузол i не має лівого сина;

3) правий син вузла i, Rson (i) визначатиметься формулою Rson (i) = $= 2 \times i + 1$, коли $2 \times i + 1 \le n$. Коли ж $2 \times i + 1 > n$, тоді вузол i не має правого сина.

14.5. Пошук у глибину і ширину

Розглянемо метод пошуку в неорієнтованому графі G, який дістав назву ***пошук у глибину***. Як уже зазначалося, загальна ідея методу полягає в такому. Процес перевірки деякої умови в вершині графу асоціюватимемо з процесом відвідування вершини. Відвідаємо вершину v_1. Потім вибираємо вершину u навколо вершини v_1 (суміжну з v_1), яка не була ще відвідана, і повторюємо процес знову. Нехай ми відвідали вершину v. Якщо існує нова, ще не відвідана суміжна вершина u з оточення v, тоді відвідуємо її і відмічаємо, що вона перестала бути новою. Процес пошуку далі продовжується з вершини u. Якщо ж для v не існує жодної нової суміжної вершини, тоді відмічаємо, що вершина v вже використана, повертаємось у вершину, з якої ми потрапили у v, і продовжуємо процес пошуку. Якщо вершина v була вершиною v_1, то процес пошуку закінчується.

Оцінимо часову оцінку запропонованого алгоритму. Для визначеності вважатимемо, що час, потрібний на відвідування вершини, має константний характер і граф заданий списком суміжності. Враховуючи, що для вибору чергової вершини відвідування потрібно переглянути всі суміжні вершини попередньої вершини відвідування (максимальна кількість суміжних елементів визначається m), а кількість усіх вершин у графі n, часова складність становитиме $O(n + m)$.

Зрозуміло, що у разі задання графу матрицею суміжності часова складність визначатиметься $O(n^2)$.

Пошук у ширину можна отримати з алгоритму пошуку в глибину заміною стека чергою. Після відвідування вершини проглядаються всі її суміжні вершини.

Розглянемо застосування алгоритмів пошуку в задачах з графами.

14.6. Остові дерева

Для багатьох застосувань корисним є поняття остових дерев.

Розглянемо зв'язний неорієнтований граф $G = <V, E>$. Довільне дерево $<V, T>$, де $T \subseteq E$ називається остовим деревом графу G. Зафіксуємо остове дерево $<V, T>$. Тоді ребра такого дерева називають гілками, а інші ребра графу — хордами.

Методом математичної індукції відносно u можна довести, що довільне дерево з n вершинами має $n - 1$ ребро. Процедури пошуку в глибину і ширину можуть використовуватись для побудови остових дерев з такою модифікацією: при досягненні нової вершини u із v проходить включення ребра $\{v, u\}$ в множину ребер T. Крім того, у разі застосування процедури пошуку в ширину, за умови, що ребрам графу приписані довжини, що дорівнюють 1, можна довести, що шлях у $<V, T>$ із довільної вершини v до кореня дерева r є найкоротшим шляхом з вершини v у вершину r у графі G.

Іншим цікавим поняттям є фундаментальна множина циклів графу.

Розглянемо граф $G = <V, E>$ та його остове дерево $<V, T>$. Якщо до остового дерева додати хорду $e \in E \setminus T$, тоді отримаємо рівно один цикл у новоутвореному графі. Позначимо його через C_e. Множину $b = \{C_e: e \in E \setminus T\}$ називають фундаментальною множиною циклів графу G відносно остового дерева $<V, T>$. Це означає, що кожний цикл графу G можна деяким природним чином отримати з множини b.

Симетричною різницею множин A і B (позначаємо $A \oplus B$) називають множину $A \oplus B = (A \cup B) \setminus (A \cap B)$.

Можна довести [165], що симетрична різниця множин $A_1, A_2, \ldots, A_k$ містить незалежно від розміщення дужок рівно ті елементи, які з'являються в непарному числі множин A_i. Отже, у симетричній різниці множин $A_1, A_2, \ldots, A_k$ ми можемо не вживати дужки.

Множину C ребер графу називають псевдоциклом, якщо кожна вершина графу $<V, C>$ має парний степінь (пуста множина і довільний цикл графу). Симетрична різниця довільного числа псевдоциклів є псевдоциклом. Відома також теорема [165]:

Теорема 14.1. *Нехай $G = <V, E>$ зв'язний неорієнтований граф, а $<V, T>$ — його остове дерево. Довільний цикл графу G можна однозначно задати як симетричну різницю деякого числа фундаментальних циклів. У загальному випадку довільний псевдоцикл C графу G можна однозначно виразити як $C = \oplus C_e$, де $e = C \setminus T$.*

В. Ліпський запропонув також ідею алгоритму знаходження фундаментальних циклів, котрий будується на пошуку в глибину і має структуру, аналогічну алгоритму знаходження остового дерева. Кожна нова вершина розміщується у стеку і видаляється з нього після відвідування. Зрозуміло, що стек завжди містить послідовність вершин від вершини відвідування v до кореня. Тому, якщо ребро аналізу $\{v, u\}$, замикає цикл, тоді вершина u за останньою теоремою знаходиться в стеку і цикл, що замикається ребром $\{v, u\}$ задаватиметься верхньою групою елементів стеку, які починаються з v і закінчуються вершиною u.

У задачах розподілу важливе місце посідає підзадача знаходження компонент двозв'язності графу.

Вершину a неорієнтованого графу $G = <V, E>$ називають точкою з'єднання, якщо видалення цієї вершини і всіх інцидентних ребер приводить до збільшення числа компонент зв'язності графу.

Неорієнтований граф називається двозв'язним, якщо він зв'язний і не має точок з'єднання. Довільний максимальний двозв'язний підграф графу G називається компонентою двозв'язності або блоком цього графу.

Двозв'язність графу — бажана властивість для багатьох задач, наприклад для задачі маршрутизації в інформаційній системі. Розглянемо граф з рис. 14.10, *а*). Точками з'єднання в ньому будуть вершини V_4 і V_6 (його блоки наводяться на рис. 14.10, *б*).

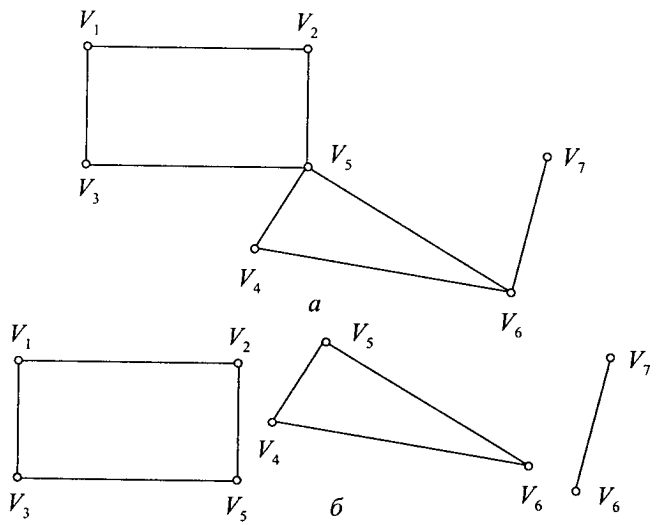


Рис. 14.10. *а* **— граф з точками з'єднання;** *б* **— блоки цього графу**

Якщо $<V_1, B_1>$, $<V_2, B_2>$ — два різні блоки графу G, тоді $V_1 \cap V_2 = \varnothing$ або $V_1 \cap V_2 = \{a\}$, де a — точка з'єднання графу G.

Знаходження точок з'єднання і блоків графу можна вирішити, застосувавши методологію пошуку в глибину [16]. Ідея алгоритму базується на результатах нижченаведеної теореми [165].

Теорема 14.2. *Нехай $D = <V, T>$ — остове дерево з коренем r зв'язного графу $G = (V, E)$, побудоване процедурою пошуку в глибину. Вершина $v \in V$ є точкою з'єднання графу G тоді і тільки тоді, коли або $v = r$ і r має не менше двох синів в D, або $v \neq r$ і існує син w вершини v, такий що ні w, ні будь-який з його нащадків не зв'язані ребром з жодним предком v.*

Отже, для знаходження компонент двозв'язності достатньо провести пошук у глибину з деякої вершини r, обчислюючи для кожної вершини v два параметри: WGN [v] і L [v]. WGN [v] визначає номер вершини v у порядку, в якому вершини відвідуються при пошуку в глибину, починаючи з вершини r. L [v] визначає найменше значення WGN [u], де $u = v$ або вершина u зв'язана хордою з вершиною v або її довільним нащадком у D, де $D = <V, T>$ — дерево, яке відповідає нашому пошуку в глибину. Параметр

L [v] обчислюється індукцією відносно дерева D, якщо відомі L [w] для всіх синів w вершини v. Позначивши

$A = \min \{L [w] : w$ — син вершини $v\}$, $B = \min$ [WGN [u] : $\{u, v\} \in E \setminus T\}$,

матимемо L [v] = min {WGN [v], A, B}. З теореми випливає, що v буде точкою з'єднання або коренем тоді і тільки тоді, коли L [w] ≥ WGN [v] для деякого сина u вершини v (припустимо, що $n > 1$).

14.7. Ейлерові шляхи

Для багатьох прикладних проблем актуальною залишається задача знаходження ейлерових шляхів. Такими у графі називають шляхи, які проходять через кожне ребро графу тільки один раз. Інакше кажучи, $v_1, \ldots, v_{m-1}$ такий, що кожне ребро $e \in E$ з'являється в послідовності $v_1, \ldots, v_m$ тільки один раз, як $\{v_i, v_{i+1}\}$. Коли $v_1 = v_m$, тоді такий шлях називається ейлеровим циклом.

Задача існування ейлерового циклу має давню історію. Місто Кенігсберг мало багато каналів і мостів через них. Виникла природна задача: чи можна обійти всі мости, побувавши на кожному тільки один раз, і повернутись назад. Ця задача була розв'язана Л. Ейлером в 1736 р. Він сформулював теорему (першу теорему теорії графів) про необхідні і достатні умови існування такого шляху.

Теорема 14.3. *У графі існує ейлерів шлях тоді і тільки тоді, коли граф зв'язний і містить не більше двох вершин непарного ступеня.*

Якщо в зв'язному графі немає вершин непарного ступеня, тоді довільний ейлерів шлях буде циклом, оскільки кінці ейлерового шляху, який не є циклом, будуть вершинами наступного степеня. Припустимо, що u і v — єдині вершини непарного степеня зв'язного графу $G = <V, E>$. Створимо граф G^*, додавши додаткову вершину t і ребра $\{u, t\}$ і $\{v, t\}$ (або просто $\{u, v\}$, якщо $\{u, v\} \notin E$). Тоді G^* — зв'язаний граф без вершин непарного степеня, а ейлерові шляхи в G знаходитимуться у взаємно-однозначній відповідності з ейлеровими циклами в G^*. Тому далі можна розглядати тільки ейлерові цикли.

Основна ідея алгоритму полягає в такому. Зафіксувавши будь-яку з вершин як початкову (наприклад, вершина з номером 1), будується шлях від неї до тих пір, доки шлях не можна подовжити, включивши нову вершину. Ребра цього шляху видаляються з графу. Процес повинен закінчитись при поверненні в початкову вершину, інакше це означало б непарність вершини v_{m+1} обраного шляху $v_1, \ldots, v_{m+1}$. Отже, з графу видалився цикл, для запам'ятовування вершин якого можна використати стек. Можливо довести, що після такого видалення степінь довільної вершини залишається парною. Вибрана початкова вершина переноситься з першого стека в другий, і черговою вершиною аналогічного процесу побудови шляху є верхній елемент першого стека. Знову ж через парність ступенів усіх вершин графу знайдеться новий цикл. Він аналогічно додається до першого стека.

Процес повторюється до моменту, коли перший стек є пустим. Очевидно, що вершини, розміщені в другому стеку, утворюють для них шлях. Також відмітимо, що вершина переноситься в другий стек тільки у разі, коли всі її ребра задані парами сусідніх вершин в одному із стеків. Тому після закінчення перебору другий стек міститиме ейлерів цикл.

14.8. Знаходження найкоротших шляхів у графі

Важливе місце в теорії графів посідає задача знаходження шляхів між вершинами графу, особливо мінімальних. Вона має великий теоретичний інтерес і значне практичне застосування: складання розкладу виконання робіт, транспортна задача, ефективний розподіл електроенергії і т. п. Нагадаємо, що ця задача має безпосереднє відношення до планування в просторі станів, оскільки розв'язок багатьох інтелектуальних задач можна звести саме до пошуку найкоротшого шляху на графі.

У цьому параграфі розглянемо навантажені орієнтовані графи. Нагадаємо, що довжина шляху між довільними двома вершинами визначається як сума ваг ребер, що входять у цей шлях. Якщо в довільному графі вага кожного ребра дорівнює 1, тоді отримаємо традиційне визначення довжини шляху через кількість ребер; довжина нульового шляху дорівнює 0. Цикл в орієнтованому графі іноді ще називають контуром. Серед усіх шляхів між двома вершинами u, v найцікавішим є найкоротший шлях. Довжину його позначають $d(u, v)$ і називають *відстанню* від u до v. Зрозуміло, що визначена таким чином відстань може бути і від'ємною. Якщо не існує жодного шляху з u до v, тоді покладають $d(u, v) = \infty$.

Відмітимо [165], що, якщо кожний контур графу має додатню довжину, тоді найкоротший шлях буде завжди елементарним шляхом. Якщо ж в графі існує контур від'ємної довжини, тоді відстань між деякими парами вершин є невизначеною (обходячи цей контур потрібну кількість разів, можна визначити шлях між цими вершинами з довжиною, меншою за довільне дійсне число). У даному разі можна говорити про довжину найкоротшого елементарного шляху, але ця задача буде значно складнішою.

Для вирішення багатьох задач корисно розглядати спеціальні типи графів, структура яких має достатнє застосування на практиці. Одним з таких класів графів є безконтурні графи з виокремленою однією вершиною-джерелом. Вершину орієнтованого графу, в яку не входить жодне ребро, називають джерелом. Якщо граф не має такої вершини, досить часто таку вершину вводять фіктивно. Справа в тому, що для багатьох, більш ширших класів графів не існує кращих алгоритмів за ті, що працюють з графами, в яких виокремлена вершина-джерело.

Основою більшості алгоритмів знаходження відстаней від джерела до всіх вершин в ациклічному непустому графі є такі факти [165, 295, 296]:

1) вершини можна перенумерувати так, що кожна дуга матиме вигляд $<v_i, v_j>$, де $i < j$;

2) існує вершина, в яку не заходить жодна дуга.

Щоб пересвідчитись у цьому, вибирають довільну вершину w_1, потім вершину w_2, таку що $<w_2, w_1> \in E$, потім вершину w_3, таку що $(w_3, w_2) \in E$ і т. д. За скінченну кількість кроків ми повинні дійти до деякої вершини w_i, в яку не заходить жодна дуга, бо через ациклічність жодна вершина не може повторюватися в послідовності w_1, w_2, w_3, ...

Вершини, в які не заходить жодна дуга, можна зберегти в стеку. Після початкового заповнення стека такими вершинами починаємо аналізувати його вершини. Вибираємо верхній елемент стека — вершину u, якій присвоюється мінімальний номер із ще не використаних номерів. Цим ми гарантуємо, що всі дуги, які виходять з вершини u, будуть вести до вершини з більшими номерами. Потім вершина u разом з дугами, що виходять з неї, видаляється з графу. Це зменшує на одиницю число дуг, що заходять в кожну вершину v зі списку інцидентності вершини u, тобто таких, що $(u, v) \in E$. Отже, для цього в структуру вузла головного списку задання потрібно додати поле **count**, значення котрого визначатиме напівстепінь входу відповідної вершини (кількість вхідних дуг вершини). Якщо для деякої з вершин значення **count** дорівнює нулю, тоді ця вершина додається до стека. Через ациклічність графу повне спустошення стека, що приводить до зменшення роботи алгоритму, настає тільки після того, як всі вершини графу будуть перенумеровані.

Для контролю над числом вхідних вершин введемо в структуру вузла головного списку поле **count**, яке міститиме напівстепінь входу для даної вершини. Отже, опис вузла графу може бути таким:

```
type pntrp = ^recp;
     recp = record
              flag : boolean;
              right, left : pntrq;
              nextt : pntrp;
              count, val : word
              end;
```

Поле **flag** використовуємо для позначки того, чи вершина вже перейменована, чи ні (**true** означає перейменована). Поля **right, left** — це вказівники на початок списку вихідних і вхідних вершин відповідно. Ці списки складаються з вузлів, що містять два поля: **info** — номер вузла, **next** — вказівник на наступний елемент списку. Поле **val** містить поточний номер вершини.

Якщо після видалення з графу певної вершини і дуг значення поля **count** для якоїсь з вершин, суміжних видаленій, стало рівним нулю, то її теж заносимо до стека та аналізуємо за описаним вище принципом. Повне спустошення стека означатиме, що всі вершини вже перенумеровані. Процедура перенумерації **numbering** повинна включати процедуру **numeringlispin**, завдання якої — забезпечити відповідність списків суміжності з новим розкладом номерів вершин графу. Процедури **Addst** і **deletest** традиційно ви-

конуватимуть роль внесення і зчитування вершини стека. Потрібна також процедура **Findpointn**, що шукає вершину графу, номер якої дорівнює параметру процедури, починаючи пошук з вершини, на яку вказує змінна **heap**. Масив *D* використаємо для збереження зв'язку між старими і новими номерами вершин. У двовимірному масиві *A* зберігаємо відстані між вершинами.

Підрахунок відстані від джерела до кожної вершини робить нижченаведена частина програми:

```
Begin
    numbering (head);
    for j := 1 to n do
    begin
        for k := 1 to n do
        begin
            a1 [d [j], d [k]] := a [j, k]
        end
    end;
    d1 [1] := 0;
    for j := 2 to n do
    d1 [j] := maxreal;
    for j := 2 to n do
    begin
        p := findpoint (j, head);
        q1 := p^.left;
        while q1 <> nil do
        begin
            if d1 [j] > d1 [q1^.info] + a1 [q1^.info, j] then
            d1 [j] := d1 [q1^.info] + a1 [q1^.info, j];
            q1 := q1^.next
        end;
    end;
    for j := 1 to n do
    begin
        k := 1;
        while d [k] <> j do
        inc (k);
        writeln (d1 [j], ' — відстань до вершини ', k);
    end;
end;
```

Визначимо часову складність алгоритму. Припустимо, що максимальний напівстепінь входу вершин графу дорівнює *d*. Тоді цикл визначення напівступенів входу витрачатиме $O(nd)$ кроків, де n — кількість вершин графу. Цикл початкового занесення до стека з нульовою кількістю попередників не перевищуватиме попередню оцінку.

Загальний цикл перенумерації (**while st <> nil do …**) має n кроків. В його тілі знаходиться виклик процедури **numeringlispin ()**. Остання потребує на виконання час, пропорційний $O(nd)$. На зменшення значення поля **count** для всіх вершин, що виходять з тільки-но перенумерованої вершини (цикл **while q <> nil do**), потрібно $O(d)$ кроків. Тому часова складність циклу перенумерації $O(n^2d)$ визначає і всю часову складність алгоритму нумерації.

Наведений вище алгоритм перенумерації вершин за своєю структурою нагадує алгоритм топологічного впорядкування [142].

Тоді процедура **finddist** відшукуватиме відстані від джерела v до всіх вершин в ациклічному графі. Граф $G = <V, E>$ задається списками інцидентності. На голову цього списку вказує **heap**. Результатом будуть відстані від v_1 до всіх вершин графу:

$$D[v_i] = d(v_1, v_i),\ i = 1, 2, \ldots, n.$$

```
Procedure finddist (v, n: 1..n1; heap: line; A: ar22;var D: arr11);
const maxreal = 1000000.0;
Var j: integer;
    p: line;
    q: line1;
Procedure findpointn (…;
Procedure numbering (…;
Begin
   numbering (heap);
   D [1] := 0;
   for j:= 2 to n do
   D [j] := maxreal; {* maxreal позначає + ∞ *}
   for j:= 2 to n do
   begin
      p:= findpointn (j, heap);
      q:= p^.left;
      while q <> nil do
      if D [j] > D [q^.info] + A [q^.info, j] then
      D [j] = D [q^.info] + A [q^.info, j]
   end;
end;
```

Зрозуміло, що часова оцінка процедури **finddist** визначатиметься $O(n^2d)$.

Для розв'язку багатьох задач потрібно також знати алгоритм найкоротших шляхів між усіма парами вершин у графі. Зрозуміло, що для цього можна використати алгоритм знаходження найкоротших шляхів від джерела до всіх вершин у графі з тільки-но розглянутого розділу, вибираючи в черговий раз за джерело нову вершину графу. Процес повторити потрібно стільки разів, скільки є вершин у графі. Але часова складність такого алгоритму буде не найкращою.

Розглянемо два найшвидші алгоритми знаходження відстаней між довільними вершинами в різнотипних графах.

Розглянемо орієнтований граф $G = <V, E>$, де $V = \{v_1, v_2, \ldots, v_n\}$ з матрицею ваг $A = [a_{ij}] = [a(v_i, v_j)]$. Позначивши через d_{ij}^m довжину найкоротшого шляху з v_i і v_j, який містить не більше m дуг, отримують рівняння:

$$d_{ij}^0 = \begin{cases} 0, \text{ якщо } i = j \\ 1, \text{ якщо } i \neq j, \end{cases}$$

$$d_{ij}^{m+1} = \min\{d_{ik}^m + a_{kj} : 1 \le k \le n\}.$$

Останнє рівняння за структурою нагадує визначення добутку двох квадратних матриць: операцію min асоціюють з "сумою", а операцію "+" — з "добутком двох матриць". Позначимо такий "добуток" двох матриць A і B через $A*B$. Відмітимо, що одиничним елементом служить матриця:

$$U = \begin{pmatrix} 0 & \infty & \infty & \ldots & \infty \\ \infty & 0 & \infty & \ldots & \infty \\ \infty & \infty & 0 & \ldots & \infty \\ \ldots & \ldots & \ldots & \ldots & \ldots \\ \infty & \infty & \infty & \ldots & 0 \end{pmatrix}$$

Зрозуміло, що $[d_{ij}^0] = U$ і $d_{ij}^m = \underbrace{((\ldots((A * A) * A)\ldots) * A)}_{m}$ $(m \ge 1)$.

Можливий один з варіантів:

1) $d_{ij}^{n-1} = d_{ij}^n$, і в результаті $d_{ij}^m = d_{ij}^{n-1}$ для кожного $m \ge n$, тоді $d_{ij}^{n-1} = d(v_i, v_j)$.

2) $d_{ij}^{n-1} \neq d_{ij}^n$, — це говорить про те, що граф має цикл від'ємної довжини.

Враховуючи, що добуток $A*B$ двох матриць розмірності $n \times n$ можна обчислити за час $O(n^3)$, матрицю $[d_{ij}^{n-1}]$, а отже і відстані між всіма парами вершин можна обчислити за час $O(n^4)$.

Але часову оцінку можливо знизити, скориставшись фактом, що у формулі визначення d_{ij}^m операція "*" асоціативна — можемо обчислювати добуток визначення d_{ij}^m поетапно, покроково підносячи матрицю A до квадрата. Це призведе до заміни $n-1$ множень матриці на $\lceil \log n \rceil$ множень. Отже, знайти відстані між усіма парами вершин у графі без циклів (контурів) від'ємної довжини можна за час $O(n^3 \log n)$.

Ще одну оптимізацію запропонували С. Уоршал [328] і Р. Флойд [295]. Позначимо через d_{ij}^m довжину найкоротшого шляху із вершини v_i у вершину v_j з проміжними вершинами на множині $\{v_1, \ldots, v_m\}$. Ми хочемо знайти найкоротший шлях від v_i до v_j, але такий, що вже проходить і через вершину v_{m+1}. Якщо такий шлях не містить вершину v_{m+1}, тоді $d_{ij}^{m+1} = d_{ij}^m$. В іншому разі матимемо можливість розбити весь шлях на два підшляхи $\{v_i, \ldots, v_{m+1}\}$ і $\{v_{m+1}, \ldots, v_j\}$. Тоді матимемо такі рівняння:

$$d_{ij}^0 = a_{ij},$$

$$d_{ij}^{m+1} = \min(d_{ij}^m, d_{ij}^m + d_{mj}^m).$$

З цих рівнянь можна обчислити відстань $d(v_i, v_j) = d_{ij}^m$, $1 \le i, j \le n$. Процедура **finddistfloyd** реалізує відповідну ідею знаходження відстані між всіма парами вершин в ациклічному графі без контурів від'ємної довжини.

```
Procedure finddistfloyd (n: arr11; A: arr22; var D: arr22);
{* A — матриця ваг дуг графу *}
Var i, j, m: integer;
begin
   for i := 1 to n do
   for j := 1 to n do
   D [i, j] := A [i, j];
   for i := 1 to n do
   D [i, j] := 0;
   for m := 1 to n do
   for i := 1 to n do
   for j := 1 to n do
   if D [i, m] + D [m, j] < D [i, j] then
   D [i, j] := D [i, m] + D [m, j]
end;
```

Зрозуміло, що часова складність процедури буде $O(n^3)$.

Важливо відмітити, що поки невідомо алгоритм, який знаходив би відстань між двома фіксованими вершинами за час, кращий за $O(n^3)$.

Під бінарним відношенням на множині V ми розуміємо довільну підмножину $E \subseteq V \times V$. Відношення буде транзитивним, якщо задовольняється умова: коли $<x, y> \in E$, $<y, z> \in E$, тоді і $<x, z> \in E$ для довільних $x, y, z \in V$. Зрозуміло, що довільне бінарне відношення можна зобразити орієнтованим графом $G = <V, E>$. Для довільного такого відношення можна виразити транзитивне відношення $E^* = \{<x, y>$: в $<V, E>$ існує шлях ненульової довжини із x в $y\}$. До того ж, E^* є найменшим транзитивним відношенням, що містить E. Відношення E^* називають *транзитивним замиканням* відношення E.

Якщо відношення E задати графом $<V, E>$ з вагами дуг, що дорівнюють одиниці, тоді транзитивне замикання E^* можна отримати за допомогою процедури **finddistfloyd** за час $O(n^3)$. Після завершення роботи алгоритму матимемо: $<v_i, v_j> \in E$ тоді і тільки тоді, коли $D[I, j] < +\infty$.

С. Уоршалл [328] запропонував ввести таку модифікацію. Матрицю ваг зручно задавати таким чином:

$$A[i,j] = \begin{cases} 0, & \text{якщо } < v_i, v_j > \notin E, \\ 1, & \text{якщо } < v_i, v_j > \in E. \end{cases}$$

Тоді рядок результату визначення добутку в процедурі **finddistfloyd** можна замінити на такий запис:

D [i, j] := D [i, j] and (D [o, m] or D [m, j]),

де **and** і **or** — традиційні булеві операції диз'юнкції та кон'юнкції відповідно. Тоді після завершення алгоритму

$$D[i,j]=\begin{cases}1, & \text{якщо } <v_i, v_j> \in E^*,\\ 0, & \text{якщо } <v_i, v_j> \notin E^*.\end{cases}$$

Коли відношення E симетричне, зручно будувати його транзитивне замикання за допомогою методів пошуку в глибину або ширину.

14.9. Загальна схема алгоритму Харта, Нільсона і Рафаеля

Узагальненням для багатьох алгоритмів на графах, розглянутих у цьому розділі, є алгоритм Харта, Нільсона і Рафаеля, призначений для пошуку найкоротшого шляху між двома вершинами графу. Розглянемо основні ідеї цього алгоритму [200, 320].

Введемо такі позначення:

s — початкова вершина;

q — цільова вершина;

$c(i,j)$ — довжина ребра, яке з'єднує i-ту вершину з j-ю;

$d(i,j)$ — довжина найкоротшого шляху між i-ю та j-ю вершинами; зокрема; $g(n)$ — довжина найкоротшого шляху від початкової вершини до n-ї; $h(n)$ — довжина найкоротшого шляху від n-ї вершини до цільової; $f(n)$ — довжина найкоротшого шляху від початкової вершини до цільової серед усіх шляхів, які проходять через n-ну вершину; при цьому $f(n)=g(n)+h(n)$;

$\hat{g}(n)$ — оцінка довжини найкоротшого шляху від початкової вершини до n-ї; ця оцінка змінюється, якщо знайдено деякий новий (не обов'язково оптимальний) шлях до n-ї вершини; для оптимального шляху $\hat{g}(n)=g(n)$.

$\hat{h}(n)$ — ***евристична функція***, яка задає оцінку довжини найкоротшого шляху від n-ї вершини до цільової; у переважній більшості випадків ми **не можемо знати точне значення** $h(n)$, але можемо **оцінити** його, виходячи зі специфіки конкретної задачі; приклади таких оцінок будуть наведені далі;

$\hat{f}(n)=\hat{g}(n)+\hat{h}(n)$ — оцінка для $f(n)$;

$L(n)$ — множина всіх наступників вершини n. Говоритимемо, що алгоритм ***розкриває*** вершину n, якщо він знаходить множину її наступників.

Далі нам будуть потрібні дві робочі множини: *OPEN* і *CLOSE*.

Відповідно з [320] загальну схему пошуку найкоротшого шляху на графі від початкової вершини до цільової можна описати так:

1. Сформувати граф пошуку G, який на початку роботи алгоритму складається з однієї початкової вершини s. Занести s до *OPEN*. Покласти $\hat{g}(s)=g(s)=0$.
2. Сформувати множину *CLOSE*, яка на початку роботи алгоритму є порожньою.
3. Якщо множина *OPEN* порожня, вихід: невдача, потрібного шляху не існує.

4. Взяти з множини *OPEN* перший елемент m (відповідно до порядку, встановленому кроком 9). Вилучити m з *OPEN* та занести її до *CLOSE*.
5. Якщо m — цільова вершина, успіх. Відновити шлях від s до m на основі відновлюючих вказівників, що були встановлені на кроці 6—8, і завершити алгоритм.
6. Розкрити вершину m: отримати множину наступників $L(m)$. Додати до графу G всі вершини, які належать $L(m)$, але не належать G, разом з відповідними дугами. Додати ці вершини до *OPEN*. Для кожної вершини $k \in L(m) \setminus G$ обчислити оцінки $\hat{f}(k) = \hat{g}(k) + \hat{h}(k)$, поклавши $\hat{g}(k) = \hat{g}(m) + c(m,k)$; встановити з k до m відновлюючий вказівник.
7. Для кожної вершини n з тих, які потрапили до $L(m)$, але вже належали *OPEN* або *CLOSE*, переобчислити оцінки $\hat{g}(n) = \min((\hat{g}(m) + c(m,n)), \hat{g}(n))$. Якщо для деякої вершини попередня оцінка, що була обчислена раніше, зменшилася, перевстановити з неї відновлюючий вказівник до m (в напрямку найкоротшого шляху).
8. Для всіх вершин з $L(m)$, які до цього знаходилися в *CLOSE*: перевстановити відновлюючі вказівники для кожного з наступників цих вершин у напрямку найкоротшого шляху, а також переобчислити оцінки наступників.
9. Переупорядкувати множину *OPEN* за зростанням значень $\hat{f}$.
10. Повернутися на крок 3.

Взагалі евристичну функцію $\hat{h}$ в алгоритмі Харта, Нільсона і Рафаеля можна обирати довільно і, залежно від вибору цієї функції, надійність і якість алгоритму можуть суттєво змінюватися. Сформульовані три важливі теореми, які встановлюють, за яких умов алгоритм гарантовано матиме ті чи інші бажані властивості. Доведення цих теорем можна знайти в [320].

Алгоритм Харта, Нільсона і Рафаеля називається ***допустимим***, якщо він гарантовано знаходить оптимальний шлях до цільової вершини.

Теорема 14.4. *Алгоритм Харта, Нільсона і Рафаеля є допустимим, якщо виконуються такі умови:*

- *будь-яка вершина або взагалі не має наступників, або має скінченну кількість наступників;*
- *довжина будь-якої дуги перевищує деяку величину* $\varepsilon > 0$;
- *для будь-якої вершини* n *виконується нерівність* $\hat{h}(n) \le h(n)$,

тобто евристична функція є оцінкою знизу і не перевищує справжньої відстані до цільової вершини.

Нехай два допустимих алгоритми Харта, Нільсона і Рафаеля A і B відрізняються лише своїми евристичними функціями $\hat{h}_A$ та $\hat{h}_B$. Кажуть, що алгоритм A є ***інформованішим***, ніж алгоритм B, якщо для будь-якої нецільової вершини $\hat{h}_A(n) > \hat{h}_B(n)$.

Теорема 14.5. *Якщо алгоритм A є інформованішим, ніж алгоритм B, то при завершенні роботи алгоритму для будь-якого графу, для якого існує шлях до цільової вершини, будь-яка вершина, що розкривається алгоритмом A, розкривається і алгоритмом B.*

Інакше кажучи, теорема стверджує, що інформованіший алгоритм розкриває не більше вершин, ніж менш інформований, і тому є оптимальнішим за швидкодією.

Наступна важлива вимога, яка накладається на евристичну функцію, полягає в її ***консистентності*** (від англ. consistency). Евристична функція $\hat{h}(n)$ називається консистентною, якщо для будь-яких двох вершин m і n таких, що m є безпосереднім наступником n, виконується нерівність $\hat{h}(n)-\hat{h}(m)\le c(n,m)$.

Це означає, що при переході від будь-якої вершини до її наступника консистентна евристична функція не може зменшитися на величину, що перевищує довжину з'єднуючого ребра.

Теорема 14.6. *Якщо евристична функція, що використовується в алгоритмі Харта, Нільсона і Рафаеля, є консистентною, то алгоритм розкриває будь-яку вершину не раніше, ніж знайде оптимальний шлях до неї.*

Інакше кажучи, якщо виконується умова консистентності, то в момент, коли алгоритм розкриває вершину n, для неї виконується рівність $\hat{g}(n)=g(n)$.

Консистентність евристичної функції має дуже велике значення. Вона дозволяє не повертатися до тих вершин, які вже були проаналізовані і потрапили до множини *CLOSE*.

Якщо евристична функція не використовується, тобто якщо $\hat{h}(n)\equiv 0$ і, відповідно, $\hat{f}(n)=\hat{g}(n)$, маємо **алгоритм Дейкстри**. Теорема 14.5 стверджує, що введення належної евристичної функції дозволяє у більшості випадків прискорити роботу алгоритму.

Якщо взяти $\hat{g}(n)\equiv 0$, утворюється **алгоритм Дорана і Мічі** [129]. Цей алгоритм у ряді випадків дозволяє швидко отримати шлях до цільової вершини. Але гарантії того, що цей шлях буде оптимальним, немає. Більше того, не гарантується навіть знаходження хоча б якогось шляху до цільової вершини.

Проілюструємо методику застосування алгоритму Харта, Нільсона і Рафаеля до вирішення головоломки "гра у 8". Цю головоломку можна розглядати як спрощений варіант відомої головоломки С. Лойда "гра у 15". В коробці розміром 3 × 3 лежать 8 фішок, що пронумеровані цифрами від 1 до 8; одне місце є пустим (рис. 14.11).

Фішки можна довільно переміщувати, але не можна виймати з коробки. Задача полягає в тому, щоб будь-яку початкову позицію (початкову розстановку фішок) перевести до цільової позиції, зображеної на рис. 14.12 (якщо це можливо), при цьому потрібно зробити це за мінімальну кількість ходів.

1	5	7
6	8	
2	4	3

Рис. 14.11. Головоломка "гра у 8"

1	2	3
8		4
7	6	5

Рис. 14.12. Цільова позиція

Не будь-яку початкову позицію можна з додержанням правил гри перевести до цільової. Можна показати, що **це можливо тільки для тих позицій, які переходять до цільової за парну кількість перестановок**. ***Перестановкою*** називається така операція: дві фішки виймаються з коробки та міняються місцями. Втім, якщо для даної початкової позиції задача не має розв'язку, це буде з'ясовано в ході роботи алгоритму.

Розв'язок задачі можна автоматизувати на основі планування в просторі станів; побудова графу станів є очевидною. Кожна вершина графу відповідає певній позиції; якщо існує хід, який дозволяє перейти від позиції *A* до позиції *B*, то з *A* до *B* йде орієнтована дуга.

Так, у позиції на рис. 14.11 можна зробити два ходи:

пересунути фішку "8" вправо;

пересунути фішку "3" вгору.

Тому відповідний фрагмент графу станів має вигляд, зображений на рис. 14.13:

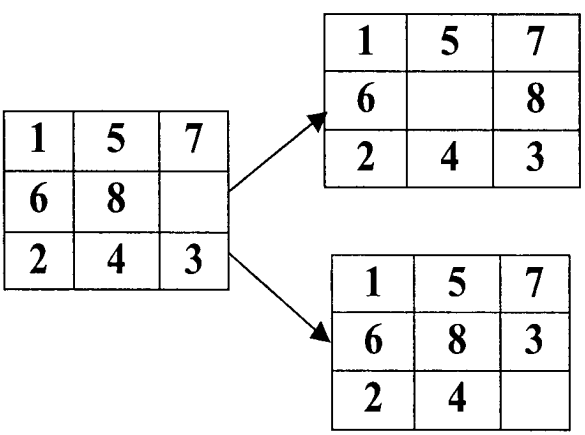


Рис. 14.13. Фрагмент графу станів

Тепер, коли граф станів побудовано, для розв'язання задачі можна застосовувати будь-який алгоритм пошуку найкоротшого шляху. Для алгоритму Харта, Нільсона та Рафаеля необхідно задати конкретні евристичні функції. У [200] розглядаються такі функції:

- $\hat{h}_1$ — кількість фішок, які стоять не на своїх місцях;

- $\hat{h}_2$ — сума відстаней усіх фішок до свого місця, тобто сума ходів, які потрібно зробити, щоб перевести кожну фішку на своє місце, за умови, що інших фішок на дошці немає.

Легко побачити, що обидві евристичні функції є допустимими, тобто є оцінками знизу. Обидві функції також є консистентними. Легко також бачити, що алгоритм, який використовує функцію $\hat{h}_2$, є інформованіший, ніж той, який використовує функцію $\hat{h}_1$.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте методику розв'язання інтелектуальних задач на основі планування в просторі станів.
2. Що таке граф станів задачі?
3. Дайте визначення поняття "граф".
4. Охарактеризуйте способи зберігання графів.
5. Що таке матриця суміжності і списки суміжності?
6. Що таке дерево? Чи можна розглядати дерево як частковий випадок графу?
7. Охарактеризуйте пошук у глибину і ширину.
8. Що таке остові дерева?
9. Опишіть алгоритми знаходження ейлерових циклів.
10. Опишіть загальну схему алгоритму Харта, Нільсона і Рафаеля.
11. Що таке евристична функція?
12. В якому випадку алгоритм Харта, Нільсона і Рафаеля є допустимим?
13. Що означає твердження "алгоритм А є інформованіший, ніж алгоритм В"? У чому полягають переваги інформованішого алгоритму?
14. У чому полягає вимога консистентності? Чому вона є важливою?
15. В якому випадку алгоритм Харта, Нільсона і Рафаеля зводиться до алгоритму Дейкстри?
16. Охарактеризуйте алгоритм Дорана і Мічі.
17. Наведіть відомі вам приклади евристичних функцій для "гри в 8".

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Порівняйте способи зберігання графів. Для яких задач вигіднішим виявляється зв'язане зберігання, а для яких — матриці суміжності?
2. Наведіть власний приклад задачі, яку можна розв'язувати на основі планування в просторі станів з використанням алгоритму Харта, Нільсона і Рафаеля. Якими можуть бути евристичні функції для цього прикладу?

ЗАДАЧІ І ВПРАВИ

1. Доведіть припустимість і консистентність евристичних функцій для "гри в 8", які розглядалися в п. 12.5.
2. Доведіть, що якщо алгоритм Харта, Нільсона і Рафаеля є допустимим, то евристична оцінка цільової вершини завжди дорівнює нулю.
3. Наведіть приклад невдалого вибору евристичної функції, при застосуванні якої доводиться постійно повертатися до тих вершин, які уже були розкриті. (Вказівка: у даному разі повинна порушуватися вимога консистентності.)

Розділ 15
ПЛАНУВАННЯ В ПРОСТОРІ ЗАДАЧ

Поділяй і пануй...
Давньоримський афоризм

15.1. Базові поняття

Ідеологія планування в просторі задач полягає в такому. Задача, яку необхідно розв'язати, розбивається на підзадачі; кожна з цих підзадач, у свою чергу, поділяється на простіші підзадачі і т. д. Таке розбиття ***(декомпозиція)*** продовжується, поки задача не зведеться до елементарних.

Елементарною *називається задача, для якої існує готовий алгоритм розв'язку.*

Якщо ж для деякої задачі готового алгоритму не існує, але її подальша декомпозиція неможлива, таку задачу слід вважати нерозв'язною.

Для планування в просторі задач традиційно використовується формалізм *І-АБО*-графів. ***І-АБО-графи*** зарекомендували себе як зручний спосіб представлення задач і методів їх розв'язання.

І-АБО-графом *називається орієнтований граф, вершини якого відповідають задачам, а дуги — відношенням між задачами. Між дугами вводяться відношення І та АБО.*

Нехай задача A зв'язана дугами із задачами B та C (при плануванні розв'язків це означає, що задачу A можна звести до задач B та C). Говоритимемо, що дуга AB зв'язана з дугою AC відношенням I, якщо для визначення A необхідно вирішити і B, і C. Якщо ж для визначення A достатньо розв'язати лише одну з цих задач, то казатимемо, що відповідні дуги зв'язані відношенням *АБО*.

Будь-який І-АБО-граф може бути зведений до певної нормальної форми, в якій з будь-якої вершини виходять або тільки І-дуги, або тільки АБО-дуги.

Вершину, з якої виходять лише І-дуги, називатимемо ***І-вершиною****, вершину, з якої виходять лише АБО-дуги, —* ***АБО-вершиною****.*

Одну з вершин графу, що відповідає початковій задачі, називатимемо ***початковою вершиною***.

Листи І-АБО-графу (тобто завершальні вершини, з яких не виходять дуги) відповідають або елементарним, або нерозв'язним задачам.

Вершина називається розв'язною, якщо задача, що відповідає цій вершині, має розв'язок.

Метою пошуку, що здійснюється на І-АБО-графі, є встановлення того, чи є початкова вершина розв'язною, чи ні.

І-АБО-графи тісно пов'язані з продукційними системами (див. п. 8.2) і в ряді випадків **фактично є фрагментами мереж виведення продукційних систем**.

Згадаємо таку продукційну систему:

$A \rightarrow C$

$B \rightarrow C$

$B, F \rightarrow L$

$F \rightarrow Q$

$D, C \rightarrow G$

Мережа виведення має такий вигляд:

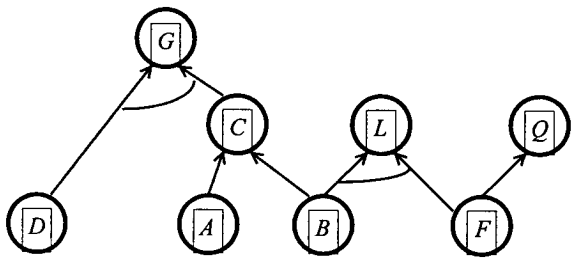


Рис. 15.1. Приклад мережі виведення

Тоді для розв'язання задачі G необхідно вирішити і задачу D, і задачу C (відношення І). Розв'язання задачі C, у свою чергу, потребує вирішення або A, або B (відношення АБО). Тому І-АБО-граф для задачі G можна зобразити у вигляді:

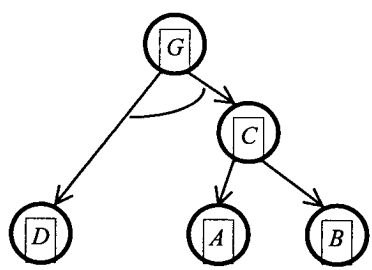


Рис. 15.2. І-АБО-граф як фрагмент мережі виведення

Можна навести таке рекурсивне правило для встановлення того, є дана вершина розв'язною чи ні:

1. *Заключні вершини, які відповідають елементарним задачам, розв'язні.*
2. *І-вершина розв'язна, якщо всі її сини є розв'язними.*
3. *АБО-вершина розв'язна, якщо хоча б один її син є розв'язним.*

Аналогічно можна дати визначення нерозв'язної вершини:

1. *Заключні вершини, які не відповідають елементарним задачам, нерозв'язні.*
2. *І-вершина нерозв'язна, якщо хоча б один її син не є розв'язним.*
3. *АБО-вершина нерозв'язна, якщо всі її сини нерозв'язні.*

Оскільки планування в просторі задач тісно пов'язано з логічним виведенням і продукційними системами, методики такого планування в основ-

ному являють собою ті чи інші бектрекінгові алгоритми. Багато прикладів такого планування, а також зведення конкретних задач до підзадач, можна знайти в [200, 320].

Якщо ж говорити про конкретніші та формалізованіші схеми, то вони тісно пов'язані з ***вирішувачами інтелектуальних задач***, які розглядатимуться у розд. 19.

15.2. Метод "поділяй і пануй"

Нехай нам потрібно обчислити деяку функцію на вхідних даних довжини n. Тоді головна стратегія методу "поділяй і пануй" (ПП) полягає в розділенні входу на k вхідних наборів $1 < k \le n$ і вирішенні загальної задачі на цих наборах. Потім шукаємо алгоритм, який би дозволив знайти загальний розв'язок задачі, комбінуючи розв'язки отриманих підзадач. Часто для вирішення підзадачі потрібно знову використовувати метод ПП. Коли такий поділ має характер базової проблеми, тоді отримуємо рекурсивні алгоритми. Спробуємо формалізувати зазначене.

Нехай вхід задачі довжини n задається даними глобального масиву A $(1..n)$, а загальна проблема, яку потрібно вирішити на цьому вході, описується процедурою **dandz**. Параметри процедури **dandz** $(1, n)$ говорять про вирішення цієї проблеми на початковому вході довжини n, а **dandz** (p, q) — розв'язання підзадачі на вході, що визначається значеннями масиву A $(p..q)$. Булева функція **small** (p, q) визначає достатність **(true)** поділу даних для вирішення задачі, тобто на вході довжини $q - p + 1$ може бути отриманий якийсь проміжний розв'язок за допомогою функції G. Якщо припустити, що значенням **divide** (p, q) є таке m, що розбиває вхід A (p, q) на дві підзадачі A $(p..m)$ і A $(m + 1, q)$, а x та y — це розв'язання задачі на цих підвходах, тоді функція **combine** (x, y) буде функцією побудови загального розв'язку на A (p, q).

Процедура, яка описує структуру загального абстрактного обчислення за методом ПП, може мати вигляд:

```
procedure dandz (p, q: int);
var m: int;
begin
    if small (p, q) then G (p, q)
    else
    begin
    m := divide (p, q) /*p <= m < q */
    combine (dandz (p, m), dandz (m + 1, q))
    end;
end;
```

Для визначення часових оцінок алгоритмів, які базуються на методі ПП, можна запропонувати таку схему:

$$T(n) = \begin{cases} g(n), & \text{якщо } n \text{ достатнє для розв'язку,} \\ 2T(n/2) + f(n) & \text{в іншому випадку,} \end{cases}$$

де $g(n)$ є часовою оцінкою розв'язку задачі на досить малому (достатньому) вході, а $f(n)$ — часом, потрібним для комбінації загального розв'язку з підрозв'язків.

Розглянемо перехід від загальної схеми використання методу ПП до конкретної задачі на прикладі задачі бінарного пошуку.

Нехай маємо послідовність a_i, $i = 1..n$ елементів, які впорядковані за зростанням. Потрібно вирішити задачу належності елемента x цій послідовності чисел, тобто визначити таке j, що $a_j = x$, або $j = 0$, якщо послідовність не має елемента, що дорівнює x.

Формалізуємо нашу задачу так: $I = (n, a_1, a_2, a_3, \ldots, a_n, x)$. Використаємо метод ПП:

$$I = I_1 \cup I_2 \cup I_3,$$

де $I_1 = (k-1, a_1, a_2, a_3, \ldots, a_{k-1}, x)$, $I_2 = (1, a_k, x)$, $I_3 = (n-k, a_{k+1}, \ldots, a_n, x)$.

Якщо $x = a_k$, тоді $j = k$, і підзадачі I_1 та I_3 не потрібно вирішувати. В іншому разі визначається та підзадача, де можливий розв'язок: якщо $x < a_k$, тоді потрібно шукати розв'язок аналогічним чином в I_1, якщо $x > a_k$, тоді в I_3. У випадку подальшого перегляду підзадач I_1 або I_3 і переходу правої межі I_1 через ліву межу I_2 процес пошуку закінчується невдало, тому $j := 0$. Якщо k завжди вибирається таким чином, що a_k є серединним елементом ($k = \lfloor (n+1)/2 \rfloor$), тоді алгоритм результуючого пошуку називається бінарним пошуком.

Розглянемо застосування методу ПП для рекурсивного розв'язку задачі знаходження максимального і мінімального елементів у множині з n елементів. Згідно із загальною схемою проблему $I = (n, a_1, a_2, a_3, \ldots, a_n)$ потрібно розбити на менші підпроблеми (наприклад, дві): нехай $r = \lfloor n/2 \rfloor$,

$I_1 = (li, a_1, a_2, a_3, \ldots, a_r)$,

$I_2 = (n-r, a_{r+1}, a_{r+2}, a_{r+3}, \ldots, a_n)$.

Тоді **max** (I) = **max** (**max** (I_1), **max** (I_2)), а **min** (I) = **min** (**min** (I_1), **min** (I_2)).

Зворотним кроком рекурсії (розбиття далі непотрібне) буде наявність у черговій підмножині розбиття I_k одного ($i := j$) або двох ($i := j - 1$) елементів. У першому випадку **max** = **min** = *значенню елемента*, в другому проводиться порівняння для визначення **max** і **min**. Отримуємо процедуру **maxmin**, яка описує цей процес.

```
procedure maxmin (i, j : int; a : arr; var fmin, fmax : integer);
var gmax, gmin, hmax, hmin : integer;
mid : int;
begin
   case (j – i) of
   0: begin
         fmax := a [i]; fmin := a [j];
      end;
   1: if a [i] < a [j] then
      begin
         fmax := a [j]; fmin := a [i];
```

```
        end;
        else
            begin
                fmax := a [i]; fmin := a [j];
            end;
    else
        begin
            mid := (i + j) div 2;
            maxmin (i, mid, a, gmin, gmax);
            maxmin (mid + 1, j, a, hmin, hmax);
            if gmin < hmin then fmin := gmin
            else fmin := hmin;
            if gmax > hmax then fmax := gmax
            else fmax := hmax;
        end;
    end;
end;
```

Побудуємо часову оцінку роботи процедури maxmin, використовуючи загальну схему побудови часових оцінок для методу ПП:

$$T(n)=\begin{cases} T(n/2)+T(n/2)+2, & n>2 \\ 1, & n=2 \\ 0, & n=1 \end{cases}$$

Припустимо, що $n = 2k$ для деякого додатного k. Тоді

$$T(n) = 2T(n/2) + 2 = 2(2T(n/4) + 2) + 2 = (4T(n/4) + 4) + 2 =$$
$$= 2^{k-1}T(2) + \sum_{i=1}^{k-1} 2^i = 2^{k-1} + 2^k - 2 = 3n/2 - 2.$$

Розглянемо використання методу ПП для вирішення задачі впорядкування послідовності з n цілих чисел.

При використанні методу обмінного впорядкування за зростанням задачу $I = (n, a_1, a_2, \ldots, a_n)$ розбивають на $I_1 = (1, a_1)$ та $I_2 = (n-1, a_2, \ldots, a_n)$. Для цього потрібно знайти найменше значення серед $a_1, a_2, \ldots, a_n$ і поміняти місцями значення a_1 з цим найменшим значенням. Для розв'язку задачі розмірності $n-1$ можна знову використати попереднє розбиття. Повторюючи процес n разів, отримаємо розв'язок початкової задачі. Побудуємо часову оцінку роботи обмінного впорядкування. Для роботи алгоритму потрібно n разів відшукувати найменше значення та записувати його на потрібне місце. Для першого разу потрібно провести $n-1$ порівнянь, на другому — виконати $n-2$ порівнянь і т. д. Таким чином, загальна кількість порівнянь визначатиметься формулою $(n-1) + (n-2) + (n-3) + \ldots + (n-(n-1)) + (n-n) = n \times n - n \times (n+1)/2 = n \times (n-1)/2$. Тому часова оцінка алгоритму буде $O(n^2)$.

Недоліком розглянутого алгоритму є незначне зменшення розмірності задачі на кожному кроці її розбиття на підзадачі.

Швидший шлях розв'язання задачі впорядкування послідовності з n цілих чисел — розбиття чергової задачі $I = (n, a_1, a_2, \ldots, a_n)$ на кожному кроці на дві рівні частини:

$I_1 = (\lfloor n/2 \rfloor, a_1, a_2, \ldots, a_{\lfloor n/2 \rfloor})$, $I_2 = (\lceil n/2 \rceil, a_{\lceil n/2 \rceil}, \ldots, a_n)$.

Тоді задача комбінування загального розв'язку I полягатиме в злитті двох впорядкованих підпослідовностей. Для вирішення підзадач I_1 та I_2 можна використати рекурсивний розв'язок базової задачі. Нехай $n = 2^k$ для деякого k. Тоді після $\log_2 (n)$ поділів навпіл отримаємо підпослідовності значень довжиною 1 (зворотний крок рекурсії).

Якщо час, потрібний для операції злиття, буде пропорційний n, тоді часова оцінка роботи процедури **mersort** описуватиметься рекурсивним співвідношенням:

$$T(n) = \begin{cases} a, & n = 1 \\ 2T(n/2) + cn, \end{cases}$$

де a і c константи.

Припустимо, що n є ступенем двійки $n = 2^k$, тоді

$T(n) = 2(2T(n/4) + cn/2) + cn = 4T(n/4) + 2cn = \ldots = 2^k T(1) + kcn =$
$= an + cn \log n$.

Отже, якщо $2^k < n < 2^{k+1}$, тоді

$T(n) \le T(2^{k+1})$ і $T(n) = O(n \log n)$.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. У чому полягає основний принцип планування в просторі задач?
2. Дайте визначення І-АБО-графу.
3. Що таке елементарні задачі?
4. Що таке тупикові задачі?
5. Яким чином І-АБО-графи пов'язані з мережами виведення для продукційних систем? Наведіть власний приклад.
6. В якому випадку вершина є розв'язною? Дайте рекурсивне визначення.
7. В якому випадку вершина є нерозв'язною? Дайте рекурсивне визначення.
8. Охарактеризуйте метод "поділяй і пануй". Наведіть приклади його застосування для вирішення практичних задач.
9. Опишіть загальну схему побудови часових оцінок для методу "поділяй і пануй".

ЗАДАЧІ І ВПРАВИ

1. Побудуйте власний приклад І-АБО-графу для довільної задачі.
2. Як на основі планування в просторі задач можна автоматизувати вирішення проблеми символічного інтегрування (отримання первісних від аналітично заданих виразів на основі тотожних перетворень без числових розрахунків)? Одна з можливих методик наведена в [200].

Розділ 16

ЖАДІБНІ АЛГОРИТМИ

Скупий платить двічі.
Прислів'я

16.1. Основні поняття

Розглянемо ще раз постановку класичної задачі пошуку екстремумів скалярної функції $f(x)$ n-мірного векторного аргументу x за деяких обмежень. Цю задачу можна описати так:

$\min f(x), x \in D.$

Тут D — деяка підмножина n-мірного евклідового простору E_n. Тоді D називають допустимою множиною задачі пошуку екстремуму, а точки, які належать D, — її допустимими точками. Якщо на область зміни аргументу функції мінімізації накласти більше обмежень, тоді пошук екстремуму функції стає важчим. Якщо ж обмеження досить жорсткі, наприклад, ця множина складається з кількох точок, і ці точки легко знайти, тоді загальна задача зводиться до простого перебору кількох чисел.

Розглянемо інший приклад. Нехай потрібно знайти найкоротший шлях між двома точками, які з'єднані вузьким коридором — допустимою областю руху. Тоді оптимізаційна задача стає досить тривіальною: просто потрібно прямувати тільки визначеним коридором — довільний шлях у ньому буде практично оптимальним. Тому в цьому розділі ми розглянемо алгоритми, побудовані на схемі послідовного аналізу варіантів. Вони використовують процедури, які на основі побічних оцінок відкидають всі ті допустимі розв'язки, серед яких не може бути оптимального. З часом проходить поступове звуження множини конкурентних варіантів. Наприкінці залишається один або кілька, які можна безпосередньо порівняти. Суттєвим у цих алгоритмах є значне знання природи задач розв'язку.

Розглянемо один з таких оптимізаційних методів, який дістав назву "жадібний". На основі нього будуються покрокові алгоритми, які за один крок розглядають один вхід n-мірного вектора x. На кожному кроці робиться перевірка входження цього часткового розв'язку до оптимального розв'язку.

Опишемо процедуру абстрактного керування в таких алгоритмах [304]:

```
Procedure Greedy (n, A, solution:...)
/*в масиві A (n) знаходиться вектор X*/
begin
    solution := 0;
    for i := 1 to n do
    begin
        x := select (A);
```

```
        if Feasible (solution, x)
        then solution := union (solution, x)
    end;
end;
```

Функція **SELECT** вибирає один вхід з масиву A. **FEASIBLE** є булевою функцією, яка визначає, допустиме чи ні включення x до вектора розв'язку. **UNION** включає x до загального розв'язку.

Повернемося до постановки задачі оптимізації в загальному вигляді: потрібно знайти $x = (x_1, \ldots, x_n)$, для якого $f(x_1, \ldots, x_n) = \min$ і задовольняються обмеження

$g_i(x_1, \ldots, x_n) = 0;\ i = \overline{1,m};$

$x_j \in D_j;\ j = \overline{1,n}.$

Нагадаємо, що будь-який x, за якого $x_j \in D_j$, називається *варіантом* розв'язку, а будь-який x, за якого задовольняються обмеження, називається допустимим розв'язком. Допустимий розв'язок, на якому $f(x)$ досягає мінімуму, називається *оптимальним розв'язком*. Якщо обмежень немає, мають на увазі *безумовну оптимізацію*. Ми вже також розглядали часткові випадки загальної задачі оптимізації: задачу лінійного програмування, задачу цілочислового програмування, задачу булевої оптимізації.

Одним з базових методів оптимізації є *метод послідовних локальних поліпшень*, який полягає в побудові послідовності $x^{(0)}, x^{(1)}, \ldots, x^{(n)}$, такої що $x^{(k)} \in D$ для всіх k; $x^{(k+1)} = h_k(x^{(k)})$, де h_k — деяка наперед задана функція, яка, в принципі, може залежати від k, і $f(x^{(k+1)}) < f(x^{(k)})$.

Алгоритми на основі методу послідовних локальних поліпшень можна назвати "жадібними" алгоритмами в широкому розумінні, оскільки вони на кожному кроці намагаються наблизитися до мети. "Жадібним" алгоритмом у вужчому розумінні називається алгоритм, який намагається на кожному кроці підійти до мети якнайближче.

16.2. Градієнтний метод

Як один з класичних прикладів можна навести *градієнтний метод* [13, 116, 164], який застосовується, якщо $D = R^n$, а цільова функція — диференційована. Градієнтом функції $f(x_1, \ldots, x_n)$ у точці x називається вектор:

$$\nabla f(x^*) = \left(\frac{\partial f(x^*)}{\partial x_1}, \ldots, \frac{\partial f(x^*)}{\partial x_n} \right).$$

Слід нагадати, що градієнт задає напрямок найшвидшого зростання функції. Тоді у градієнтному методі $x^{(k+1)} = x^{(k)} - \lambda_k \nabla f(x_k)$, де λ_k — величина кроку в напрямку антиградієнта. Якщо λ_k вибирається з умови $\lambda_k = \min_{\lambda > 0} (f(x^{(k)}) - \lambda \nabla f(x^{(k)}))$, алгоритм називається *методом найшвидшого спуску* [13, 116, 164].

Основним недоліком градієнтних методів (як і методів локальних послідовних поліпшень) є те, що вони дозволяють досягти не глобального екстремуму, а лише локального. Але існує важлива теорема, яка визначає достатні умови того, що локальний мінімум є водночас і глобальним [13, 116, 164].

Теорема 16.1. *Якщо $f(x)$ — опукла функція, визначена на опуклій множині X, то локальний мінімум є водночас і глобальним мінімумом.*

Цей результат зумовлює доцільність окремого розгляду задач опуклого програмування.

16.3. Жадібні алгоритми для задачі цілочислового програмування

Для випадку цілочислового програмування можна навести таке загальне визначення жадібного алгоритму.

Жадібним *називатимемо алгоритм, який на кожному кроці переходить від $x^{(k)}$ до $x^{(k+1)}$, причому $x^{(k+1)}$ вибирається з множини $P = \{x: x = h(x^{(k)})\}$, виходячи з умови оптимізації або самої цільової функції, або деякої евристичної функції и $(x^{(k)}, x^{(k+1)})$.*

Як приклад можна навести задачу про рюкзак, розглянуту в п. 12.2. Нехай маємо два типи предметів:

Вартість P_i	100	60
Вага W_i	100	90

Максимальна вага, що може бути розміщена в рюкзаку, $M = 180$. Потрібно розв'язати задачу:

$$\sum_{i=1}^{n} P_i X_i \to \max$$

$$\sum_{i=1}^{n} W_i X_i \le M$$

X_i — цілі; $X_i \ge 0$.

Розглянемо два варіанти застосування жадібного алгоритму.

Варіант 1.

$x^{(0)} = (0, 0)$. Сусідами $x^{(0)}$ є точки (0, 1), (1, 0), (1, 1). Остання точка взагалі не належить допустимій множині. Маємо: $f(0, 1) = 60$; $f(1, 0) = 100$. Тому $x^{(1)} = (1, 0)$. Поліпшити цей розв'язок неможливо, але він не є оптимальним, оскільки оптимум досягається в точці (0, 3).

Варіант 2.

Перехід від $x^{(k)}$ до $x^{(k+1)}$ здійснюється шляхом додавання одного предмета. Предмети упорядковуються за співвідношенням $\frac{P_i}{W_i}$, і додається предмет з множини тих предметів, які ще можна додавати, для якого це співвідношення найкраще. Знову ситуація аналогічна.

Уточнимо проведені над задачею роздуми.

Нехай є n предметів і рюкзак. Предмет i має вагову характеристику w_i, а рюкзак має обсяг M. Якщо візьмемо x_i, $0 \le x_i \le 1$, тоді розміщення предмета i в рюкзаку буде визначати зиск $p_i x_i$. Оптимальним наповненням рюкзака вважатимемо таке наповнення рюкзака предметами, яке максимізує загальний зиск. Оскільки вміст рюкзака обмежений M, вимагатимемо, щоб загальна вага предметів була не більшою за M. Формально, задача може бути сформульована таким чином:

максимізувати $\sum_{i=1}^{n} P_i X_i$ таких предметів i, для яких $\sum_{i=1}^{n} W_i X_i \le M$ та $0 \le x_i \le 1$, $p_i > 0$, $w_i > 0$, $i = \overline{1,n}$.

У випадку, коли $\sum_{i=1}^{n} W_i \le M$, для всіх x_i візьмемо $x_i = 1$, $i = \overline{1,n}$, яке й визначатиме оптимальний розв'язок. Тому розглянемо випадок, коли сума ваг всіх предметів більша за M.

З розглянутих стратегій розв'язання нашої задачі зупинимось на такій. На черговому кроці включатимемо в розв'язок той предмет, який має максимальний зиск за рахунок використаної частини обсягу рюкзака. Це означає, що предмети розглядатимуться стосовно відношення p_i / w_i. Зазначимо, що часова оцінка алгоритму цілком залежить від часової оцінки використаного алгоритму впорядкування, оскільки для реалізації самої стратегії наповнення після впорядкування предметів потрібен час $O(n)$.

Ще одним прикладом вдалого застосування техніки жадібного методу в цілочисловому варіанті може служити вдалий розв'язок задачі оптимального збереження програм у пам'яті. Один з варіантів її постановки може бути таким.

Потрібно n програм розмістити в пам'яті довжиною L. Нехай довжина кожної програми буде l_i, $i = \overline{1,n}$. Зрозуміло, що всі програми можуть бути збережені тільки тоді, коли $\sum_{i=1}^{n} l_i \le L$. Припустимо, що після чергового звертання до якоїсь з програм читаючий пристрій повертається на початкову позицію. Якщо програми зберігаються у порядку $I = i_1, i_2, i_3, \ldots, i_n$, тоді час t_j, необхідний для доступу до програми i_j, буде пропорційний $\sum_{k=1}^{j} l_{jk}$. Якщо всі програми викликаються рівномірно, то очікуваний час доступу (retrival time) **MRT** буде $\frac{1}{n}\sum_{j=1}^{n} t_j$. Тоді проблема оптимального збереження програм формулюється так. Потрібно знайти таке розміщення програм у пам'яті, щоб час доступу до них **MRT** був мінімальний. Мінімізація **MRT** є еквівалентною до мінімізації

$$D(I) = \sum_{j=1}^{n} \sum_{k=1}^{j} l_{jk}.$$

Приклад 16.1. Нехай $n = 3$, а $l_1 = 6$, $l_2 = 8$, $l_3 = 4$. Тоді ці три програми ми можемо розмістити в пам'яті $3! = 6$ способами. Порахуємо для кожного з варіантів розміщення $D(I)$:

Порядок I	Значення $D(I)$
1, 2, 3	$6 + 6 + 8 + 6 + 8 + 4 = 38$
1, 3, 2	$6 + 6 + 4 + 6 + 4 + 8 = 34$
2, 1, 3	$8 + 8 + 6 + 8 + 6 + 4 = 40$
2, 3, 1	$8 + 8 + 4 + 8 + 4 + 6 = 38$
3, 1, 2	$4 + 4 + 6 + 4 + 6 + 8 = 32$
3, 2, 1	$4 + 4 + 8 + 4 + 8 + 6 = 34$

Тому оптимальне розміщення буде 3, 1, 2.

Проаналізуємо цю задачу відповідно до загальної схеми "жадібного" алгоритму. Будемо на кожному кроці вибирати один розв'язок (одну роботу) та аналізувати поведінку функції $D(I)$. Помічаємо, що зростання $D(I)$ мінімізується, якщо чергова програма вибору має найменшу довжину серед програм, які залишилися.

Близько до задачі оптимального збереження програм у пам'яті стоїть задача оптимізації з'єднань файлів. Зрозуміло, що два впорядковані файли, відповідно довжини n та m записів, можуть бути перетворені на один впорядкований файл за час $O(n + m)$. Якщо ми маємо більше початкових впорядкованих файлів, тоді кінцевий файл можна отримати шляхом поетапного попарного використання попереднього підходу. Існує багато послідовностей попарного з'єднання файлів. Різні послідовності попарного поєднання вимагатимуть різної кількості комп'ютерного часу. Перед дослідниками виникає проблема оптимізації послідовності попарних з'єднань n файлів разом.

З нескладних роздумів можна сформулювати ідею обмежень "жадібного" алгоритму, які дозволять отримати оптимальний розв'язок. Критерій відбору буде такий: на черговому кроці об'єднання з'єднуватимуться два найменших за розміром файли. Наприклад, якщо ми маємо набір з 7 файлів ($F_1 = 10$, $F_2 = 5$, $F_3 = 11$, $F_4 = 20$, $F_5 = 4$, $F_6 = 30$, $F_7 = 18$), тоді згідно із запропонованим критерієм вибору буде така послідовність об'єднань: $F_2 || F_5 = Z_1$ (9), $Z_1 || F_1 = Z_2$ (19), $F_7 || Z_2 = Z_3$ (37), $F_4 || F_6 = Z_4$ (50), $Z_3 || Z_4 = Z_5$ (87). Легко перевірити, що це — оптимальне вирішення нашої проблеми.

На кожному кроці модель обчислень вимагає порівняння двох об'єктів. Тому графічно така модель зручно описується бінарним деревом. Листки такого дерева (квадрати) визначають набір початкових файлів впорядкування. Внутрішні вузли (кружки) визначають файли з'єднання. Їх сини — це файли, які беруть участь у черговому з'єднанні. Усередині вузлів стоять числа. Вони характеризують довжину відповідних файлів. Дерево, зображене на рис. 16.1, описує модель з'єднання файлів для останнього прикладу.

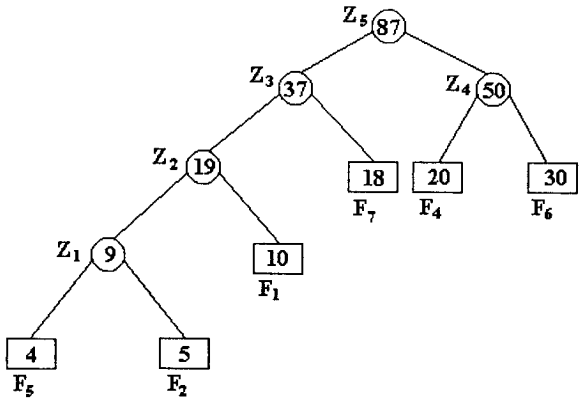


Рис. 16.1. Задання моделі поєднання бінарним деревом

Зовнішній вузол F_5 знаходиться на відстані 4 від кореня (вузол рівня i перебуватиме на відстані $i - 1$ від кореня). Отже, записи файла F_5 будуть перекинуті 4 рази для отримання остаточного з'єднання Z_5. Для файла F_i введемо відповідні позначення: d_i — відстань від кореня до вузла F_i та q_i — довжина файла F_i. Тоді загальне число пересувань записів для бінарного дерева поєднання визначатиметься формулою $\sum_{i=1}^{n} d_i q_i$, де n — кількість початкових файлів з'єднання. Ця сума називається *зваженою довжиною* зовнішнього шляху дерева.

Оптимальна модель поєднання файлів відповідатиме бінарному дереву поєднання, яке буде мати мінімальну зважену довжину зовнішнього шляху.

"Жадібний" метод генеруватиме дерева злиття і в випадку k-арного злиття. Тут деревом злиття буде дерево, кожен вузол якого може мати k синів. У даному разі правило "жадібності" для генерування оптимального дерева злиття буде модифікуватись так: на кожному кроці вибрати k піддерев з найменшими довжинами для злиття.

Мінімальна зважена довжина зовнішнього шляху дерева може використовуватись для отримання оптимальної множини кодів повідомлень $M_1, M_2, \ldots, M_{n+1}$. Кожний код буде двійковим рядком, який використовується при передачі відповідного повідомлення. Для декодування повідомлення застосовуватиметься дерево декодування, що є бінарним деревом, в якому зовнішні вузли містять інформаційні частини повідомлення.

Дерево будується аналогічно побудові дерева з'єднань файлів, тільки замість довжини файла з'єднань застосовується частота використання повідомлення. Двійкові біти слова кодування відповідного повідомлення визначають потрібну гілку кожного рівня, досягнувши вірно зовнішнього вузла. Розглянемо дерево, яке зображено на рис. 16.2.

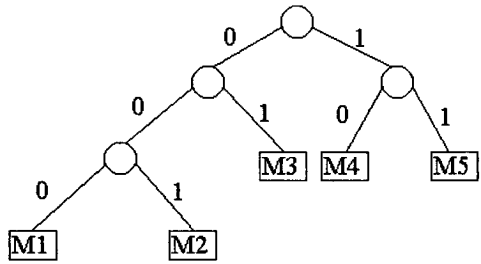


Рис. 16.2. Приклад дерева кодування

Тоді **000** кодуватиме повідомлення **M1, 001 — M2, 01 — M3, 10 — M4, 11 — M5**. Такі коди називають кодами Хаффмана. Ціна декодування слова кодування прямо пропорційна кількості бітів у коді. Це число є еквівалентним до довжини шляху між коренем і відповідним зовнішнім вузлом. Якщо розглядати q_i як відносну частоту передачі повідомлення **M1**, тоді очікуваний час розкодування визначатиметься $\sum_{1 \le i \le n+1} q_i d_i$, де d_i буде довжиною шляху між коренем і зовнішнім вузлом, який містить повідомлення M_i. Очікуваний час розкодування буде мінімальним для обраного слова кодування, якщо дерево декодування матиме мінімальну зважену довжину зовнішнього шляху. Тому для мінімізації очікуваного часу декодування потрібно мінімізувати очікувану довжину повідомлення.

16.4. Матроїди

При використанні жадібних алгоритмів актуальним є питання: коли вигідно бути жадібним? Відповідь на це запитання дає теорія матроїдів.

Розглянемо оптимізаційні задачі такого типу.

Нехай задані скінченна множина E, набір її підмножин $I \subseteq P(E)$; функція w: $E \to R^+$ позначає множину дійсних невід'ємних чисел. Потрібно знайти підмножину $S \in I$ з найбільшою сумою $\sum_{e \in S} w(e)$.

Тоді запитання про корисність використання жадібності можна перефразувати так: за яких умов відносно набору I жадібний алгоритм вірно вирішує поставлену задачу для довільної функції w? На це запитання відповідь відома: достатньо, щоб пара $<E, I>$ була матроїдом [16, 165].

Матроїди були вперше введені Х. Уїтні у праці [330] для дослідження абстрактної теорії лінійної залежності. Використовуватимемо таке визначення матроїда [165].

***Матроїдом** називають довільну пару $M = <E, I>$, де E — скінченна множина, а $I \subseteq P(E)$ — набір, що задовольняє умовам:*

1) *$\varnothing \in I$, і якщо $A \in I$ та $B \subseteq A$, тоді $B \in I$;*

2) *Для довільних $A, B \in I$, таких що $|B| = |A| + 1$, існує елемент $e \in B \backslash A$, такий що $A \cup \{e\} \in I$.*

(Умова $\varnothing \in I$ виключає вироджений випадок $I = \varnothing$).

Множини набору I називають незалежними множинами, наголошуючи на зв'язок теорії матроїдів з теорією лінійної залежності, а інші підмножини із $P(E) \setminus I$ — залежними множинами матроїда M.

Кажуть, що два матроїди $<E, I>$ і $<E', I'>$ ізоморфні, якщо існує взаємно однозначне відображення f множини E на множину E', таке що $A \in I$ тоді і тільки тоді, коли $f(A) \in I'$.

Для визначеності матроїда часто використовується теорема 16.2 [165].

Теорема 16.2. *Нехай E — скінченна множина, I — набір її підмножин, які задовольняють умові* 1). *$M = <E, I>$ буде матроїдом тоді і тільки тоді, коли виконується умова:*

3) *Для довільної підмножини $C \subseteq E$ кожні дві максимальні підмножини множини C мають одну потужність.*

Еквівалентну систему аксіом для матроїдів утворюють умови (1)—(2) і (3).

Потужність максимальної підмножини множини $C \in E$ називають рангом множини C і позначають $r(C)$:

$$r(C) = \max \{|A| : A \in I, A \subseteq C\}.$$

Зрозуміло, що підмножина $C \subseteq E$ є незалежною тоді і тільки тоді, коли $r(C) = |C|$. Кожна максимально незалежна множина матроїда $M = <E, I>$ називається базою цього матроїда, а ранг $r(E)$ — рангом матроїда.

Повернемося до оптимізаційної задачі і сформулюємо загальну схему жадібних алгоритмів у термінах теорії матроїдів.

Нехай маємо скінченну множину E, функцію $w: E \to R^+$ і набір $I \subseteq P(E)$. Значення $w(e)$ називають вагою елемента e. Вага підмножини $A \subseteq E$ визначається так:

$$w(A) = \sum_{e \in A} w(e).$$

Тоді загальну схему жадібних алгоритмів можна описати так [165]:

begin

впорядкувати множину E за спаданням ваги так, щоб

$\mathbf{E} = \{e_1, e_2, \ldots e_n\}$, де $w(e_1) \geq w(e_2) \geq \ldots \geq w(e_n)$;

S:= $\varnothing$**;**

for i := 1 **to** n **do**

if $S \cup \{e_i\} \in I$ **then** $S := S \cup \{e_i\}$

end;

Характеристику множин результату X дає теорема Радо-Едмондса (теорема 16.3) [294].

Теорема 16.3. *Якщо $M = <E, I>$ матроїд, тоді знайдена жадібним алгоритмом множина S є незалежною множиною з найбільшою вагою. В іншому разі знайдеться така функція $w: E \to R^+$, що S не буде незалежною множиною з найбільшою вагою.*

Розглянемо матрицю

$$A = \begin{pmatrix} a_{11} & a_{12} & \dots & a_{1n} \\ a_{21} & a_{22} & \dots & a_{2n} \\ \dots & \dots & \dots & \dots \\ a_{n1} & a_{n2} & \dots & a_{nn} \end{pmatrix}$$

Кожна така матриця визначає матроїд $M(A) = <E, I>$, де E — множина її стовпців, а $B \in I$ тоді і тільки тоді, коли множина стовпців B лінійно незалежна. Цей матроїд називається матроїдом матриці A. Матроїд називається ***матричним***, якщо він є (ізоморфним з) матроїдом деякої матриці.

У випадку матричного матроїда загальна схема жадібного відбору перетворюється на метод виключення Гауса для матриці A розмірності $m \times n$, стовпці якої впорядковані за спаданням ваг (ваги невід'ємні). Жадібний алгоритм для матричного матроїда [165] зображений на рис. 16.3.

Вхід. Матриця A розмірності $m \times n$, стовпці якої впорядковані за неспаданням ваг (ваги невід'ємні).

Вихід. Незалежна множина стовпців з найбільшою сумою ваг. Номери стовпців знаходяться в S.

```
begin
    S := ∅
    for j := 1 to n do
    begin
        i := 0;
        while (A [i, j] = ∅) and (i < m) do i := i + 1;
        if a [i, j] <> 0 then (* j-й стовпець ненульовий*)
        begin
            S := S ∪ {j};
            for k := j = 1 to n do
            for l := 1 to m do
            A [l, k] := A [l, k] – A [l, j] *A [i, k] / A [i, j]
        end (* A [l, k] = 0 для n ≥ m ≥ j + 1*)
    end
end;
```

Рис. 16.3. Жадібний алгоритм для матричного матроїда

Неважко довести, що числова оцінка алгоритму становить $O(n^{2m})$. Він може бути застосований для вирішення задачі, пов'язаної з плануванням експерименту, в якому об'єкт експерименту складається з певних компонент. Задачею дослідження буде задача вияву окремих компонент зміни об'єкта. Є можливість кількісно виміряти сумарну зміну об'єкта, яка має лінійний вигляд:

$$b = c_1x_1 + c_2x_2 + \dots + c_mx_m,$$

де x_i є інтенсивністю i-го фактора, а $c_1, \ldots, c_m$ — коефіцієнти, які можна визначити.

Для неорієнтованого графу $G = <V, E>$ матроїд визначається так:

$M(G) = <E, I>$,

де $I = \{A \subseteq E$: *граф* $<V, A>$ *— ациклічний*$\}$.

Матроїд $M(G)$ називається матроїдом графу G. Довільний ***матроїд*** називається ***графовим***, якщо він є матроїдом деякого графу.

Враховуючи, що граф можна задати матрицею інцидентності, кожний графовий матроїд можна трактувати як матричний. Останній відповідає матриці інцидентності графу G, елементи якої приймають значення з поля $Z^2 = \{0, 1\}$. Додавання в такому полі виконується за модулем 2, а множення — звичайне множення цілих чисел.

Відома теорема [165].

Теорема 16.4. *Нехай $G = <V, E>$ — звичайний граф, і A — його матриця інцидентності. Підмножина стовпців матриці A є лінійно залежною над полем Z^2 тоді і тільки тоді, коли відповідна їй підмножина ребер містить цикл.*

Тому для пошуку бази матроїда $M(G)$ з найменшою вагою (мінімального дерева стягування, в цілому — графу, всі компоненти зв'язності якого є деревами) можна використати жадібний алгоритм для матроїдів. Отже, за умови, що $u = |V|$ і $m = |E|$, таку базу матроїда можна знайти за $O(n^{2m})$.

16.5. Алгоритми Пріма і Крускала

Проте відомі [304] алгоритми Пріма і Крускала є кращими для вирішення цієї задачі. Останній посідає значне місце в теорії матроїдів. Жадібний алгоритм для матроїдів був сформульований як узагальнення алгоритму Крускала з матроїда графу на довільний матроїд.

Розглянемо основні ідеї цих алгоритмів.

Нехай $G = <V, E>$ — зв'язний неорієнтований граф. Для нього задана функція вартості, яка відображає ребра в дійсні числа. Остовим деревом для графу $G = <V, E>$ є неорієнтоване дерево $T = (V, E')$, яке містить усі вершини графу G. Вартість остового дерева визначається як сума вартостей його ребер. Спробуємо знайти для G остове дерево мінімальної вартості. Основні алгоритми знаходження остових дерев базуються на наступних двох лемах [15].

Лема 16.1. *Нехай $G = <V, E>$ — зв'язний неорієнтований граф і $T = (V, E')$ — його остове дерево. Тоді*

1) для довільних двох вузлів v_1 і v_2 із V шлях між ними в T єдиний і

2) якщо до T додати довільне ребро із $E \setminus E'$, тоді виникне рівно один цикл.

Лема 16.2. *Нехай $G = <V, E>$ — зв'язний неорієнтований граф і c — функція вартості, визначена на його ребрах. Нехай $\{(V_1, E'_1), (V_2, E'_2), \ldots, (V_k, E'_k)\}$ — довільний остовий ліс для G, $k > 1$ і $E' = \bigcup_{i=1}^{k} E'_i$. Припустимо, що $e = (v, w)$ — ребро найменшої вартості в $E \setminus E'$, таке, що $v \in V_1$ і $w \notin V_1$. Тоді знайдеться остове дерево для G, яке містить $E' \cup \{e\}$, вартість якого не більша від вартості довільного остового дерева для G, яке містить E'.*

Застосуємо для побудови остового дерева мінімальної вартості "жадібний алгоритм". Він будуватиме остове дерево крок за кроком, додаючи до нього по одному ребру згідно з певним оптимізаційним критерієм. Один з найпростіших критеріїв такий. На черговому кроці вибираємо ребро, яке приводить до мінімального збільшення суми вартостей ребер, які вже включені в допустимий розв'язок. Існує два варіанти інтерпретації цього критерію.

Варіант 1. Нехай A — множина ребер, вибраних на попередньому кроці, і A має форму дерева. Тоді чергове ребро під'єднання (u, v) є ребром з мінімальною вартістю, якого ще немає в A і приєднання якого до A залишає $A \cup \{(u, v)\}$ деревом. Відповідний алгоритм називають алгоритмом Пріма. Для графу з рис. 16.4 роботу алгоритму Пріма описує рис. 16.5.

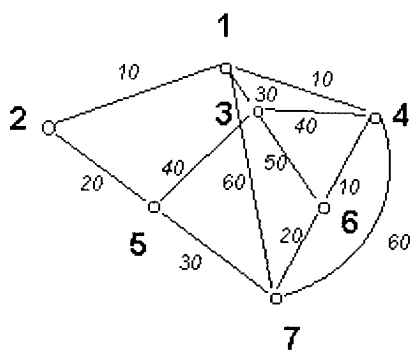


Рис. 16.4. Граф, який ілюструє роботу алгоритма Пріма

Спробуємо формалізувати алгоритм Пріма. Побудову дерева починаємо з розгляду ребер, які мають мінімальну вартість. Вибираємо одне з них, наприклад, відповідно до лексикографічного порядку кортежів відносно (i, j). Далі приєднуємо до остового дерева T таке ребро (l, k), в якого $l \in T$, а $k \notin T$ і вартість **cost** (l, k) є мінімальною серед тих ребер, які задовольняють першій частині умови. До того ж, приєднання цього ребра повинно залишити T деревом, конкретніше, дерево T має залишитись зв'язним, і не повинен з'явитись цикл. Процес побудови закінчиться, коли T матиме всі вершини із G.

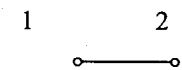
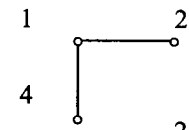
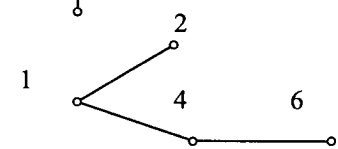
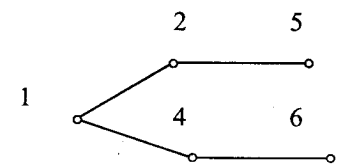
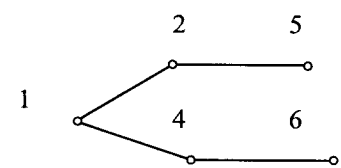
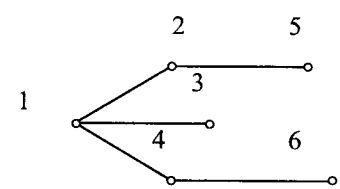
Кандидати	Ребра	Вартість	Остове дерево
(1, 2)	(1, 2)	10	
(2, 5) (1, 4)	(1, 4)	10	
(2, 5) (4, 6)	(4, 6)	10	
(2, 5) (6, 7)	(2, 5)	20	
(6, 7)	(6, 7)		
(1, 3) (5, 7)	(1, 3)	30	

Рис. 16.5. Остове дерево для графу з рис. 16.4

Очевидно, що часова складність алгоритму визначається тим фрагментом, де відбувається початкове заповнення (ініціалізація) матриці задання остового дерева.

Варіант 2. Звернувшись уважно до леми 16.2, можемо отримати другу інтерпретацію оптимізаційного критерію, яка дістала назву алгоритму Крускала знаходження остового дерева мінімальної вартості графу.

Введемо до розгляду ліс остових дерев, який описується набором VS неперетних множин вузлів, що перетинаються. Кожна множина W із VS задаватиме зв'язну множину вузлів остового дерева в остовому лісі. Спочатку $VS = \{W_i,\ i = 1..n\}$, де $W_i = v_i,\ i = 1..n$. Ребра вибираються в порядку зростання вартості. Нехай на черговому кроці вибрали ребро $(v, w) \in E$.

Якщо ребро (v, w) належить тільки одному дереву в остовому лісі дерев VS, тоді це ребро видаляється з пріоритетної черги і з подальшого розгляду. В іншому разі, коли $v \in W_1$, а $w \in W_2$, $W_1 \in VS$, $W_2 \in VS$, тоді зливаємо два дерева W_1 і W_2 в одне і додаємо ребро (v, w) до кінцевого остового дерева T. Процес закінчується, коли ліс VS складатиметься лише з одного дерева. Цей алгоритм називають алгоритмом Крускала [304]. Структура цього алгоритму описується послідовністю інструкцій:

procedure Kruscal (V — множина вершин, E — множина ребер і C — вартість ребра)

begin

1. $T \leftarrow 0$.
2. $VS \leftarrow 0$.
3. Побудувати купу Q усіх ребер E згідно зі зростанням функції вартості c (e).
4. Для $v_i \in V$ створити множину $W_i = \{v_i\}$ в VS, $i = 1, 2, \ldots, n$.
5. Поки ліс остових дерев VS не є одним деревом робити

початок.

6. Взяти ребро (v, u) із голови купи Q.
7. Видалити (v, u) із купи і поновити в Q властивість купи.
8. Якщо в лісі VS знайдуться такі два дерева W_i і W_j, що $v \in W_i$, а $u \in W_j$, тоді зробити:

початок.

9. У лісі VS замінити множини W_i і W_j на $W_i \cup W_j$
10. $T \leftarrow (v, u)$

кінець

кінець

end;

Розглянемо роботу алгоритму Крускала на графі, зображеному на рис. 16.6.

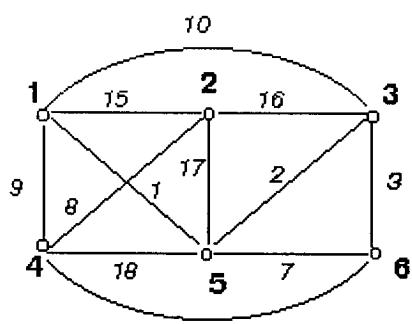


Рис. 16.6. Неорієнтований граф

Таблиця 16.1 ілюструє послідовність дій, які приводять до побудови остового дерева для цього графу.

Таблиця 16.1

Послідовність кроків побудови остового дерева (рис. 16.7) для графу з рис. 16.6 згідно з алгоритмом Крускала

Голова купи Q	Остовий ліс дерев VS після дії пунктів 8—9	Дія пунктів 8—9	Результуюче дерево T
пуста	ще не дійшла справа	ще не дійшла справа	0
(V1, V5)	{V1, V5}	додати	{V1, V5}, {V2}, {V3}, {V4}, {V6}
(V3, V5)	{(V1, V5) (V3, V5)}	додати	{V1, V3, V5} {V2}, {V4} {V6}
(V3, V6)	{(V1, V5) (V3, V5) (V3, V6)}	додати	{V1, V3, V5, V6} {V2} {V4}
(V5, V6)	{(V1, V5) (V3, V5) (V3, V6)}	залишити	{V1, V3, V5, V6} {V2} {V4}
(V2, V4)	{(V1, V5) (V3, V5) (V3, V6), (V2, V4)}	додати	{V1, V3, V5, V6}, {V2, V4}
(V1, V4)	{(V1, V5) (V3, V5) (V3, V6) (V2, V4) (V1V4)}	додати	{V1, V2, V3, V4, V5, V6}

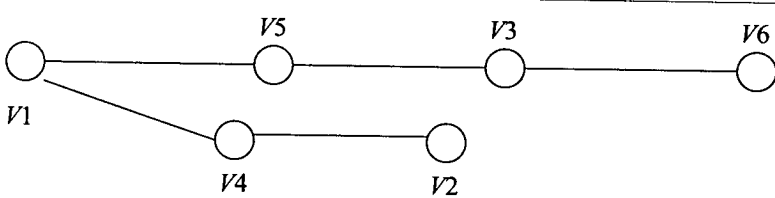


Рис. 16.7. Остове дерево мінімальної вартості

Можна довести [304], що алгоритм Крускала знаходить остове дерево найменшої вартості для зв'язного графу G за час $O(e \log e)$, де $e = |E|$.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте визначення жадібного алгоритму.
2. Поясніть, яким чином застосування жадібних алгоритмів приводить до скорочення перебору при розв'язанні інтелектуальних задач.
3. Сформулюйте основну ідею методу послідовних поліпшень.
4. Охарактеризуйте градієнтний метод як різновид жадібного алгоритму.
5. Чи завжди градієнтний метод дозволяє отримати:
   а) глобальний екстремум;
   б) хоча б один з екстремумів?
6. Сформулюйте умови, за яких градієнтний метод приводить до оптимального рішення.

7. Що таке опукле програмування?
8. Опишіть алгоритм Дейкстри для пошуку найкоротшого шляху на графі.
9. Охарактеризуйте клас задач цілочислового програмування, для яких жадібний алгоритм дозволяє отримати оптимальний розв'язок.
10. Що таке матроїд?
11. Уточніть алгоритми Пріма і Крускала.

Розділ 17

ДИНАМІЧНЕ ПРОГРАМУВАННЯ

Вирішуватимемо проблеми у
міру їх надходження ...
Афоризм

17.1. Основні поняття

У цьому розділі розглянемо метод розв'язку задач, побудований на схемі послідовного аналізу варіантів — з використанням процедур, які на основі побічних оцінок відкидають всі ті допустимі розв'язки, серед яких не може бути оптимального. З часом проходить поступове стиснення множини конкурентоспроможних варіантів і нарешті залишається один або два варіанти, які й порівнюються.

Перетворити цей загальний підхід на систему формальних процедур дуже важко. Але вже багато зроблено в цьому напрямі. Перші ідеї такого підходу належать А. Маркову [186], розвивались вони і у працях американців Д. Вальда, Р. Айзекса, Р. Беллмана [20]. Значний внесок у створення загального формалізму послідовного аналізу варіантів (схема формалізації Михалевича, метод "київський вінник") належить київській школі математиків на чолі з В. Михалевичем [183—186].

Методи динамічної оптимізації значною мірою використовують природу задач розв'язку. Тому ми тут більше звертатимемо уваги не на загальну методологію підходу, а обмежимося розглядом кількох задач, які досить повно ілюструють загальні концепції підходу.

Динамічне програмування *— це такий алгоритмічний метод, який можна використовувати, коли вирішення задачі можна задати як результат певної послідовності розв'язків.*

У попередніх розділах було описано чимало таких задач. Дійсно, вирішуючи задачу про рюкзак, її можна розглядати і так. Для знаходження значення x_i, $1 \le i \le n$ потрібно знайти розв'язок x_1, потім x_2, потім x_3 і т. д. Оптимальна послідовність цих розв'язків повинна максимізувати цільову функцію $\Sigma p_i x_i$, та задовольняти умовам $\Sigma w_i x_i \le M$ і $x_i \in 0, 1, 2, \ldots$. У разі вирішення задачі оптимального з'єднання файлів визначається, які саме

пари файлів необхідно з'єднати першими, які другими, третіми і т. д. Оптимальною послідовністю розв'язків буде послідовність з найменшою вартістю.

Для деяких задач, які можна розглядати в такій інтерпретації, оптимальну послідовність рішень можна знайти, приймаючи на кожному кроці вірне рішення (наприклад, задачі, що можна вирішити жадібним методом). Для багатьох інших задач прийняти таке рішення неможливо.

17.2. Принцип оптимальності Беллмана

Зрозуміло, що одним із способів розв'язку задач, для яких неможливо на кожному кроці приймати рішення, що приводять до оптимальної послідовності, полягає у перевірці всіх можливих послідовностей розв'язків. Динамічне програмування значно зменшує кількість варіантів, які треба перевірити, за рахунок виключення тих послідовностей рішень, які не зможуть бути оптимальними. В динамічному програмуванні оптимальна послідовність рішень досягається за рахунок явного звернення до ***принципу оптимальності***.

Цей принцип стверджує: *оптимальна послідовність рішень має таку властивість, що, незважаючи на початковий стан і рішення, серед рішень, які залишились, завжди міститься оптимальна послідовність рішень стосовно стану, що утворився після прийняття першого рішення.* Отже, суттєва різниця між жадібним методом і динамічним програмуванням полягає в тому, що жадібний метод завжди генерує одну послідовність рішень. У динамічному програмуванні може утворюватись кілька послідовностей рішень.

Повернемось до розглянутої вже нами задачі знаходження найкоротшого шляху. Припустимо, що $i, i_1, i_2, \ldots, i_k$ — найкоротший шлях від i до j. У вершині i було прийнято рішення відвідати вершину i_1. Після цього стан задачі визначатиметься вершиною i_1 та необхідністю знайти шлях від i_1 до j. Цілком зрозуміло, що послідовність $i_1, i_2, \ldots, i_k, j$ повинна містити найкоротший шлях від i_1 до j. Якщо це не так, тоді нехай $i_1, r_1, r_2, \ldots, r_q, j$ буде найкоротшим шляхом від i_1 до j. Тоді шлях $i, i_1, r_1, \ldots, r_q, j$ буде коротшим за шлях $i, i_1, i_2, \ldots, i_k, j$. Тому принцип оптимальності можна застосувати і до цієї задачі.

Аналізуючи хід розв'язку цієї задачі, можна виокремити такі загальні риси методу. Нехай S_0 — початковий стан задачі. Припустимо, що треба прийняти n рішень d_i, $1 \le i \le n$. Нехай $D_1 = \{r_1, r_2, \ldots, r_j\}$ — це множина можливих значень для роз'язку d_1. Нехай S_i — це стан задачі, який утворюється після вибору рішення r_i, $1 \le i \le j$. Нехай Γ_i буде оптимальною послідовністю рішень відповідно до стану задачі S_i. Тоді згідно з принципом оптимальності оптимальна послідовність рішень відповідно до S_0 визначається як найкраща послідовність рішень $r_i\Gamma_i$, $1 \le i \le j$.

Так, у застосуванні до задачі про найкоротший шлях ця стратегія набуде такого вигляду: нехай A_i — множина вершин, суміжних з вершиною i.

Для кожної вершини $k \in A_i$ нехай Γ_k буде найкоротшим шляхом від k до j. Тоді найкоротший шлях від i до j визначається як найкоротший шлях з множини $\{i, \Gamma_k | k \in A_i\}$.

17.3. Рекурентні рівняння в динамічному програмуванні

Для більшості задач рекурсивна природа задання принципу оптимальності приводить до рекурентного типу відношення. Алгоритми динамічного програмування розв'язують ці рекурентні відношення, щоб знайти оптимальний розв'язок заданої задачі (існує окрема теорія розв'язку рекурентних рівнянь [15]).

Формулювати рекурентні відношення динамічного програмування можна за допомогою використання двох підходів перегляду варіантів розв'язків: прямого і зворотного. Нехай $x_1, x_2, \ldots, x_n$ — це змінні, що характеризують шукану послідовність рішень. У прямому перегляді побудова розв'язку x_i формулюється в термінах оптимальної послідовності рішень для $x_{i+1}, \ldots, x_n$. Для зворотного перегляду формулювання тверджень відносно рішення x_i проводиться в термінах оптимальної послідовності рішень для $x_1, \ldots, x_{i-1}$. Відмітимо, що, якщо рекурентні відношення сформульовані, використовуючи прямий підхід, тоді рівняння *"вирішуються технікою перегляду назад"*, тобто починаючи з останнього рішення. З іншого боку, якщо відношення сформульовані, використовуючи зворотний підхід, вони *"вирішуються технікою перегляду вперед"*. Завдяки використанню принципу оптимальності послідовності рішень, що містять підпослідовності, які є неоптимальними, в більшості випадків *не розглядаються*. Тому алгоритми динамічного програмування, як правило, мають поліноміальну оцінку.

Інша важлива риса динамічного програмування полягає в тому, що оптимальні розв'язки підзадач зберігаються в такому вигляді, який запобігає перерахуванню їх значень при подальших використаннях цих підзадач. У більшості випадків для цього використовується табличне зберігання. Застосування табличних значень робить природним перетворення рекурсивних рівнянь на ітеративні програми. Більшість алгоритмів динамічного програмування, описані в цій частині, будуть представлені саме в такий спосіб.

За зворотного проходу побудови загального розв'язку із розв'язків підзадач меншої розмірності обчислення йде від менших підзадач до більших, і відповіді запам'ятовуються в таблиці. Перевага використання таблиці полягає в тому, що, тільки-но якась підзадача розв'язана, її відповідь заноситься в таблицю і ніколи вже не обчислюється знову.

17.4. Числа Фібоначчі

Розглянемо цю техніку на класичному прикладі побудови послідовності *чисел Фібоначчі*.

Для розв'язку багатьох задач використовується спеціальна послідовність чисел, яку ввів Фібоначчі. *N*-не число Фібоначчі рекурсивно визначається так:

$$F(N)=\begin{cases}1 & \text{якщо } N=0;\\ 1 & \text{якщо } N=1;\\ F(N-1)+F(N-2) & \text{якщо } N>1.\end{cases}$$

Потрібно написати програму, яка обчислює *N*-не число Фібоначчі. Цю задачу можна розв'язати двома способами: за допомогою рекурсії або за допомогою динамічного програмування.

Розглянемо перший варіант. Його реалізовує рекурсивна функція fib:

```
function fib (n: integer): integer;
begin
   if n = 0 then fib := 1
   else if n = 1 then fib := 1
   else fib := fib (n – 1) + fib (n – 2)
end;
```

Таке природне використання рекурсії не завжди буває оптимальним. Недолік полягає в тому, що кожного разу для обчислення більшого числа Фібоначчі ми змушені менші числа обчислювати по кілька разів. Наочніше цей факт представлено на рис. 17.1.

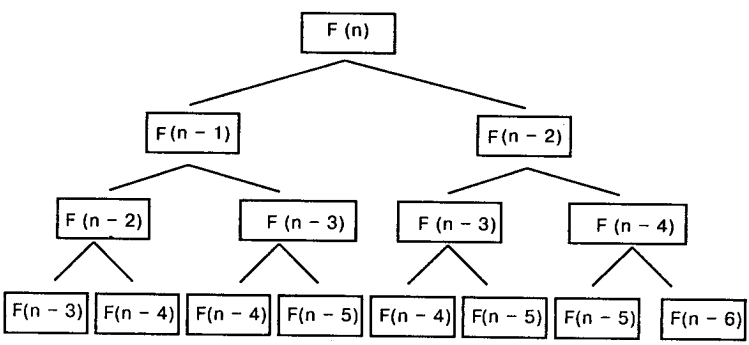


Рис. 17.1. Структура викликів процедури для обчислення *N*-го числа Фібоначчі

Розглянемо інший підхід. Почнемо обчислення з породження значень менших чисел Фібоначчі. Зберігатимемо два останні числа в таблиці, яка містить дві змінні. Наступне число Фібоначчі утворюється шляхом сумування цих двох змінних. Потім таблиця поновлюється і процес продовжується. В даному разі алгоритм буде обчислюватись швидше.

*Отже, під **динамічним програмуванням** можна розуміти такий спосіб побудови алгоритмів, який може бути застосований у разі, коли вирішення загальної задачі можна розбити на розв'язок її підзадач і отримувати розв'язок наступної задачі як суперпозицію попередніх розв'язків.*

17.5. Задача про критичний шлях

Розглянемо ациклічний зважений граф $G = <V, E>$, ваги ребер якого можуть бути довільними. Спробуємо вирішити для нього задачу з одним джерелом за час $O(n^2)$. Алгоритми такого типу використовуються в багатьох прикладних задачах і особливо інтенсивно в методах керування виконанням проекту, які називають PETR *(Project Evaluation and Review Technique)* або CPM *(Critical Path Method)*. Для цих методів [304] основою дослідження є побудований граф мережі PETR або CPM, дуги якого відповідають деяким елементарним задачам, що складають проект, а їх ваги вказують на час розв'язання окремої задачі. На графі визначено частковий порядок. Припускають, що для довільних дуг цього графу $<u, v>$ і $<v, t>$ задача, якій відповідає дуга $<v, t>$, повинна розпочатись тільки після закінчення роботи, відміченої дугою $<u, v>$. Зрозуміло, чому такий граф повинен бути безконтурним. Тоді розв'язком задачі є знаходження найдовшого шляху з вершини s, що відповідає початку проекту, до вершини t — вершини закінчення проекту. Цей шлях називають критичним. Його довжина визначає час, потрібний для реалізації всього проекту.

Багаторівневий граф $G = <V, E>$ — це орієнтований граф, вершини якого містяться в $k \geq 2$ множинах V_i, $1 \leq i \leq k$, що не перетинаються. Крім того, якщо $<u, v>$ — це ребро в E, тоді $u \in V_i$ і $v \in V_{i+1}$ для будь-якого i, $1 \leq i < k$. Множини V_i і V_k такі, що $|V_i| = |V_k| = 1$. Нехай s та t будуть відповідно вершинами з V_1 і V_k. При цьому s — вершина джерела, а t — вершина стоку. Нехай $c(i, j)$ — вартість ребра $<i, j>$. Вартість шляху від s до t визначається як сума вартостей ребер цього шляху. *Задача багаторівневого графу* полягає у знаходженні шляху від s до t мінімальної вартості [304]. Кожна множина V_i визначає рівень графу. Через обмеження, що накладені на множину E, кожний шлях від s до t починається з 1-го рівня, потім продовжується на 2-му рівні, потім на 3-му і т. д., поки не закінчується на рівні k.

Багато задач можна сформулювати як задачі на багаторівневих графах. Достатньо навести лише один приклад [304]. Розглянемо проблему розподілу ресурсів, в якій n одиниць ресурсу треба розподілити між r проектами. Якщо j $(0 \leq j \leq n)$ одиниць ресурсу віддати ресурсу i, тоді результуючий прибуток становитиме $N(i, j)$. Задача розподілу ресурсів полягає в тому, щоб поділити ресурс між r проектами таким чином, аби одержати максимальний загальний прибуток. Цю задачу можна сформулювати як задачу пошуку на $r + 1$-рівневому графі. Рівень i, $1 \leq i \leq r$, представляє проект i. $n + 1$ вершина $V(i, j)$, $0 \leq j \leq n$ асоціюється з рівнем i, $2 \leq i \leq r$. Рівні 1 та $r + 1$ мають по одній вершині: $V(1, 0) = s$ та $V(r + 1, n) = t$ відповідно. Вершина $V(i, j)$, $2 \leq i \leq r$, представляє рівень, на якому j одиниць ресурсів розподіляються між проектами 1, 2, ..., $i - 1$. Ребра в G мають форму $<V(i, j), V(i + 1, l)>$ для всіх $j \leq l$ та $1 \leq i < r$. Ребру $<V(i, j), V(i + 1, l)>$, $j \leq l$ приписується ваговий коефіцієнт $N(i, l - j)$ і кожне ребро відповідає за розподіл $l - j$ одиниць ресурсу проекту i, $1 \leq i < r$. Крім того, G має ребра типу

$<V(r,j), V(r+1,n)>$. Кожному такому ребру приписується максимальний ваговий коефіцієнт:

$$\max_{0 \le p \le n-j} \{N(r,p)\}.$$

Формулювання для задачі k-рівневого графу з точки зору динамічного програмування базується на спостереженні, що кожний шлях від s до t буде результатом послідовності $k-2$ рішень; i-те рішення впливає на знаходження тієї вершини з V_{i+1}, $1 \le i \le k-2$, яку необхідно включити в шлях. Легко побачити, що таким чином зберігається принцип оптимальності. Нехай $P(i,j)$ — це мінімальна вартість шляху з вершини j з V_i до вершини t. Нехай COST (i,j) — це вартість такого шляху. Тоді використовуючи підхід прямого перегляду, отримуємо:

$$\text{COST}(i,j) = \min_{l \in V_{i+1}, <j,l> \in E} \{c(j,l) + \text{COST}(i+1,l)\} \qquad (17.1)$$

Оскільки COST $(k-1,j) = c(j,t)$, якщо $<j,t> \in E$, і COST $(k-1,j) = \infty$, якщо $<j,t> \notin E$, (17.1) можна розв'язати для COST $(1,s)$, спочатку обрахувавши COST $(k-2,j)$ для всіх $j \in V_{k-2}$, потім COST $(k-3,j)$ для всіх $j \in V_{k-3}$, і так далі до COST $(1,s)$. Шлях розв'язку можна також легко визначити, якщо запам'ятовувати прийняті рішення на кожному рівні. Нехай $D(i,j)$ — це значення l, яке мінімізує $c(j,l)$ + COST $(i+1,l)$.

Задача пошуку на багаторівневому графі також може бути розв'язана з використанням підходу зворотного перегляду. Нехай $BP(i,j)$ — мінімальна вартість шляху від s до j в V_i. Нехай BCOST (i,j) — це вартість $BP(i,j)$. Згідно із зворотним переглядом матимемо:

$$\text{BCOST}(i,j) = \min_{l \in V_{i-1}, <l,j> \in E} \{\text{BCOST}(i-1,l) + c(l,j)\} \qquad (17.2)$$

Оскільки BCOST $(2,j) = c(1,j)$, якщо $<1,j> \in E$, і BCOST $(2,j) = \infty$, якщо $<1,j> \notin E$, тоді обчислення BCOST (i,j) можна організувати, використовуючи (17.2), знайшовши BCOST для $i = 3$, потім для $i = 4$ і т. д.

17.6. Задача пошуку найкоротшого шляху

Повернемось до знаходження найкоротших шляхів для всіх пар вершин графу G, що не має циклів від'ємної довжини, з погляду динамічного програмування. Нагадаємо, що нам потрібно визначити матрицю D, враховуючи, що $d(i,j)$ — це довжина найкоротшого шляху від i до j.

Розглянемо найкоротший шлях від i до j в G, $i \neq j$. Цей шлях починається у вершині i, проходить через кілька проміжних вершин (їх може й не бути) і закінчується у вершині j. Ми можемо припустити, що цей шлях не має циклів, оскільки існуючий цикл можна знищити без збільшення довжини шляху (немає циклів з від'ємною ціною). Якщо k — проміжна вершина на цьому найкоротшому шляху, тоді підшляхи від i до k та від k до j мають бути найкоротшими шляхами від i до k та від k до j відповідно. Інакше шлях від i до j буде не мінімальної довжини. Отже, принцип оптимальності справджується. Це дає можливість використовувати динамічне про-

грамування. Якщо k — проміжна вершина з найбільшим індексом, тоді шлях від i до k буде найкоротшим шляхом від i до k в G, що не проходить через вершини з індексом, більшим за $k-1$. Також шлях від k до j буде найкоротшим шляхом від k до j в G, якщо він проходить через вершини з індексом, не більшим за $k-1$. Побудова найкоротшого шляху від i до j спершу потребує визначення найбільшого індексу проміжної вершини k. Після цього нам треба знайти два найкоротші шляхи. Один від i до k, другий — від k до j. Жоден з них не може проходити через вершину з індексом, більшим за $k-1$. Використовуючи $d^k(i,j)$ для задання довжини найкоротшого шляху від i до j, що не проходить через вершини з індексом, більшим за k, отримаємо:

$$d(i,j) = \min\{\min_{1\le k\le n}\{d^{k-1}(i,k) + d^{k-1}(k,j)\}, a(i,j)\} \qquad (17.3)$$

Очевидно, $d^0(i,j) = a(i,j)$, $1 \le i \le n$, $1 \le j \le n$. Можемо отримати рекурентне співвідношення для $d^k(i,j)$, враховуючи міркування, схоже з використаним раніше. Найкоротший шлях від i до j проходить через вершини, що менші за k, або проходить через k, або не проходить ніде взагалі. Якщо проходить $d^k(i,j) = d^{k-1}(i,k) + d^{k-1}(k,j)$. Якщо ні, тоді жодна проміжна вершина не має індексу, більшого за $k-1$. Тому $d^k(i,j) = d^{k-1}(i,j)$. Поєднуючи, отримаємо:

$$d^k(i,j) = \min_{k\ge 1}\{d^{k-1}(i,j), d^{k-1}(i,k) + d^{k-1}(k,j)\} \qquad (17.4)$$

Зрозуміло, що (17.4) не справджується для графів з циклами від'ємної довжини.

Рекурентне співвідношення (17.4) може бути вирішене для D^n, якщо знайти спочатку D^1, потім D^2, D^3 і т. д., оскільки в G немає індексу, більшого за n, $D(i,j) = D^n(i,j)$. Визначення робиться ітераційно, крім того: $D^k(i,k) = D^{k-1}(i,k)$ і $D^k(k,j) = D^{k-1}(k,j)$. Отже, коли D^k формується, k-й стовпчик i рядка не змінюється і тому верхній індекс у D непотрібний.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте поняття динамічного програмування.
2. Сформулюйте принцип оптимальності Беллмана.
3. Охарактеризуйте застосування динамічного програмування до розв'язання задачі знаходження найкоротшого шляху.
4. Опишіть, яким чином формуються рекурентні рівняння при прямому і зворотному порядку перегляду варіантів розв'язку.

ТЕМА ДЛЯ ОБГОВОРЕННЯ

Порівняйте динамічне програмування і жадібні алгоритми.

ЗАДАЧІ І ВПРАВИ

1. Задача комівояжера. Комівояжер повинен відвідати n міст, побувавши в кожному лише 1 раз так, щоб сумарна довжина шляху була наймен-

шою. Відстань між містами i та j позначатимемо C_{ij}.

Нехай $T(i; J_1, \ldots, J_k)$ — вартість оптимального переходу з міста i в місто j, при цьому маршрут проходить через міста $J_1, \ldots, J_k$ в довільному порядку і не проходить ні через які інші міста. Зрозуміло, що:

$$T(i; J_1) = C_{iJ_1} + C_{J_1 1}$$

$$T(i; J_1 J_2) = \min\{C_{ij} + C_{j_1 j_2} + C_{j_2 1},\ C_{ij_2} + C_{j_2 j_1} + C_{j_1 1}\} =$$

$$= \min\{C_{ij_1} + T(J_1; J_2),\ C_{ij_2} + T(J_2; J_1)\}$$

...

Використовуючи метод математичної індукції, можна вивести:

$T(i; J_1 \ldots J_k) = \min\{C_{ijm} + T(J_m; J_1 \ldots J_{m-1} J_{m+1} \ldots J_k)\}$, $1 = m = k$.

Написати програму реалізації і побудувати часову оцінку алгоритму вирішення задачі комівояжера.

2. Нехай є контекстно-вільна граматика G, яка задає мову $L(G)$ в алфавіті A. Перевірити, чи $w \in L(G)$, де $w \in A^*$. Потрібно написати алгоритм розв'язку цієї задачі за час $O(|w|^3)$ з використанням ідей динамічного програмування.
3. *Задача про триангуляцію.* Задано n-кутник. Триангуляція — розбиття його діагоналями на трикутники, що не перетинаються. Оцінка триангуляції — сума довжин діагоналей. Зробити триангуляцію з мінімальною оцінкою.
4. *Проблема корекції.* Задано два слова x та y та операції видалення і вставки символа. Яким чином можна за мінімальну кількість операцій з x зробити y?
5. Припустимо, що n програм зберігаються на двох стрічках. Нехай l_i буде довжиною стрічки, яка потрібна для зберігання i-ї програми. Припустимо, що $\Sigma\, l_i \le L$, де L — довжина кожної стрічки. Програма може зберігатись на будь-якій з двох стрічок. Якщо S_1 — набір програм на стрічці 1, тоді найгірший час доступу до програми пропорційний $\max\{\Sigma_{i \in S_1} l_i, \Sigma_{i \notin S_1} l_i\}$. Оптимальний розподіл програм на стрічках мінімізує найгірший час доступу. Сформулюйте згідно з методом динамічного програмування підхід до визначення найгіршого випадку часу доступу при оптимальному розташуванні. Яка часова складність вашого алгоритму?
6. N робіт мають бути виконані на двох машинах A та B. Якщо робота i виконується на машині A, тоді потрібно a_i одиниць часу для обробки. Якщо вона обробляється на машині B, тоді потрібно b_i одиниць часу для обробки. Цілком можливо, що $a_i \ge b_i$ для деяких i, коли $a_j < b_j$ для деяких j, $j \ne i$.

Умови:

а) сформулюйте згідно з принципами динамічного програмування підхід до визначення мінімального часу, що потрібний для виконання всіх робіт;

б) вкажіть, як ви вирішуватимете отримані рекурентні співвідношення і як ви визначатимете оптимальний розподіл робіт на машинах.

7. N робіт мають бути виконані на одній машині. З роботою i зв'язаний кортеж (p_i, t_i, d_i); t_i — час обробки, що потрібний для виконання i. Якщо робота i виконана до терміну d_i, тоді одержується прибуток p_i. Якщо ні, то нічого не отримується. Нехай J — це підмножина робіт, усі з яких можна завершити за їх терміни виконання тоді і тільки тоді, коли роботи в J можуть бути оброблені в неспадному порядку їх термінів виконання без порушення жодного з термінів. Припустимо, що $d_i \le d_{i+1}$, $1 \le i < n$. Нехай $f_i(x)$ — максимальний прибуток, що можна одержати з підмножини J робіт, коли $n = i$, $f_n(d_n)$ — значення оптимальної вибірки робіт J. $f_0(x) = 0$. Покажіть, що для $x \le t_i$, $f_i(x) = \max\{f_{i-1}(x), f_{i-1}(x - t_i) + p_i\}$.

Розділ 18

БЕКТРЕКІНГ

Возвращаться — плохая примета...

З пісні

18.1. Основні поняття

Серед фундаментальних принципів побудови багатьох алгоритмів розв'язку задач значне місце посідає ***бектрекінг***, або пошук розв'язку з можливістю повернення назад на визначену кількість кроків при досягненні такого стану розв'язку, коли не можна отримати подальших просувань у вибраному раніше напрямі досягнення кінцевого результату вирішення задачі. Термін "бектрекінг" вперше був введений Лемером у 50-х роках і дістав початкову теоретичну розробку в працях Р. Волкера у 1960 р. [327]. С. Голомб, Л. Баумен [300] продемонстрували широке практичне застосування методу. Найбільш вдале поєднання опису різнобічних характеристик бектрекінгу можна знайти в роботах [15, 165, 290, 319].

Для більшості випадків застосування методу бектрекінгу зручною формою задання бажаного розв'язку є можливість його подання у вигляді n-мірного кортежу $(x_1, \ldots, x_n)$, де x_i вибрані з певного скінченного набору A_i. Починаємо будувати розв'язок з пустого кортежу є довжини 0. Маючи частковий розв'язок $(x_1, \ldots, x_i)$, намагаються знайти таке допустиме значення x_{i+1}, що приводить або до розширення розв'язку, або до кінцевого розв'язку проблеми у вигляді $(x_1, \ldots, x_i, x_{i+1})$. Якщо таке припустиме, проте ще не використане значення x_{i+1} існує, тоді додаємо його до попереднього часткового розв'язку і продовжуємо процес, але вже для $(x_1, \ldots, x_i, x_{i+1})$. Якщо ж такого x_{i+1} не вдається знайти, то потрібно повернутись на попередній крок до $(x_1, \ldots, x_{i-1})$ і повторити процес знаходження нового, ще не використаного допустимого значення x_i.

Отже, для застосування методу потрібно, щоб шуканий розв'язок можна було б подати у вигляді послідовності $(x_1, \ldots, x_n)$. Формальніше припус-

кають, що для кожного $k > 0$ існує деяка множина A_k, з якої вибирають кандидатів для k-ї координати часткового розв'язку. Зрозуміло, що множина A_k повинна бути визначена таким чином, що для кожного цілочислового розв'язку $(x_1, \ldots, x_n)$ поставленої задачі і для кожного $h \le n$ множина A_k містила елемент x_h. Для багатьох задач множина A_k містить деякі надлишкові елементи, що не з'являються у k-й координаті жодного цілочислового розв'язку.

Припускають також існування певної булевої функції обмеження, яка довільному частковому розв'язку $(x_1, \ldots, x_i)$ ставить у відповідність значення $P(x_1, \ldots, x_i)$ таким чином:

$$P(x_1,\ldots,x_i) = \begin{cases} \textbf{хибність}, & \textit{якщо послідовність } (x_1, \ldots, x_i) \textit{ неможливо розширити до розв'язку;} \\ \textbf{істина}, & \textit{якщо } x_i \textit{ допустиме для часткового розв'язку } (x_1, \ldots, x_{i-1}). \end{cases}$$

Останнє не означає, що $(x_1, \ldots, x_{i-1})$ обов'язково розшириться до повного розв'язку.

Усі отримані розв'язки можна умовно поділити на дві категорії:

1) при вирішенні задачі організується пошук одного вектора, що максимізує (або мінімізує, або задовольняє) функцію критерію $P(x_1, \ldots, x_n)$;

2) інколи шукаються всі такі вектори, що задовольняють P.

18.2. Базовий алгоритм

Процес бектрекінгового алгоритму можна описати за такою схемою:

```
begin
    m := 1;
    while m > 0 do
    if існує ще невикористаний елемент y ∈ Ak, такий, що
    P (X [1], ..., X [k1], y) then
    begin X [k] := y;
        if <X [1], ..., X [k]> є цілочисловим розв'язком
       then write (X [1], ..., X [k]);
       k := k + 1;
    end
    else (*повернення на коротший частковий розв'язок; усі
    елементи множини Ak знову стають невикористаними*)
    k := k – 1
end;
```

Цей алгоритм знаходить усі розв'язки з припущенням, що множини A_k скінченні і що існує n, таке що $P(x_1, \ldots, x_n)$ = хибність для всіх $x_i \in A_i, \ldots, x_n \in A_n$.

Можна довести [165] більш загальну властивість. Нехай $r > 0$ і $(x_1, \ldots, x_{r-1})$ — деякий частковий розв'язок. Розглянемо першу ітерацію головного циклу в схемі бектрекінгу з $h = r$, $X[i] = x_i$, $1 \le i \le r$. Починаючи з цієї ітерації, алгоритм генерує всі цілочислові розв'язки, які є розширенням послідовності $(x_1, \ldots, x_{r-1})$ і приходить до стану, коли $k = r - 1$. Для $r = 1$ ця властивість означає просто коректність алгоритму.

18.3. Рекурсивна схема

Використовуючи цю властивість, можна запрограмувати рекурсивну схему організації бектрекінгу у вигляді процедури AP (k). Елемент повернення в ній явно не присутній і реалізується за рахунок механізму рекурсії:

```
procedure AP (k);
(*генерування всіх розв'язків, що є розширенням послідовності
X [1], ..., X [k – 1]; масив X — глобальний *)
begin
   for y ∈ Ak такого, що P (X [1], ..., X [k – 1], y) do
   begin X [k] := y;
      if X [1], ..., X [k] є цілочисловим розв'язком then
      write (X [1], ..., X [k]);
      AP (k + 1);
   end
end;
```

Наприклад, у разі вирішення задачі сортування цілих чисел $B\ (1:n)$, розв'язок можна описати n-мірним кортежем, де x_i буде індексом в B найменшого i-го елемента. Функцію критерію P задаватиме нерівність $B\ (x_i) \le B\ (x_{i+1})$, де $1 \le i < n$. Набір A_i є скінченним і включає цілі числа від 1 до n.

Для багатьох задач функції обмеження результату мають вигляд множини *експліцитних* (явні) та *імпліцитних* (неявні) обмежень. Експліцитні обмеження — правила, що обмежують набір вибору кожного x_i. Наприклад,

$x_i \ge 0$ або $A_i = \{$*усі невід'ємні дійсні числа*$\}$.

Експліцитні обмеження — можливий простір розв'язків задачі розгляду, імпліцитні обмеження, які з наборів у просторі розв'язків задачі фактично задовольняють функцію критерію. Вони описують, яким чином x_i повинні співвідноситися один до одного.

18.4. Типові задачі

Розглянемо використання запропонованої термінології на прикладі задачі про n ферзів. Вона належить до класу комбінаторних і полягає в тому, щоб розмістити n ферзів на шаховій дошці розміром $(n \times n)$ так, щоб жодні два не атакували один одного, тобто не знаходилися в тому самому рядку, стовпці або діагоналі. Пронумеруємо рядки і стовпці шахівниці

числами від 1 до n. Ферзі також можуть бути пронумеровані від 1 до n. Без втрати загальності можемо припустити, що ферзь i повинен бути поміщений у рядок i. Усі розв'язки задачі можуть бути виражені у вигляді n-мірних кортежів $(x_1, \ldots, x_n)$, де x_i — стовпчик розміщення i-го ферзя. Експліцитні обмеження набудуть вигляду: $A_i = \{1, 2, \ldots, n\}$, $1 \le i \le n$. Отже, простір рішення складається з n^n кортежів. Імпліцитні обмеження задачі полягають у тому, що жодні два x_i не можуть бути однаковими. Перше з цих двох обмежень говорить, що всі розв'язки — перестановки з n-мірних кортежів $(1, 2, 3, \ldots, n)$. Це дозволяє зменшити розмір простору розв'язків від n^n кортежів до $n!$ кортежів. Пізніше ми встановимо, як сформулювати друге обмеження для x.

Розглянемо ще один приклад застосування бектрекінгу [304] для розв'язку задачі про суму підмножин. Нехай задано $n + 1$ додатних чисел: w_i, $1 \le i \le n$ та M. Потрібно знайти всі підмножини множини $\{w_i\}$, сума яких дорівнює M. Наприклад, якщо $n = 4$, $(w_1, w_2, w_3, w_4) = (11, 13, 24, 7)$ і $M = 31$, тоді бажаними підмножинами будуть (11, 13, 7) і (24, 7). Замість запису в векторі розв'язку чисел w_i, які в сумі дають M, ми могли задати вектор розв'язків шляхом внесення індексів цих w_i. Тоді наші два розв'язки опишуться векторами (1, 2, 4) і (3, 4). Експліцитні обмеження набувають вигляду $x_i \in \{j \mid j$ — *ціле число* і $1 \le j \le n\}$. Імпліцитні обмеження вимагають, щоб жодні два доданки не були однаковими і щоб їх сума дорівнювала M, оскільки ми хочемо уникнути генерування кратних випадків тієї ж самої підмножини (для прикладу (1, 2, 4) і (1, 4, 2) являють собою ті самі підмножини). Інше імпліцитне обмеження, що накладається,— це $x_i < x_{i+1}$, $1 < i \le n$.

В іншому варіанті розв'язку задачі про знаходження суми підмножин кожна підмножина розв'язків може відображатися кортежем $(x_1, x_2, \ldots, x_n)$ таким, що $x_i \in \{0, 1\}$, $1 \le i \le n$; $x_i = 0$, *якщо* w_i *не вибране*, і $x_i = 1$,— відповідно, *якщо вибране*. Розв'язки у згаданому випадку набуватимуть вигляду — (1, 1, 0, 1) і (0, 0, 1, 1). Для побудови всіх розв'язків використовуватимуться кортежі фіксованого розміру. Отже, можна вважати, що розв'язок однієї задачі може приймати різні форми. Можна перевірити, що для обох згаданих вище формулювань простір розв'язків складається з 2^n різних кортежів.

Зрозуміло, що для бектрекінгових алгоритмів пошук розв'язків полегшиться, якщо простір розв'язків задати у вигляді дерева [298]. Тоді пошук розв'язку зручно описувати за допомогою техніки пошуку в дереві. Кожна вершина дерева відповідає деякій послідовності $(x_1, \ldots, x_n)$, за винятком кореня. Вершина кореня відповідає пустій послідовності ε. Вершини позначені $(x_1, \ldots, x_k, y)$, є синами вершини $(x_1, \ldots, x_k)$.

Розглянемо повне дерево T, яке складається з усіх можливих послідовностей типу $(x_1, \ldots, x_k)$, де $0 \le k \le n$, $x_i \in A_i$ для $1 \le i \le k$. Тимчасово припустимо, що кожний елемент $y \in A_k$ є допустимим для $(x_1, \ldots, x_{k-1})$, якщо $k \le n$ і жоден елемент не буде допустимим для $(x_1, \ldots, x_{k-1})$, $k > n$, тобто

$$P(x_1, \ldots, x_k) = \begin{cases} true, & \text{якщо } k \le n \\ false, & \text{якщо } k > n \end{cases}$$

$x_i \in A_i$ для $1 \le i \le k$. Тоді зрозуміло, що виклик AP (1) викличе пошук у глибину в дереві T. У разі складнішої функції P процес пошуку пропускає розгляд вершини піддерев з коренем у кожній зустрінутій "недопустимій" вершині. За рахунок цього й позбавляються повного перебору елементів. Але для багатьох задач часова складність бектрекінгового алгоритму в "найгірших" випадках носитиме експоненційний характер.

Повернемось знову до задачі про n ферзів. Розглянемо згідно з [304] випадок $n = 4$. За функцію обмеження використаємо очевидні критерії: якщо $(x_1, x_2, \ldots, x_i)$ — шлях до поточної E-вершини, тоді всі дочірні вершини з поміткою батько-син x_{i+1} такі, що $(x_1, \ldots, x_{i+1})$ відображають конфігурацію шахової дошки, в якій жодні два ферзі не нападають. Ми починаємо з кореня як єдиної живої вершини. Він стає E-вершиною, при цьому шлях буде пустим — (ε). Потім генеруємо одну дочірню клітину. Занесемо в неї вершину за номером 2. Шлях набуде вигляду (1) і вершина 2 першого ферзя знаходитиметься в першій колонці. Вершина 2 стає E-вершиною. Вершина 3 після створення негайно знищується. Наступною генерується 8-ма вершина і шлях стає (1, 3). Вершина 8 стає E-вершиною. Вона знищується, оскільки всі дочірні вершини описують конфігурацію дошки, яка не зможе привести до вершини відповіді. Ми повертаємось до вершини 2 і генеруємо іншу дочірню вершину — 13. Тепер шлях — (1, 4). Рис. 18.1 описує бектрекінговий процес такого пошуку розв'язків:

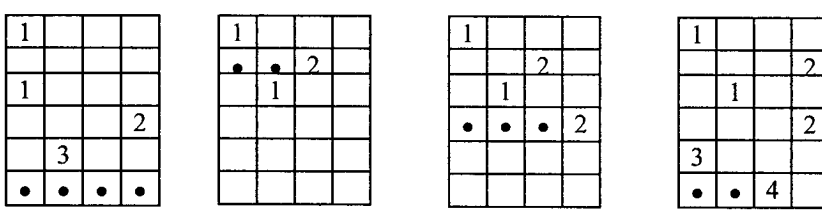


Рис. 18.1. Процес бектрекінгового пошуку розв'язку

Крапки вказують на спроби розміщення ферзя, які були відхилені.

Техніка оцінювання ефективності бектрекінгових алгоритмів була запропонована вперше Кнутом [301] і дістала подальшого розвитку в [308].

Ефективність реалізації бектрекінгу залежить від чотирьох чинників: часу генерування наступного x (k); часу визначення x (k)-го, що задовольняє експліцитним обмеженням; часу реалізації функцій обмеження P_i; часу визначення номера x (k), що задовольняє P_i для всіх i. Функції обмеження будуть вживаними, якщо вони фактично зменшують число вершин генерування. Якщо для задачі розмір простору станів дерева є завеликий, щоб дозволити генерування всіх вершин, тоді функції обмеження повинні будуватися так, щоб принаймні один розв'язок був знайдений у прийнятний час.

Використання перестановок спеціального типу — загальний принцип побудови ефективного пошуку. Для багатьох задач множини A_i можуть використовуватися в будь-якому порядку. Тобто ефективніше було б вибирати наступний елемент з найменшої множини за рівності всіх інших елементів. Але теоретично можна показати, що при пересічному виборі використання найменшої множини виявляється ефективнішим. Із розглянутих факторів, що впливають на ефективність бектрекінгових алгоритмів, три не залежать від конкретної задачі і є постійними за фіксованої організації дерева простору станів. Четвертий чинник (число вершин генерування) змінюється від задачі до задачі і може давати або поліноміальну, або експоненційну оцінку. Якість алгоритму і визначається в його спроможності вирішити задачі з великим вхідним набором за малий проміжок часу.

Часто використовують імовірнісний підхід для оцінки числа згенерованих бектрекінгом вершин [304]. Особливо вдалим є його використання у задачах, які вимагають знаходження всіх можливих розв'язків за рахунок перегляду всіх необмежених вершин. Наприклад, можна застосувати метод Монте Карло, основна ідея якого полягає в генеруванні довільного шляху в дереві простору станів.

Нехай x буде вершиною з цього шляху, що знаходиться на i-му рівні дерева простору станів. Функції обмеження у вершині x визначають номер m_i дочірньої вершини, що не підлягає обмеженню. Вибирають одну з таких вершин. Ґенерування шляху закінчується, якщо вершина є листком або всі її дочірні вершини — обмежені. Використовуючи m_i, можна визначити загальне число m необмежених вершин дерева простору станів, але при цьому потрібно робити припущення щодо типу застосовуваних функцій обмежень. Розглянемо випадок, коли ці функції є статичними.

Легко побачити, що число необмежених вершин на рівні 1 дорівнює 1, на рівні 2 — m_1. Якщо дерево пошуку задовольняє умові, що всі вершини певного рівня мають один ступінь, тоді можна очікувати, що кожний рівень з двома вершинами мав би в середньому m_2 необмежених дочірніх. Тому на рівні 3 матимемо $m_1 \times m_2$ необмежених вершин. Очікуване число таких вершин на рівні 4 — $m_1m_2m_3$. Можна довести, що число необмежених вершин рівня $i + 1$ буде $m_1m_2 \ldots m_i$. Тому, $m = 1 + m_1 + m_1m_2 + m_1m_2m_3 + \ldots$.

Продемонструємо використання описаної узагальненої техніки бектрекінгових алгоритмів на конкретних задачах. Спочатку уточнимо розв'язок задачі про розміщення ферзів.

Класична шахівниця має розмір 8×8. Тому розглянемо задачу про розміщення на ній 8 ферзів. З викладеного можемо заключити, що розв'язок може бути представлений кортежем $(x_1, \ldots, x_n)$, де x_i є стовпцем i-го рядка шахівниці, в якому буде розміщений i-й ферзь. Усі x_i будуть відмінними, бо жодні два ферзі не можуть бути поміщені в один стовпчик. Залишається лише подумати, як перевіряти, чи два ферзі знаходяться на одній діагоналі.

Для вирішення задачі пронумеруємо вертикальні і горизонтальні рядки шахової дошки від 1 до n. Для нумерації діагоналі розділимо їх на два типи таким чином, щоб діагональ специфікувалась типом і номером, які обчислюються з її вертикальних і горизонтальних рядів:

Diagonal = N + Column – Row (type 1)
Diagonal = Row + Column – 1 (type 2)

Якщо ви дивитесь на шахівницю (ряд 1 по горизонталі та колонку 1 по вертикалі) з лівої сторони, тоді type 1 розділяє клітку діагоналлю як символ похилої риски вліво (\), а type 2 — вправо (/). Рис. 18.2 демонструє нумерацію діагоналей type 2.

	1	2	3	4	5	6	7	8
1	1	2	3	4	5	6	7	8
2	2	3	4	5	6	7	8	9
3	3	4	5	6	7	8	9	10
4	4	5	6	7	8	9	10	11
5	5	6	7	8	9	10	11	12
6	6	7	8	9	10	11	12	13
7	7	8	9	10	11	12	13	14
8	8	9	10	11	12	13	14	15

Рис. 18.2. Нумерація діагоналей шахівниці другого типу

Позицію ферзя можна описати за допомогою зазначення номера рядка та колонки. Отже, уявимо, що квадрати шахової дошки є елементами двовимірного масиву $C(1{:}n, l{:}n)$. Припустимо, що два ферзі розміщені відповідно (i, j) і (k, l). Тоді вони знаходяться на одній діагоналі, тільки якщо $i - j = k - l$ або $i + j = k + l$. Перше рівняння є рівносильним $j - l = i - k$, а друге $j - l = k - i$. Отже, два ферзі розміщуються на одній діагоналі тоді і тільки тоді, коли $|j - l| = |i - k|$.

Процедура **PLACE** (A) повертає булеве значення "істина", якщо k-й ферзь може бути розміщений у поточному значенні x (k). Вона перевіряє також, чи відмінне x (k) від усіх попередніх значень x (1), …, x (k – 1), і чи немає іншого ферзя на тій же діагоналі. Обчислювальний час становить O (k – 1).

```
Procedure PLACE (k)
//повертає true, якщо ферзь може бути розміщений на k-му рядку//
begin
    i := 1;
    while i < k do
    begin
        if X (i) = X (k) //два ферзі в одній колонці//
        or ABS (X (i) – X (k)) = ABS (i – k) //на одній діагоналі//
        then return (false);
        i := i + 1
    end
    return (true)
end PLACE;
```

Використовуючи процедуру PLACE та алгоритм опису загальної стратегії бектрекінгу, доходимо до розв'язку задачі про n ферзів (процедура NQUEENS):

```
Procedure NQUEENS (n)
//використовуючи бектрекінг, ця процедура друкує всі можливі//
//розміщення n ферзів на шаховій дошці n n так, щоб вони//
//не атакували один одного//
begin
    X (1) := 0;
    k := 1;
    while k > 0 do //виконати для всіх рядків//
    begin
        X (k) ← X (k) + 1
        while X (k) ≤ n and not (PLACE (k)) do
        //чи можна розмістити ферзя?//
        X (k) ← X (k) + 1;
        if X (k) ≤ n //позицію знайдено//
        then
        if k = n
        then print (X)
        else
        begin
            k := k + 1;
            X (k) := 0
        end
        else k := k – 1;
    end
end NQUEENS;
```

18.5. Сума підмножин

Припустимо, що дано n різних невід'ємних чисел і додатнє число M. Потрібно знайти всі комбінації цих чисел, сума яких дорівнює M. При обговоренні загальної стратегії бектрекінгових алгоритмів ми зазначали, що переформулювати цю задачу можна з використанням фіксованих або змінних кортежів. Зупинимось на першому варіанті. Нагадаємо, що в даному разі елемент $x\,(i)$ з вектора розв'язків дорівнює 0 або 1 залежно від того, включений $w\,(i)$-й доданок до суми чи ні.

Легко бачити, що для цієї задачі підійде така функція обмеження:

$$P_k(x(1),\ldots,x(k)) = true, \quad \text{якщо } \sum_{i=1}^{k} w(i)x(i) + \sum_{i=k+1}^{n} w(i) \geq M.$$

Зрозуміло, що перебір варіантів можна поліпшити, якщо розмістити доданки в зростаючому порядку. В такому разі ($x\,(1)$, …, $x\,(k)$) не може вести до вершини відповіді, якщо:

$$\sum_{i=1}^{k} w(i)x(i) + w(k+1) > M.$$

Отже, функцію обмеження можна підсилити таким чином:

$$P_k(x(1),\ldots,x(k)) = true, \quad \text{якщо } \sum_{i=1}^{k} w(i)x(i) + \sum_{i=k+1}^{n} w(i) \geq M \text{ і}$$

$$\sum_{i=1}^{k} w(i)x(i) + w(k+1) \leq M.$$

Оскільки визначили вже все потрібне для застосування бектрекінгового алгоритму, виберемо лише схему реалізації. З останнього зауваження щодо визначення функції обмеження природно випливає надання переваги рекурсивному варіанту:

```
procedure SUMOFSUB (s, k, r);
begin
    X (k) := 1;
    if s + W (k) = M //підмножина знайдена//
    then
    for j := 1 to k do
    print (X (j));
    else
    if s + W (k) + W (k + 1) ≤ M then
    call SUMOFSUB (s + W (k), k + 1, r – W (k));
    if s + r – W (k) ≥ M and s + W (k + 1) ≤ M then
    begin
        X (k := 0;
        call SUMOFSUB (s, k + l, r – W (k))
    end
end SUMOFSUB;
```

Процедура SUMOFSUB уникає обчислювання

$$\sum_{i=1}^{k} w(i)x(i) + \sum_{i=k+1}^{n} w(i)$$

щоразу за рахунок збереження цих значень у вигляді змінних s і r відповідно. Алгоритм припускає $W(1) \leq M$ і $\sum W(i) \geq M$. Початковим зверненням є **call SUMOFSUB (0, 1, $\sum W(i)$)**. Цікаво зазначити, що алгоритм явно не використовує умову $k > n$, щоб завершити рекурсію. Ця умова не потрібна, оскільки на вході в алгоритм $s \neq M$ і $s + r \geq M$. Отже, $r \neq 0$ і таким чином k не може бути більшим за n.

18.6. Гамільтонові цикли

Розглянемо задачу знаходження гамільтонового циклу в графі [165, 304]. Спочатку згадаємо задачу знаходження ейлерового циклу. Різниця між цими задачами полягає в тому, що в разі гамільтонового циклу нас цікавитиме шлях, який проходить тільки по одному разу через кож-

ну вершину (а не ребро, як у задачі про ейлерів цикл) даного графу. Такий шлях називають гамільтоновим. Аналогічно визначається і гамільтонів цикл. Але така незначна модифікація задачі суттєво змінює характер проблеми. Зрозуміло, що не для кожного графу існуватиме гамільтонів цикл. Для задачі про гамільтонів цикл не відомо необхідної і достатньої умови існування гамільтонових шляхів. Не знайдено також поліноміального алгоритму, який би перевіряв існування гамільтонового шляху в довільному графі.

Гамільтонів цикл *— шлях, що проходить через n вершин графу G за умови, що кожна вершина відвідується один раз і остання вершина відвідування є вершиною початку руху. Інакше кажучи, якщо гамільтонів цикл починається в деякій вершині* $v_1 \in G$, *і вершини графу G відвідані в порядку* $v_1, v_2, \ldots, v_{n+1}$, *тоді ребра* (v_i, v_{i+1}) *належать E,* $1 \le i \le n$ *і* v_i *відмінні, за винятком* v_1 *та* v_{n+1}, *які рівні між собою.*

Нехай маємо зв'язний граф $G = <V, E>$. Тоді можна згенерувати $n!$ різних послідовностей вершин для перевірки на задоволення умові гамільтонового шляху. Такі дії потребують не менше $n!n$ різних кроків. Але функція такого типу зростає швидше за експоненційну функцію. Тому знову використаємо ідею бектрекінгу для скорочення кількості кроків в алгоритмі повного перебору варіантів.

Розглянемо ідею бектрекінгового алгоритму, що знаходить усі гамільтонові цикли в орієнтованому або неорієнтованому графі. Кортеж розв'язку $(x_1, \ldots, x_n)$ побудуємо так, що x_i визначатиме, чи була відвідана i-та вершина запропонованого циклу. Для вибору множини можливих вершин-кандидатів до x_k, за умови, що $x_1, \ldots, x_{k-1}$ вже вибрані, будемо дотримуватись правила:

if $k = 1$ **then** $X(1)$ *може бути будь-якою з n вершин*

else if $1 < k < n$ **then** $X(k)$ *може бути будь-якою вершиною v, відмінною від* $X(1), X(2), \ldots, X(k-1)$ *і v зв'язана ребром з* $X(k-1)$

else $X(n)$ *— вершина v, що повинна бути зв'язана з* $X(n-1)$ *та* $X(1)$.

Для уникнення виведення одного циклу n разів накладається умова $X(1) = 1$.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте поняття бектрекінгу.
2. Наведіть базовий алгоритм бектрекінгового пошуку.
3. Охарактеризуйте можливості, пов'язані із застосуванням рекурсії в бектрекінгових алгоритмах.
4. Охарактеризуйте поняття експліцитних та імпліцитних обмежень.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Перелічіть основні чинники, які впливають на ефективність бектрекінгового пошуку.
2. Охарактеризуйте типові задачі, які можна розв'язувати за допомогою бектрекінгового пошуку.

ЗАДАЧІ І ВПРАВИ

1. Зробити такі модифікації в алгоритмах розділу:
   а) де шукались всі розв'язки, знайти один, і навпаки;
   б) де застосовувались рекурсивні алгоритми, використати ітераційний підхід і навпаки.
2. Запропонуйте оптимізацію процедури NQUEENS.
3. Дано шахову дошку ($n \times n$), кінь ставиться на довільний квадрат з координатами (x, y). Задача полягає в тому, щоб визначити $n^2 - 1$ переміщень коня, за умови, що кожний квадрат дошки відвідується один раз, якщо така послідовність ходів існує. Напишіть алгоритм до даної задачі.
4. Нехай дано n чоловіків і n жінок і два $n \times n$ масиви P і Q таких, що $P(i, j)$ вказує на надання переваги i-чоловіка j-жінці, а $Q(i, j)$ — надання переваги i-жінки j-чоловікові. Придумайте алгоритм, який знаходить пари чоловіків і жінок таким, щоб оптимізувати їх переваги.
5. Використати алгоритм з поверненням для вирішення таких задач:
   а) знаходження розміщення 8 взаємно неатакуючих тур на шаховій дошці;
   б) знаходження розфарбування вершин графу мінімальним числом кольорів так, щоб жодне ребро не з'єднувало двох вершин одного кольору;
   в) встановлення ізоморфізму двох графів;
   г) знаходження шляху в лабіринті. Лабіринт задається матрицею з'єднань, в якій для кожної пари кімнат вказано, чи з'єднані вони коридором. Кількість кімнат дорівнює n. Побудувати шлях із кімнати i в кімнату j;
   д) у послідовності дійсних чисел $a_1, a_2, \ldots, a_n$ знайти підпослідовність найбільшої довжини.

Розділ 19

ВИРІШУВАЧІ ІНТЕЛЕКТУАЛЬНИХ ЗАДАЧ

Будь-яка проблема має розв'язок.
Єдина складність полягає в тому,
щоб його знайти.

Е. Неф, американський журналіст

19.1. Базові поняття

У широкому розумінні цього терміна ***вирішувачем інтелектуальних задач*** є будь-яка інтелектуальна система.

Вирішувач інтелектуальних задач *у вузькому розумінні цього слова — це штучна система, яка сприймає формалізований опис деякої задачі і на основі даного опису розробляє план її вирішення.*

Важливо те, що мова опису задач повинна бути повністю або відносно **предметно-незалежною**, тобто на її основі можна описувати не одну конкретну задачу, а значну їх кількість.

При розробці вирішувача інтелектуальних задач ми повинні:

- задати мову формалізованого опису ситуацій, на основі якої можна формалізувати постановку задачі;
- формалізувати знання, доступні вирішувачеві;
- задати методику вирішення задач.

Вирішувачі інтелектуальних задач можуть бути спеціалізованими, розрахованими на застосування в обмеженій предметній галузі, та універсальними. Універсальність при цьому носить формальний характер і означає лише те, що мова опису задач не залежить від предметної області, і ми в принципі можемо описати цією мовою будь-яку постановку задачі. Але такий опис може виявитися надто громіздким або саме рішення, отримане вирішувачем, може виявитися незадовільним.

Універсальні вирішувачі інколи можна використовувати як основу для побудови реальних вирішувачів, які більш-менш ефективно розв'язують визначене обмежене коло задач.

Історично склалося так, що переважна кількість вирішувачів інтелектуальних задач була реалізована на основі планування в просторі задач, і часто для кожного вирішувача розроблялися конкретні формальні методики такого планування. Проте існують і вирішувачі на основі планування в просторі станів. Таким є, наприклад, вирішувач ПРИЗ [253].

19.2. Загальний вирішувач задач

У кінці 50-х — на початку 60-х років А. Ньюелл, Г. Саймон і Дж. Шоу опублікували піонерські роботи, які заклали початок класичного евристичного підходу до вирішення інтелектуальних задач. При цьому основний акцент було зроблено на плануванні цілеспрямованих дій. Першою інтелектуальною програмою стала розроблена ними **програма Логік-Теоретик**, за допомогою якої можна було доводити теореми математичної логіки. Ідеологія нового підходу полягала в породженні різноманітних здогадок і припущень з подальшою перевіркою їх істинності.

Розвиток цього напряму був пов'язаний з **програмою *"Загальний вирішувач задач"*** (**GPS — General Problem Solver**, інша назва — "Універсальний вирішувач задач"). Були запропоновані деякі принципи розв'язку задач з довільної предметної області. Підхід ґрунтувався на розгляді ситуацій, що виникають при вирішенні задач, та операторів, які можуть змінювати ці ситуації.

Методика **GPS** передбачає аналіз цілей і засобів для їх досягнення. Перший випливає з дослідження розбіжностей між поточною та бажаною ситуаціями. Схему прийняття рішення можна записати у такому вигляді [282]:

1. *Проаналізувати поточну ситуацію.*
2. *Порівняти ситуацію з бажаною, якщо відмінностей немає — кінець роботи.*
3. *З'ясувати, яких оператора або операторів можна застосувати для зменшення існуючої різниці.*

4. *Послідовно застосовувати операторів, знайдених на кроці* 3, *поки один з них не спрацює.*

5. *Повернутися на крок* 1.

Ця проста схема, очевидно, не залежить від конкретної предметної області та в принципі може бути застосована до вирішення будь-якої задачі. На попередньому етапі необхідно зафіксувати *перелік можливих відмінностей між поточною та бажаною ситуаціями і перелік операторів, які можуть ліквідовувати ці відмінності.*

Як приклад ілюстрації цього підходу А. Ньюелл, Дж. Шоу і Г. Саймон наводять такий монолог: *"Я хочу доставити мого сина до дитячого садка. У чому полягає різниця між тим, що є, і бажаною ситуацією? У відстані. Що може змінити відстань? Мій автомобіль. Але він пошкоджений. Що потрібно щоб його полагодити? Новий акумулятор. Де знайти новий акумулятор? У майстерні або в крамниці. Далі, відмінність полягає у тому, що у майстерні не знають, що мені потрібний новий акумулятор. Що може зняти цю відмінність? Зв'язок. Які є засоби для зв'язку? Телефон"* і т. д. При цьому часто виникає необхідність повернення назад. Так, акумулятори можуть бути в крамниці Сміта або в крамниці Брауна. Нехай обрано крамницю Сміта. Якщо з'ясовується, що телефон у крамниці Сміта не працює, доведеться повертатися на кілька кроків назад, змінювати рішення та телефонувати до крамниці Брауна.

Можна сказати, що GPS у жорсткій послідовності вирішує три типи задач:

- переведення від ситуації A до ситуації B; позначимо цю задачу $T(A, B)$;
- зменшити відмінність D між ситуаціями A і B за допомогою оператора O (позначимо цю задачу $C(D, O, A, B)$);
- застосувати оператор O до ситуації A (позначення — $R(O, A)$).

Отже, нехай ми маємо початкову ситуацію S і бажану ситуацію Q. Відповідно до цього ставиться задача $T(S, Q)$; в процесі вирішення якої шукається відмінність D. Якщо відмінностей немає, тоді задача успішно вирішена; в іншому разі вирішується задача $C(D, O, S, Q)$, а саме: знаходиться оператор O, який може зменшити відмінність D. Далі вирішується задача $R(O, S)$. Попередньо встановлюються умови H, за яких оператор O може бути застосований до ситуації S. Якщо ці умови виконуються, оператор застосовується, виникає нова ситуація S', і ставиться задача $T(S', Q)$ (перевести S' до Q). Якщо ж умова H не виконується, виникають дві нові підзадачі: $T(S, H)$ (перевести S до H) і $T(H, Q)$ (перевести H до Q).

Розглянемо як приклад одну з найвідоміших задач — ***про мавпу і банани.*** В кімнаті знаходиться мавпа, яка може вільно переміщуватися. В центрі кімнати зі стелі звисають банани. Мета мавпи — дістати банани, але вона може це зробити тільки тоді, якщо стоїть на скриньці, а скринька знаходиться точно під бананами. Скринька знаходиться в кімнаті, і мавпа може її вільно пересувати. Задача полягає в такому: розробити формальний опис цієї задачі і написати вирішувач, який знаходить шлях до мети за будь-якого початкового положення мавпи і скриньки. Один з можливих розв'язків цієї проблеми на основі Загального вирішувача задач можна знайти в [168].

Основна проблема Загального вирішувача задач полягає в тому, що **можливих операторів і відмінностей може бути дуже багато**. Ще один проблемний аспект — **робиться неявне припущення про незалежність відмінностей,** тобто про те, що зменшення одних відмінностей не веде до появи або посилення інших. У ряді реальних ситуацій це не так.

При розвитку і вдосконаленні ідеї GPS стало ясно, що при прийнятті рішень необхідно брати до уваги інші характеристики конкретної задачі, наприклад **причинно-наслідкові зв'язки**. В [282] наведено ще один приклад на цю тему. Є дворукий нерухомий робот-маніпулятор, призначений для переміщення предметів. Є два оператори: *"взяти лівою рукою"* та *"взяти правою рукою"*. Нехай спочатку робот спробував взяти предмет правою рукою і зазнав невдачі. Тут важлива причина цієї невдачі. Якщо вона полягає у тому, що до предмета неможливо дістати, слід спробувати інший оператор — *"взяти лівою рукою"*, оскільки ліва рука розташована в іншій точці простору. Якщо ж причиною невдачі була надто висока температура предмета, то невдача чекатиме і при спробі взяти його лівою рукою, тому цю спробу робити не варто.

Можливість застосування GPS була проілюстрована на ряді порівняно простих задач. Проте процедура виявилася не настільки універсальною. Часто неможливо навіть сформулювати раціональний перелік відмінностей та операцій, які ведуть до зменшення цих відмінностей. Все це особливо яскраво виявилося при спробі застосувати Загальний вирішувач задач до гри в шахи [222].

19.3. Деякі інші методики

Метод ключових операторів базується на такому. Нехай є задача *"перевести A до B"* і відомо, що в процесі вирішення цієї задачі обов'язково повинен бути застосований оператор f (такий оператор називається ключовим). Нехай відомо, що цей оператор може бути використаний у стані C, а результатом такого застосування є ситуація $f(C)$. Тоді початкова задача зводиться до:

- перевести A до C;
- застосувати f до C;
- перевести $f(C)$ до B.

Далі пробують знайти ключові оператори для першої і третьої підзадачі і т. д.

Метод продукцій системи STRIPS базується на системі продукцій P, $A \Rightarrow B$, де P, A, B — множини правильно побудованих формул логіки предикатів першого порядку, P — задає умови застосування ядра продукції $A \Rightarrow B$. Метод схожий на метод GPS, але, на відміну від GPS, використовується метод резолюцій.

Метод ABSTRIPS передбачає ієрархічне ранжування продукцій, і це дозволяє спочатку отримати базовий план вирішення задачі, а потім у разі необхідності деталізувати цей план.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке інтелектуальний вирішувач задач?
2. Охарактеризуйте Загальний вирішувач задач.
3. Вирішення яких задач і в якій послідовності передбачено формальною схемою GPS?
4. Перелічіть проблеми, пов'язані з практичним застосуванням Загального вирішувача задач.
5. Охарактеризуйте метод ключових операторів.
6. Охарактеризуйте методику STRIPS.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. З яких елементів, на вашу думку, повинен складатися формальний опис задачі для того, щоб до неї можна було застосувати Загальний вирішувач задач?
2. У чому полягає обмеженість Загального вирішувача задач?

Розділ 20
ІГРОВІ ЗАДАЧІ

Мне же неумение поможет —
Этот Шифер ни за что не сможет
Угадать, чем буду я ходить.
В. Висоцький. "Честь шахової корони"

20.1. Ігрові задачі як задачі прийняття рішень

Ігрові задачі є типовим класом задач, які традиційно належать до інтелектуальних. Оскільки вибір чергового ходу в іграх є не що інше, як прийняття рішення, методи програмування ігрових задач найтіснішим чином пов'язані з методами планування цілеспрямованих дій і прийняття рішень, що були описані в попередньому розділі.

Характерною особливістю ігрових задач є **наявність суперника**, який активно перешкоджає здійсненню цілей, що ставить перед собою кожний гравець. Як правило, грають два суперники, і їх цілі є прямо протилежними. Оскільки кожний суперник прагне максимізувати свій виграш, він намагається робити ходи, найвигідніші для себе і найневигідніші для свого суперника.

Утім, крім гри з суперником, можна розглядати інші типи ігор, наприклад, ***гру з природою***, яка робить випадкові ходи, або ***гру з партнером***, який робить найкращі для нас ходи (така гра часто називається ***кооперативною***).

Можна також говорити про ***рефлексію***, тобто про врахування намірів і можливостей суперника. Наприклад, гравець, роблячи ризикований хід, може сподіватися на те, що суперник не знайде найкращу відповідь. Гра у такому разі набуває рис імовірнісного, азартного характеру.

Існує математична дисципліна, яка має назву ***теорія ігор*** [156, 197, 211]. Вона є важливою складовою дослідження операцій. Її перший систематизований виклад було зроблено Нейманом і Моргенштерном у 1944 р., хоча перші результати відносяться до 20-х років минулого сторіччя.

Теорія ігор *— це математична дисципліна, яка вивчає аспекти поведінки учасників конфліктних ситуацій та має на меті виробити оптимальні для кожного з них стратегії такої поведінки. Конфліктною при цьому називається ситуація, коли гравці мають різні цілі (різні* ***функції виграшу****) та можуть вибирати різні засоби досягнення своїх цілей* ***(стратегії)****.*

Як приклад коротко охарактеризуємо найпростіший тип ігор — однокрокові антагоністичні ігри (часто вони називаються ***матричними***).

20.2. Основи теорії однокрокових ігор

Нехай грають два гравці: A та B. Гравець A може вибрати одну з m стратегій α_i, $i = 1, \ldots, m$. Гравець B може вибрати одну з n стратегій β_j, $j = 1, \ldots, n$. Обидва гравці здійснюють свій вибір **одночасно**, і після цього гравець B платить гравцеві A суму r_{ij}, що залежить від обраних стратегій. Таким чином, r_{ij} є виграшем гравця A, і цей виграш дорівнює програшеві гравця B. Оскільки виграш одного гравця дорівнює програшеві іншого (або виграшеві іншого гравця, взятому з протилежним знаком), сума виграшів обох гравців дорівнює нулю. Тому така гра називається ***грою з нульовою сумою***, або ***антагоністичною***.

І виграші, і програші можуть бути додатніми, від'ємними та нульовими. Якщо виграш даного гравця додатній, то він виграв і отримує певну суму, якщо ж виграш від'ємний — гравець програв та віддає цю суму.

Однокрокові антагоністичні ігри зручно характеризувати ***матрицею виграшів***, елементи якої дорівнюють виграшам одного з гравців (нехай для визначеності гравця A). Елемент r_{ij}, що знаходиться на перетині i-го рядка та j-ї колонки матриці виграшів, дорівнює виграшеві гравця A, якщо він обере стратегію α_i, а його суперник — стратегію β_j. Кожний гравець прагне максимізувати свій власний виграш. Таким чином, гравець A намагається обрати стратегію α_i, яка приведе до максимально можливого r_{ij}, а його суперник — обрати стратегію β_j, яка приведе до мінімально можливого r_{ij}.

Однокрокові ігри можна розглядати як задачі прийняття рішень в умовах невизначеності, в яких суперник завжди створює для гравця найнесприятливіші умови.

Розглянемо *приклад 20.1 — гру "парно-непарно"*. Гравець A пише на клаптику паперу будь-яке число від 1 до 4, а його суперник на іншому клаптику паперу намагається вгадати, яким буде число — парне чи непарне. Якщо він вгадує, він отримує одиничну суму, а якщо програє — віддає цю саму суму.

Побудуємо матрицю виграшів. Гравець A має чотири стратегії; стратегія α_i означає, що він записує число i. Гравець має дві стратегії: β_1 — написати "парно", β_2 — написати "непарно".

Матриця виграшів (нагадаємо, що вона будується з точки зору гравця A) матиме вигляд:

$$\begin{array}{c} \\ \alpha_1 \\ \alpha_2 \\ \alpha_3 \\ \alpha_4 \end{array} \begin{array}{c} \begin{array}{cc} \beta_1 & \beta_2 \end{array} \\ \begin{pmatrix} 1 & -1 \\ -1 & 1 \\ 1 & -1 \\ -1 & 1 \end{pmatrix} \end{array}$$

Методи математичного аналізу подібних, а також деяких інших типів ігор наведені у [156, 211].

20.3. Клас позиційних ігор, для яких існує оптимальна стратегія

Для теорії штучного інтелекту найбільший інтерес становлять методи знаходження планів гри і оптимальних стратегій для таких ігор, як шахи, шашки, "хрестики-нулики" і т. п. Ці ігри є найдослідженішими, і з погляду теорії вони ідентичні між собою.

Вказані ігри належать до класу ***позиційних ігор двох осіб***. Грають двоє гравців, і кожний з них може по черзі зробити будь-який хід з тих, що дозволяються правилами гри. Ці ігри є ***детермінованими*** у тому розумінні, що перебіг гри і вибір ходу не залежать від випадкових чинників. Крім того, це є ***ігри з повною інформацією***, тобто кожному гравцеві доступна вся інформація про будь-яку позицію, що утворюється в процесі гри. Інакше кажучи, гравець завжди може відрізнити одну позицію від іншої. Крім того, вказані ігри належать до класу ***антагоністичних ігор***, або ***ігор з нульовою сумою***. Звідси — замість двох функцій виграшу можна розглядати одну. Нарешті, всі ці ігри є ***скінченними*** (з кожної позиції можна зробити скінченну кількість ходів, і гра не може тривати нескінченно, а повинна через скінченне число ходів привести до певного результату).

Можна назвати відомі позиційні ігри, які належать до інших класів. Так, нарди є грою з повною інформацією, але недетермінованою у тому розумінні, що гравець не може зробити довільний хід; його вибір обмежений випадковими чинниками (результатом викидання). Преферанс не є грою з повною інформацією і детермінованою у тому розумінні, що початкова позиція залежить від випадку. Але після того як початкова позиція зафіксована, гравець може зробити будь-який хід, дозволений правилами.

Базою теорії вважається результат **Льюїса і Райфа**: **для кожного гравця у будь-якій позиції існує *оптимальна стратегія*, тобто алгоритм визначення чергового ходу, що приведе до визначеного оптимального результату.** "Оптимальність" слід розуміти таким чином: якщо обидва гравці A та B дотримуються своїх оптимальних стратегій, гра повинна закінчитися певним визначеним результатом. Якщо один з гравців, наприклад A, відхилиться від оптимальної стратегії, а гравець B продовжуватиме

її дотримуватися, то A від цього може тільки втратити: його виграш зменшиться. Звичайно, якщо обидва суперники грають неоптимальним чином, результат стає непередбачуваним.

Це положення можна довести по-різному, наприклад, застосувати методи, характерні для матричних ігор. Але домінуючим на сьогодні став підхід, що ґрунтується на побудові та аналізі ***дерева гри***.

20.4. Дерево гри та мінімаксна процедура

Ключовим поняттям у теорії позиційних ігор є ***дерево гри*** (його ще можна назвати ***деревом аналізу***). Воно будується таким чином.

1. **Кожна вершина дерева відповідає окремій позиції**.
2. **Корінь дерева відповідає позиції, що аналізується**. Аналіз може бути проведений для будь-якої позиції, включаючи початкову; у даному разі йдеться про повний аналіз усієї гри.
3. **Листки дерева відповідають завершальним позиціям**. Для завершальних позицій завжди відома ***функція виграшу***, тобто завжди відомий результат гри. Ця функція визначається правилами конкретної гри. Вона може задаватися деякою формулою або конкретними числовими значеннями; це не є принциповим.
4. **Дуги відповідають ходам**. Якщо в позиції 1 можливий хід, який переводить позицію 1 в позицію 2, то з позиції 1 до позиції 2 йде орієнтована дуга, що відповідає цьому ходу.

Суперники традиційно носять імена α і β. За домовленістю вважається, що правило вибору ходу в початковій позиції належить гравцеві α, тобто цей гравець починає гру. Вершини, в яких право ходу належить гравцеві α, прийнято називати ***альфа-вершинами***; вершини ж, в яких право ходу належить його суперникові, ***бета-вершинами***.

Функція виграшу оцінюється з точки зору α, тобто збігається з функцією виграшу цього гравця. Це означає, що α, який намагається максимізувати свій виграш, прагне до завершальних позицій з максимальними оцінками (звичайно, в тій мірі, в якій це дозволить його суперник). Гравець β також прагне максимізувати **свій** виграш. Але, оскільки гра розглядається з погляду α, він прагне до позицій з **мінімальними** оцінками.

За стандартною домовленістю при визначенні глибини дерева гри звичайний хід у шашках, шахах і т. п. вважається ***півходом***, а хід складається з двох півходів: ходу гравця та відповіді суперника.

Дерево гри може бути ***експліцитним*** та ***імпліцитним***. Експліцитне дерево задається явно; імпліцитне породжується по мірі необхідності. Для цього потрібно задати породжуючу процедуру і критерій завершення; останній встановлює, чи була досягнута завершальна позиція. Ширше застосовуються дві стратегії: "спочатку в глибину" та "спочатку в ширину", які повністю аналогічні відповідним стратегіям пошуку на графах.

Наведемо фрагмент дерева гри, що може виникнути у "хрестиках-нуликах" (рис. 20.1).

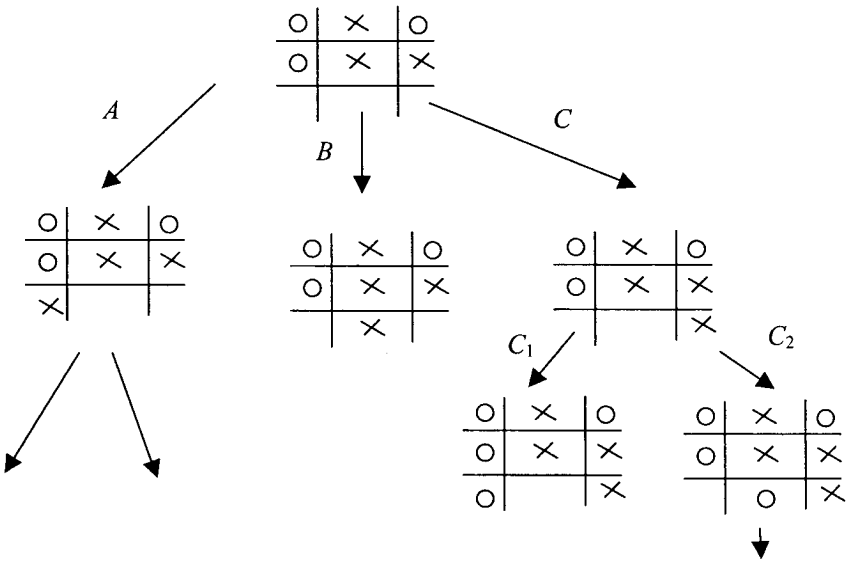


Рис. 20.1. Фрагмент дерева аналізу у "хрестиках-нуликах"

У позиції, яка аналізується, "хрестики" можуть зробити один з трьох ходів; вони помічені літерами *A*, *B*, *C*.

Хід *B* приводить до завершальної позиції, в якій перемогли "хрестики". Після ходу *C* відповідь "нуликів" C_1 приводить до їх перемоги. Відповідь C_2 формально не завершує гру, і вона повинна бути продовжена. Аналогічно гра продовжується і після першого ходу "хрестиків" *A*.

Процедура, яка дозволяє знайти оптимальну стратегію, називається ***мінімаксною процедурою***. В процесі її здійснення виконується аналіз дерева, починаючи з завершальних вершин, і кожна позиція набуває ***мінімаксної оцінки***. Ця оцінка дорівнюватиме результату гри у разі, якщо, починаючи з позиції, яка аналізується, обидва гравці гратимуть найкращим для себе чином.

Суть процедури задається такими правилами:

1. *Мінімаксна оцінка завершальних позицій дорівнює функції виграшу.*
2. *Мінімаксна оцінка альфа-вершини дорівнює максимуму мінімаксних оцінок безпосередніх наступників; відповідно до цього найкращий хід з альфа-вершини веде до позиції-наступника з максимальною оцінкою.*
3. *Мінімаксна оцінка бета-вершини дорівнює мінімуму мінімаксних оцінок безпосередніх наступників; відповідно до цього найкращий хід з бета-вершини веде до позиції-наступника з мінімальною оцінкою.*

Мінімаксна процедура є процедурою повного перебору, оскільки для її реалізації потрібно в кінцевому підсумку породити все дерево гри і розставити оцінки, рухаючись у зворотному порядку.

Розглянемо *приклад* (рис. 20.2). Нехай маємо дерево гри з такими виграшами на завершальних позиціях:

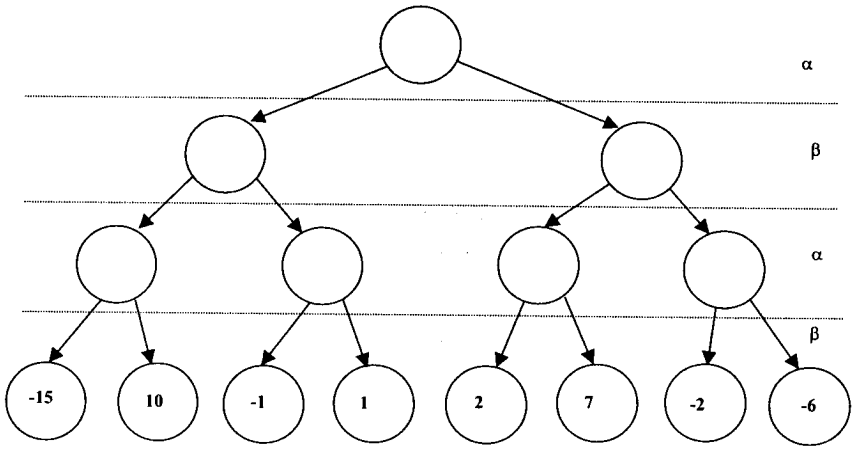


Рис. 20.2. Дерево гри

Для зручності дерево розділене на рівні і біля кожного з них стоїть мітка, яка показує, кому з гравців належить право ходу. З рис. 20.2 видно, що всі варіанти завершуються на бета-рівні; глибина дерева дорівнює півходам 3.

Зрозуміло, що гравець α хотів би потрапити до вершини з оцінкою 10, але його суперник при правильній грі цього не дозволить. Аналогічно β хотів би завершити гру з результатом –15, але такий варіант ніяк не влаштує гравця α.

Як же повинна проходити гра, якщо обидва суперники дотримуються оптимальної стратегії? На рис. 20.3 показані результати аналізу дерева гри на основі мінімаксної процедури; жирною лінією виділений оптимальний варіант.

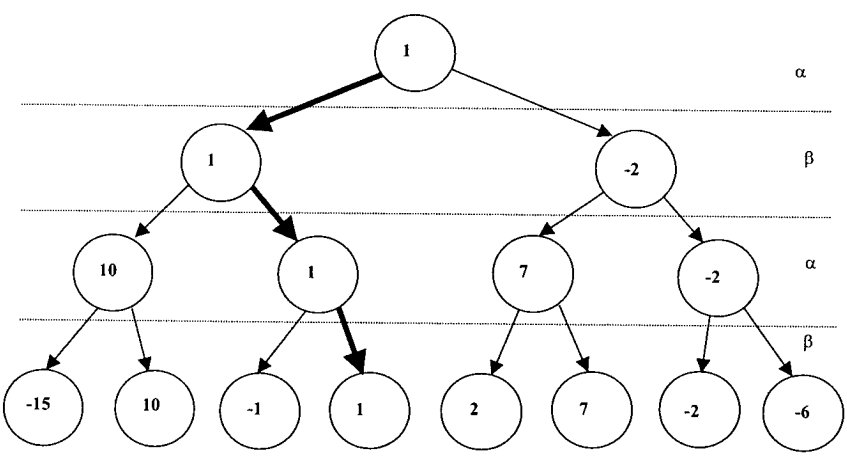


Рис. 20.3. Аналіз на основі мінімаксної процедури

Аналіз показує, що при правильній грі обох сторін α повинен виграти 1 одиницю. *Зверніть увагу на те, що всі позиції, розташовані на гілці дерева, що відповідає оптимальному варіанту, мають однакові мінімаксні оцінки* (поясніть, чому).

Для таких ігор, як "хрестики-нулики", побудова дерева гри та реалізація мінімаксної процедури не викликає особливих проблем. Але для шахів рішення може бути знайдено тільки теоретично — навіть найпотужнішому комп'ютеру на повну реалізацію цієї процедури не вистачило б часу, що пройшов з моменту створення Всесвіту.

Проаналізуємо кількість листових позицій дерева гри. Для простоти зробимо такі припущення:

1. Довжина всіх гілок дерева постійна і дорівнює d. Інакше кажучи, всі завершальні позиції знаходяться на однаковій ***глибині*** d.
2. У кожній позиції можна зробити однакову кількість ходів b, тобто кожна вершина, крім завершальних, має рівно b синів. Величину b прийнято називати ***коефіцієнтом розгалуження***.

Тоді кількість завершальних позицій дорівнює b^d, тобто складність задачі зростає експоненційно.

Проведемо приблизний розрахунок для шахів. У середньому на кожному кроці можна зробити 30 ходів; партія в середньому триває 40 ходів, або 80 півходів. Маємо величину 30^{80}. Читач може самостійно підрахувати, скільки часу повинен зайняти аналіз такого дерева.

Звичайно, в шаховій партії одній позиції можуть відповідати різні вузли дерева, але це не змінює загальної ситуації. Таким чином, основною слід вважати **проблему ефективного скорочення перебору**.

20.5. Обмеження глибини перебору

Звичайним підходом є обмеження ***глибини перебору***, тобто реалізація мінімаксної процедури для скороченого дерева гри з обмеженою кількістю ходів. При невеликій глибині перебору (5—10 півходів) дерево гри скорочується до розмірів, що є прийнятними для аналізу на сучасних комп'ютерах.

Позначимо максимальну глибину перебору (далі — просто ***глибину перебору***) через d. ***Горизонтом*** для даної позиції називається множина позицій, відстань від яких до кореня дерева аналізу піддерева точно дорівнює глибині перебору. Таким чином, аналіз варіанта повинен завершуватися при досягненні горизонту, тобто глибини d.

Але не всі позиції, завершальні для скороченого дерева, лежать на горизонті. Це пов'язано з тим, що варіант може обірватися раніше (наприклад, через мат або теоретичну нічию). З іншого боку, позиції, які лежать точно на горизонті, як правило, не є завершальними для повного дерева аналізу, але *можуть* виявитися такими.

У зв'язку з цим видається доцільним ввести поняття ***абсолютно завершальних*** і ***відносно завершальних*** позицій.

Абсолютно завершальними *називатимемо позиції, в яких гра завершується, тобто позиції, завершальні для повного дерева аналізу.*
Відносно завершальними *називатимемо позиції, які є завершальними для скороченого дерева гри, але не для повного дерева.*

У найтиповіших випадках усі відносно завершальні позиції лежать на горизонті, хоча можна навести деякі приклади процедур, для яких це не так. Надалі вважатимемо, що **всі відносно завершальні позиції лежать на горизонті**, якщо не буде явно зазначено щось інше.

Основна проблема, пов'язана з обмеженням глибини перебору, полягає в тому, **що для відносно завершальних позицій, які знаходяться на горизонті та на яких завершується аналіз варіантів з поточної вершини, оцінка позиції ще не визначена**. Тому для цих позицій замість остаточної оцінки використовують ***статичні оціночні функції***.

Статичною оціночною функцією *називається числова функція, яка призначена для оцінки виграшу гравця у даній позиції і залежить виключно від цієї позиції.*

Статичні оціночні функції можуть бути визначені, виходячи зі специфіки конкретної гри. Це робиться на основі попереднього аналізу, хоча в деяких випадках можна сподіватися на автоматизований підбір статичних оціночних функцій (як класичний приклад можна навести програму Семюеля для гри в шашки [282], яка мала здатність до самонавчання).

Часто використовуються ***лінійні*** *статичні оціночні функції*, які обчислюються за формулою:

$$S(P)=\sum_{i=1}^{n} a_i x_i,$$

де x_i, $i = 1, \ldots, n$ — числові значення факторів, що впливають на оцінку позиції, a_i, $i = 1, \ldots, n$ — вагові коефіцієнти, що визначають міру важливості того чи іншого фактора.

Так, при побудові шахових програм загальноприйнятим є аналіз матеріального співвідношення; пішак оцінюється одним очком, легка фігура — 3, тура має коефіцієнт 5 або 5,5; ферзь — 9 або 10 (ферзь слабший за дві тури). Втрата короля рівнозначна програшу, і тому він оцінюється великим числом (наприклад, 200). Певним чином оцінюються також позиційні фактори (активність фігур, пішакова структура тощо).

У шашках статична оцінка може бути обчислена як функція $6k + 4m + u$, де k — перевага в дамках, m — у простих шашках, u — беззаперечна перевага в рухливості [244, 282].

Визначення статичної оціночної функції, що було раніше, є простим і інтуїтивно зрозумілим. Але воно не зовсім чітке, оскільки і мінімаксна оцінка позиції врешті-решт однозначно визначається цією позицією. Справа в тому, що статичні оцінки, на відміну від мінімаксних, не враховують **динаміки** позиції, тобто можливостей її зміни; саме тому вони й називаються статичними.

Чіткіше визначення можна дати, якщо ввести до розгляду деяку ***формальну модель гри***.

Формальною моделлю гри називатимемо п'ятірку $M = <S, S_0, P, Q, F>$, де S — множина всіх можливих позицій; S_0 — початкова позиція; P — множина правил, які зумовлюють можливість переходу від однієї позиції до іншої, тобто — множина припустимих ходів; Q — множина завершальних позицій; F — функція, яка визначена на завершальних позиціях і встановлює виграші гравців (для антагоністичної гри достатньо задати функцію виграшів для одного з гравців).

Тепер можна сформулювати визначення мінімаксної та статичної оціночних функцій.

Мінімаксною оціночною функцією *називатимемо функцію* $f: M \to R$, *яка визначена для будь-якої позиції та обчислюється на основі описаної раніше мінімаксної процедури; значення цієї функції є мінімаксними оцінками.*

Статичною оціночною функцією *називатимемо функцію* $g: S \to R$, *яка визначена для будь-якої позиції та обчислюється на основі власних факторів цієї позиції; значення цієї функції є статичними оцінками.*

Тепер можна переформулювати мінімаксну процедуру, якщо вводяться обмеження на глибину перебору.

1. *Мінімаксна оцінка абсолютно завершальних позицій дорівнює функції виграшу; мінімаксна оцінка умовно завершальних позицій дорівнює статичній оцінці.*
2. *Мінімаксна оцінка альфа-вершини дорівнює максимуму мінімаксних оцінок безпосередніх наступників; відповідно до цього найкращий хід з альфа-вершини веде до позиції-наступника з максимальною оцінкою.*
3. *Мінімаксна оцінка бета-вершини дорівнює мінімуму мінімаксних оцінок безпосередніх наступників; відповідно до цього найкращий хід з бета-вершини веде до позиції-наступника з мінімальною оцінкою.*

Від процедури, наведеної в п. 20.6, дана процедура відрізняється лише правилом отримання оцінок для відносно завершальних позицій.

Зрозуміло, що мінімаксні оцінки, обчислені на основі скороченого дерева, можуть відрізнятися від мінімаксних оцінок, обчислених для повного дерева. Надійність аналізу скороченого дерева суттєво залежить від того, наскільки вдало вибрана статична оціночна функція.

Мінімаксна процедура, застосована до скороченого дерева, дозволяє знайти оптимальну стратегію лише для цього скороченого дерева, а не для всієї гри. Програма аналізує лише варіанти, які не виходять за горизонт (тобто які не перевищують максимальної глибини). Наприклад, у шахах цілком вірогідна ситуація, за якої з "оптимального" вузла програма зразу одержує мат. Якби збільшити глибину перебору хоча б на один півхід, програма могла б це побачити, і обрала б зовсім інший варіант. Тому *глибина перебору є одним з вирішальних чинників, що визначає силу ігрових програм та якість їх гри.*

Як узагальнення класичної мінімаксної процедури можна розглядати ***m^n-процедуру*** [244]. На відміну від класичного мінімаксу, оцінка альфа-вершини не визначається одним найкращим наступником, а є функцією від *m* найкращих наступників. Аналогічно оцінка бета-вершини є функцією від *n* найкращих наступників. Ця ідея ґрунтується на тому міркуванні, що часто краще мати декілька гарних наступників, ніж тільки одного.

Легко бачити, що звичайна мінімаксна процедура утворюється з *m^n*-процедури при $m = n = 1$. За певних умов *m^n*-процедура може показувати кращі результати, ніж звичайна мінімаксна, про що свідчать деякі експерименти, зокрема з процедурою (2, 2) [244].

Вибір оптимальних значень параметрів *m* та *n* суттєво залежить від специфіки задачі. Очевидно, якщо гра є порівняно простою і припускає повний аналіз, то класична мінімаксна процедура є оптимальною, і збільшення *m* та *n* тільки погіршить якість гри. Але *m^n*-процедуру при *m* або *n*, що не дорівнюють одиниці, можна порекомендувати за наявності різних суб'єктивних та об'єктивних невизначеностей, наприклад: не вдається отримати надійні оцінки окремих варіантів; результат ходу може залежати від випадкових чинників; суперники не завжди роблять найкращі ходи тощо.

20.6. Альфа-бета-відтинання

Застосування ***альфа-бета-процедури***, яка по суті є мінімаксною процедурою з використанням ***альфа-бета-відтинань***, є одним з найпотужніших засобів скорочення перебору. За рахунок припинення аналізу варіантів, які не вплинуть на остаточну оцінку, альфа-бета-відтинання дозволяють істотно скоротити кількість вершин, що аналізуються. При цьому досягається той самий оптимальний результат, якого було би досягнуто при повному переборі на задану кількість ходів. Якщо ж часові обмеження фіксовані, застосування альфа-бета-відтинань дозволяє істотно збільшити глибину перебору.

Відтинанням *у деякій вершині називається припинення аналізу цієї вершини разом з усіма її наступниками.*

Базову ідею, яка лежить в основі альфа-бета-відтинань, можна спрощено сформулювати так: якщо деякий хід досягає максимально можливого виграшу, інші його альтернативи можна не розглядати; якщо для деякого ходу знайдено спростування, інші відповіді суперника можна не розглядати.

Звичайно, подібне інтуїтивне положення вимагає уточнень. Дамо формальний опис альфа-бета-відтинання на основі [320]. З кожною вершиною пов'язується ***попередня оцінка***, яка змінюється в ході аналізу її наступників. Перед початком аналізу попередня оцінка альфа-вершини дорівнює $-\infty$, бета-вершини — $+\infty$. Після завершення аналізу всіх наступників або після відтинання у цій вершині згадана попередня оцінка набуває остаточного значення, тобто стає ***остаточною оцінкою***.

Попередня оцінка альфа-вершини дорівнює максимуму з остаточних оцінок безпосередніх наступників. Вона переглядається щоразу, коли бе-

та-наступник отримає остаточну оцінку, і може тільки зростати. Якщо оцінка бета-наступника є меншою, ніж власна попередня оцінка (яка збігається з остаточною оцінкою одного з раніше проаналізованих наступників), це означає тільки те, що даний варіант є гіршим, ніж той, що уже гарантований. Якщо ж остаточна оцінка вузла-наступника є більшою, ніж поточна попередня оцінка позиції, такій попередній оцінці присвоюється нове значення.

Аналогічно попередня оцінка бета-вершини дорівнює мінімуму з остаточних оцінок безпосередніх наступників і може тільки зменшуватися.

Тепер можна сформулювати правила відтинань для альфа-бета-процедури.

1. *Відтинання у альфа-вершині відбувається, коли попередня оцінка цієї вершини стає не меншою, ніж попередня оцінка будь-якого бета-попередника цієї вершини.*
2. *Відтинання у бета-вершині відбувається, коли попередня оцінка цієї вершини стає не більшою, ніж попередня оцінка будь-якого альфа-попередника цієї вершини.*

Інколи ці правила формулюються інакше: попередні оцінки альфа (бета)-вершини порівнюються з попередніми оцінками не будь-якого бета (відповідно альфа)-попередника, а лише батька. Цей варіант є менш оптимальним у тому розумінні, що він розкриває трохи більше вершин.

Рис. 20.4 ілюструє роботу альфа-бета-процедури. Коренем, як звичайно, є альфа-вершина. Попередні оцінки наводяться у прямокутниках поряд з вершинами; остаточні — в кружечках, що відповідають вершинам.

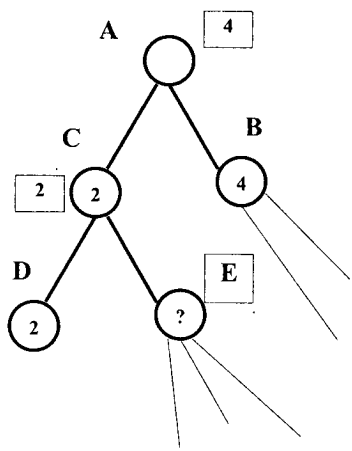


Рис. 20.4. Ілюстрація альфа-бета-відтинання

Нехай аналіз вершини *B* надав остаточну оцінку 4. Відповідно до цього попередня оцінка кореня *A*, яка раніше дорівнювала $-\infty$, змінюється на 4. Далі аналізується вершина *C*. Нехай виявилося, що остаточна оцінка його наступника *D* дорівнює 2. Тоді і вершина *C* набуває оцінки 2. Саме в цей момент відповідно до правила 2 у бета-вершині *C* можна здійснити відти-

нання: її попередня оцінка стала меншою, ніж попередня оцінка одного з альфа-попередників (у даному випадку — A). Можна сказати, що на хід до вершини C знайдено **спростування** (хід до D). Тому інші альтернативні відповіді можна не розглядати. Якщо їх оцінка виявиться більшою, гравець β все одно вибере D, якщо меншою — це тільки підсилить висновок про те, що гравцеві α ходити до C недоцільно.

Таким чином, аналіз вершини C припиняється, і вона набуває остаточної оцінки, яка необов'язково збігається з мінімаксною, але це не має суттєвого значення: вершина C все одно не може входити до оптимального варіанта та її аналіз все одно не впливає на побудову результату.

Можливі алгоритмізації мінімаксної процедури та альфа-бета-відтинання можна знайти у багатьох джерелах, наприклад [168, 320].

Ефективність скорочення перебору при альфа-бета-відтинанні суттєво залежить від порядку перегляду варіантів. Так, у наведеному вище прикладі аналіз кореневої позиції було розпочато з ходу до B, тобто з об'єктивно кращого. Якби у нашому прикладі аналіз кореневої позиції розпочати з позиції C, описане вище відтинання було б неможливим — довелося б отримувати остаточну оцінку C, а потім ще й остаточну оцінку B.

Отже, *альфа-бета-відтинання є максимальними, тобто дають найефективніше скорочення перебору, якщо насамперед аналізуються об'єктивно найкращі наступники.* Кількість позицій, що доводиться переглядати при максимальних альфа-бета-відтинаннях, приблизно дорівнює подвоєному квадратному кореню від кількості позицій, які переглядаються при повному переборі. Точніша оцінка наводиться, наприклад, у [320].

Нехай d — глибина перебору, а b — коефіцієнт розгалуження. Нагадаємо, що при повному переборі доводиться переглядати b^d завершальних позицій. Кількість завершальних позицій, які доводиться переглядати при максимальних альфа-бета-відтинаннях, дорівнює

$2b^{d/2} - 1$ — для парного d;

$b^{(d+1)/2} + b^{(d-1)/2} - 1$ — для непарного d.

Таким чином, альфа-бета-відтинання не усувають експоненційного зростання, але дозволяють зменшити його порядок. Якщо врахувати, що в шахах уже при аналізі на 3–5 ходів доводиться переглядати 10^{10}—10^{14} позицій, виявляється, що максимальні альфа-бета-відтинання можуть скоротити час на вибір ходу в сотні тисяч або мільйони разів. При заданих же часових обмеженнях можна приблизно вдвічі збільшити глибину перебору.

На практиці максимальні альфа-бета-відтинання ніколи не зустрічаються, хіба що випадково. Гравець, як правило, не може заздалегідь знати, які ходи є оптимальними — якби це знати, ні альфа-бета-відтинання, ні взагалі мінімаксна процедура не були б потрібними. Але гравець може на основі тих чи інших евристичних міркувань надати позиціям-наступникам ***робочі оцінки***, які відображають їх міру перспективності, упорядкувати їх за цими оцінками і відповідно до цього розглядати передусім ходи, які видаються найперспективнішими.

Існує багато підходів до такого впорядкування. Найпростішим з них є такий: *в першу чергу розглядати наступників з найкращою статичною оцінкою.*

Очевидною є також доцільність іншого підходу: насамперед розглядати ходи, *які істотно змінюють статичні оцінки* (наприклад, взяття фігур у шахах) або *різко скорочують кількість можливих відповідей* (наприклад, шах королю).

Широко застосовуються ***процедури неглибокого пошуку***. Вони полягають у тому, що спочатку проводиться повний аналіз на невелику глибину q, і робочі оцінки розставляються відповідно до результатів цього аналізу. Затрати на такий попередній пошук, як правило, компенсуються зростанням ефективності подальших відтинань.

Легко пересвідчитися, що впорядкування за статичними оцінками фактично є варіантом неглибокого пошуку при $q = 1$.

Упорядкування позицій за робочими оцінками може бути здійснено один раз і далі не змінюватись. У такому разі йдеться про ***фіксоване*** (інша назва — ***жорстке***) ***упорядкування***. Якщо ж по мірі додавання нової інформації, що з'являється в процесі аналізу, позиції можуть переупорядковуватися, йдеться про ***динамічне упорядкування***. Однією з давніших і найвідоміших процедур альфа-бета-відтинання з використанням такого переупорядкування є ***А-В-альфа-бета-процедура***. Очевидно, що динамічне упорядкування є гнучкішим і часто дозволяє збільшити кількість відтинань порівняно з фіксованим варіантом. Розглянемо відомий приклад на цю тему, взятий з [244].

Маємо дерево гри, зображене на рис. 20.5. Позиції другого рівня є завершальними. Їх оцінки, які виступають функціями виграшу, наведені всередині відповідних кружечків. Статичні оцінки позицій першого рівня зображені поряд з ними.

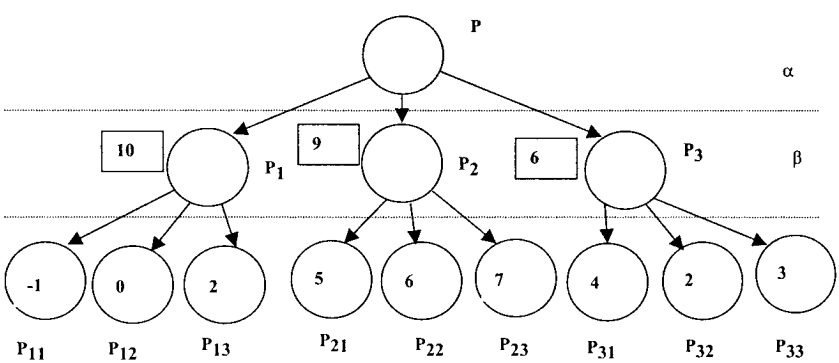


Рис. 20.5. Ілюстрація фіксованого і динамічного упорядкування

При цьому дерево є імпліцитним, тобто породжується в процесі аналізу. Відповідно до цього і оцінки завершальних позицій не можуть бути відомі зразу. Спочатку процедура здійснює упорядкування наступників першого

рівня за їх статичними оцінками; породження цих вершин здійснюється на основі стратегії "спершу в ширину". Зразу слід відмітити, що відповідно до цього упорядкування процедура оцінює хід P_1 як найперспективніший у той час, коли насправді він є найгіршим.

При фіксованому упорядкуванні, тобто якщо порядок аналізу вершин-наступників надалі не змінюється, процедура спочатку аналізує всіх наступників вершини P_1. Це дає остаточну оцінку цієї вершини -1, і відповідно змінюється попередня оцінка кореня.

Аналіз наступної позиції (P_2) змінює попередню оцінку кореня на 5. І лише після розкриття вершини P_{31}, яка має оцінку 4, вдається здійснити відтинання, яке в даному випадку завершує аналіз.

При використанні динамічного переупорядкування ситуація змінюється. Аналіз вершини P_{11} дає для вершини P_1 попередню оцінку -1, яка значно менша за робочу. Саме така обставина змушує процедуру здійснити переупорядкування, на основі якого перейти до вершини P_2.

Після аналізу наступника P_{21} робоча оцінка P_2 зменшується до 5. Але, на відміну від попереднього випадку, це зменшення є **незначним**, і тому дана процедура приймає рішення не проводити перевпорядкування, а продовжувати аналіз P_2. І це рішення в даному разі виявляється цілком правильним. У результаті аналізу вершина P_2 набуває остаточної оцінки 5, а вершина P — такої ж попередньої оцінки. Тепер можна зразу ж здійснити відтинання у вершині P_1, а після аналізу P_{31} — і в P_3. У результаті проаналізовано значно менше вершин, ніж при фіксованому упорядкуванні.

Можна сформулювати таке загальне правило [244]. Нехай поточна робоча оцінка деякої вершини дорівнює K. Для альфа-вершин переупорядкування здійснюється у разі, якщо ця вершина набуває попередньої оцінки, меншої за $K - R$, а для бета-вершин — за попередньої оцінки, більшої за $K + R$. Інакше кажучи, переупорядкування здійснюється, якщо оцінка позиції погіршується на величину, що перевищує R.

Важливий параметр R, який називається ***опірністю***, визначає, якою мірою процедура робить опір переупорядкуванню. При досить великому значенні R процедура ніколи не здійснює переупорядкування і діє подібно до жорсткого упорядкування. Якщо ж R досить мале, упорядкування буде надлишковим. Тому належний вибір R, який залежить від специфіки конкретної гри, має дуже велике значення. Очевидно також, що задачу підбору R можна, зокрема, переформулювати в рамках теорії нечітких множин.

Критерії закінчення вказують, коли треба припинити пошук. Особливе значення вони мають у тому разі, коли дерево є імпліцитним і породжується по мірі необхідності.

Ми вже знаємо такі основні критерії, як **досягнення завершальної позиції** та **досягнення максимальної глибини**. Існують деякі інші критерії, які дозволяють завершувати пошук до досягнення максимальної глибини.

Один з таких критеріїв — досягнення ***"мертвої"*** позиції, тобто такої, статична оцінка якої мало відрізняється від робочої оцінки, отриманої в результаті деякого попереднього аналізу.

20.7. Деякі графові моделі аналізу ігрових задач

Ігрові задачі можна досліджувати не тільки на основі аналізу дерев гри. Гра по суті являє собою граф станів, вершини якого відповідають позиціям, а дуги — ходам. Такий розгляд часто дозволяє істотно спростити аналіз.

Розглянемо дві задачі, які можна розв'язати за допомогою одного й того самого алгоритму.

Приклад 20.3 (камінці, гра "баше"). На столі лежить *n* камінців. Гравці роблять ходи по черзі, кожний з них при своєму ході може забрати зі столу 1, 2 або 3 камінці. Програє той, хто забирає останній камінець.

Приклад 20.4 ("ферзя в кут"). На будь-якому полі шахової дошки стоїть ферзь. Він може переміщуватися на будь-яку кількість полів по вертикалі, горизонталі і діагоналі, але лише вправо та вгору. Можливі ходи показані на рис. 20.6.

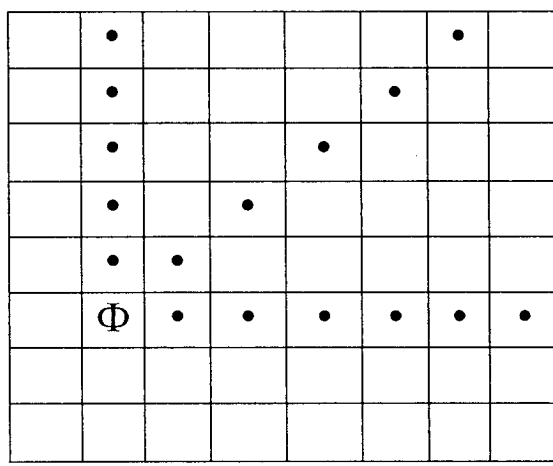


Рис. 20.6. Можливі ходи у грі "ферзя в кут"

Гравці роблять ходи по черзі. Програє той, хто перший поставить ферзя в правий верхній кут. Потрібно для будь-якого початкового положення ферзя визначити, хто виграє: той, хто починає гру, чи його суперник.

Можна переконатися, що побудова дерев для цих ігор є дуже копіткою справою. З іншого боку, легко побачити, що в цих деревах більшість позицій повторюються. Це наводить на думку про можливість застосування аналізу відповідного **графу**. Для простоти ми розглядаємо лише ациклічні орієнтовані графи, хоча в принципі цю вимогу можна послабити.

Алгоритм є хвильовим, і його основна ідея ґрунтується на динамічному програмуванні. Кожна вершина, починаючи із завершальних, повинна досягти певної мітки. Ця мітка означає, чи є дана позиція виграшною (*"В"*) чи програшною (*"П"*) **для гравця, який повинен робити хід у цій позиції**.

Наведемо схему розстановки міток у тому вигляді, в якому вона сформульована в [74].

1. *Завершальні позиції набувають міток "В" або "П" відповідно до правил гри.*
2. *Якщо з позиції X гравець змушений піти до позиції з міткою "В" (тобто до позиції, виграшної для суперника), то позиція X набуває мітки "П".*
3. *Якщо з позиції X гравець має можливість піти до позиції з міткою "П" (тобто до позиції, програшної для суперника), то позиція X набуває мітки "В".*

Застосуємо цей алгоритм до гри в "камінці". Кожна позиція характеризується кількістю камінців m. Завершальною є позиція $m = 0$. Вона є виграшною для гравця, який формально повинен робити хід — його суперник забрав останній камінець, отже, програв. Тому ця позиція набуває мітки *"В"*.

З позиції $m = 1$ гравець має єдиний хід — до позиції $m = 0$, яка має мітку *"В"*. Отже, позиція $m = 1$ набуває мітки *"П"*.

З позицій $m = 2, 3, 4$ можливий хід до мітки *"П"*. Отже, вони набувають мітки *"В"*. Подальший аналіз повністю аналогічний (наведемо його результати для m від 0 до 14).

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14
В	П	В	В	В	П	В	В	В	П	В	В	В	П	В

Застосування цього ж алгоритму до задачі "ферзя в кут" приводить до такої розстановки міток:

В	В	В	В	В	В	П	В
В	В	В	В	В	В	В	П
В	В	В	В	В	П	В	В
В	В	П	В	В	В	В	В
П	В	В	В	В	В	В	В
В	В	В	В	П	В	В	В
В	В	В	В	В	В	В	В
В	В	В	П	В	В	В	В

Рис. 20.7. Розв'язок задачі "ферзя в кут"

Алгоритм припускає узагальнення на випадок, якщо виграші і програші можуть мати різні числові значення.

Кожна позиція n у процесі аналізу повинна набути мітки $\lambda(n)$ — виграш гравця, який має право ходу в цій позиції. Аналіз починається від завершальних позицій, мітки яких розставляються відповідно до правил гри.

Нехай $G(n)$ — множина всіх наступників вершини n; нехай всі вони вже проаналізовані. Гравець при своєму ході намагається залишити суперника в найневигіднішій ситуації і тому повинен перейти до позиції-наступника з мінімальною оцінкою. Але, оскільки оцінки наступників є виграшами суперника, ця мінімальна оцінка береться з протилежним знаком. Тому оцінка позиції виражається через оцінки позицій-наступників за формулою:

$$\lambda(n) = -\min_{m \in G(n)} \lambda(m).$$

20.8. Огляд сучасних шахових програм

Можливості сучасних шахових програм постійно зростають. Ще 10—15 років тому про рівну гру шахових програм з провідними гросмейстерами практично не могло бути мови. Сьогодні ж ситуація швидко змінюється. Такі програми, як **Hiarch, Rebel, Mchess** та інші при виконанні на процесорах **Pentium** показують досить високу якість гри. Можна згадати про спеціалізований шаховий комп'ютер з серії **RS/6000**, який влітку 1997 р. в шаховому матчі з 6 партій виграв у Каспарова з рахунком 3,5 на 2,5. При цьому партії передбачали нормальний контроль часу, прийнятий у турнірах за участю людей.

Слід зазначити, що такого прогресу було досягнуто не стільки завдяки принципово новим алгоритмам, скільки завдяки зростанню швидкодії комп'ютерів. Серед практичних прийомів, які підвищують силу шахових програм, можна відмітити такі:

- наявність **дебютних** та **ендшпільних бібліотек**, які часто дозволяють зробити хід, передбачений теорією, взагалі без аналізу;
- **"служба кращих ходів"**, яка зберігає інформацію про ходи, які за певних ситуацій виявлялися найкращими;
- **метод форсованих варіантів**, який полягає в тому, що після досягнення максимальної глибини перебору аналіз продовжується, але розглядаються не всі ходи, а лише форсовані. Як правило, це шахи та взяття фігур. Метод підвищує якість комбінаційної гри та дозволяє різко скоротити ймовірність грубих помилок.

Регулярно проводяться чемпіонати світу серед шахових програм. Перший з них відбувся у 1974 р., і його переможцем стала програма "**Каїса**" (тодішній СРСР).

Останнім часом шахові програми інтенсивно використовуються для аналізу шахових закінчень. При цьому застосовується ***ретроспективний аналіз***, ідея якого описана в п. 20.7. Цей аналіз іде в зворотному порядку від матових позицій або від позицій з відомою оцінкою.

За допомогою шахових програм було повністю проаналізовано ряд ендшпілів, наприклад *"ферзь з пішаком проти ферзя"*, *"тура і кінь проти тури"* та ін. Часто такий комп'ютерний аналіз змінював попередню теоретичну оцінку. Наприклад, закінчення *"тура і кінь проти тури"* в теорії вважалося нічийним, але комп'ютер виявив великий процент виграшних

позицій. У Шаховому кодексі є правило, згідно з яким партія закінчується в нічию, якщо протягом 50 ходів не було жодного взяття і жоден пішак не зробив ходу. На основі детального комп'ютерного аналізу було встановлено, що для зміни матеріального співвідношення у багатьох закінченнях 50 ходів не вистачає. Були виявлені позиції, для яких виграш досягається за кілька сотень ходів.

Шахові програми використовуються також для перевірки шахових задач, в яких вимагається поставити мат за визначену кількість ходів. Завдяки цьому у багатьох таких задачах були виявлені помилки.

20.9. Деякі підходи до самонавчання ігрових програм

Дуже цікавою є програма для гри в шашки, створена А. Семюелем. Головною метою роботи А. Семюеля була не стільки гра, як вивчення можливостей самовдосконалення і самонавчання ігрових програм [244, 282].

Ідея одного з підходів А. Семюеля полягала у такому. У найпростішому випадку статична оціночна функція являє собою лінійний вираз:

$$s = k_1a_1 + k_2a_2 + \ldots + k_na_n,$$

де a_i — числові оцінки факторів, що впливають на оцінку позиції, k_i — деякі вагові коефіцієнти. Якість гри визначається цими коефіцієнтами, і тому їх необхідно оптимізувати. Коефіцієнти можна підігнати шляхом пробних ігор, а можна підібрати шляхом безпосереднього аналізу дерева гри.

Якби статична оціночна функція збігалася з мінімаксною оцінкою, така оцінка була б ідеальною, а її отримання було б еквівалентне повному аналізу дерева. Нехай s — статична оцінка даної позиції; q — оцінка, передана знизу; $e = s - q$ — різниця між цими оцінками. Ідея А. Семюеля полягала в адаптивній мінімізації цієї різниці. Він визначав ***коефіцієнти кореляції*** між a_i та e. Якщо кореляція додатня, відповідний коефіцієнт k_i потрібно зменшити, якщо від'ємна — то збільшити. Подібні підходи часто називають ***декореляцією помилок***.

Загальновизнано, що реалізовані у сучасних шахових та інших ігрових програмах варіанти повного перебору є прикладами "нелюдського" підходу до гри: людина ніколи не робить такого перебору, до якого залучається величезна кількість безглуздих ходів (людина і не могла б його зробити при всьому бажанні). Шахіст швидко вловлює основні особливості позиції, він швидко помічає ходи, на які слід звернути увагу в першу чергу. У мозку шахіста виникає ***дерево планів*** [168]. Ці плани з'являються на основі того, що в ході аналізу позиції шахіст має змогу поставити перед собою певні цілі. План має визначену кінцеву мету, тому насамперед аналізуються ходи, які можуть сприяти або завадити реалізації цього плану. Такий підхід деякою мірою реалізовано у програмі ROBIN [168].

Слід відзначити також дослідження М. Ботвинника та його програму "Піонер" [24]. У найзагальніших рисах його підхід ґрунтується на розгляді шахової гри на основі вирішення задачі ***багаторівневого керування з неточною метою***. Шахова гра розглядається як багаторівнева система

керування. Кожна шахова фігура має певну мету, і інші фігури можуть допомагати або заважати їй у досягненні певних цілей. Дані локальні цілі повинні узгоджуватися з глобальнішими цілями (виграш, досягнення матеріальної переваги), і за цим слідкує система керування вищого рівня.

Людська гра спирається на ряд ***евристик***: застосування евристичних правил може різко скоротити перебір і часто дозволяє зробити розумний хід майже без аналізу варіантів. До таких шахових евристик належать, наприклад, вимоги боротьби за центр, швидкого розвитку фігур, своєчасної рокіровки і т. п. Зрозуміло, що стратегія, яка спирається на такі правила, далеко не завжди є оптимальною. Вона може бути субоптимальною або просто раціональною.

Незважаючи на певні досягнення у цій галузі, спроби ухилитися від повного перебору в шахових програмах до цього часу носили експериментальний характер, і ці програми не досягали великих спортивних успіхів.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Поясніть зв'язок між ігровими задачами і плануванням цілеспрямованих дій.
2. Назвіть характерну особливість ігрових задач, які вирізняють їх серед інших задач прийняття рішень.
3. Дайте визначення теорії ігор.
4. Що таке однокрокові антагоністичні ігри?
5. Що таке оптимальна стратегія?
6. Для якого класу ігор можна довести існування оптимальної стратегії?
7. Які ігри належать до детермінованих?
8. Що означає "гра з повною інформацією?"
9. Дайте визначення дерева гри; побудуйте дерево довільної конкретної гри.
10. Що означає поняття "імпліцитне дерево гри"?
11. Опишіть мінімаксну процедуру для знаходження оптимальної стратегії в ігрових задачах.
12. Чому для більшості ігор повний аналіз на основі мінімаксної процедури виявляється неможливим?
13. Назвіть параметри, які впливають на силу ігрових програм.
14. Що таке глибина перебору, яким чином вона впливає на якість гри? Наведіть власний приклад.
15. Як змінюється мінімаксна процедура при обмеженій глибині процедури?
16. Що таке статична оціночна функція? Наведіть відомі вам приклади.
17. Що таке лінійна статична оціночна функція?
18. Опишіть *m*^*n*-процедуру. Як вона співвідноситься з класичною мінімаксною процедурою?

19. Опишіть альфа-бета-процедуру. З чим пов'язана доцільність її застосування?
20. Чи залежить ефективність альфа-бета-процедури від порядку розгляду ходів? Якщо так, наведіть приклади.
21. В якому випадку альфа-бета-відтинання є максимальними?
22. Наведіть теоретичну оцінку максимального скорочення перебору при альфа-бета-відтинанні.
23. Наведіть відомі вам рекомендації до упорядкування ходів-кандидатів. З якою метою здійснюється таке упорядкування?
24. Що таке фіксоване і динамічне упорядкування?
25. Опишіть алгоритм розстановки поміток для аналізу ігрових задач.
26. Перелічіть відомі вам шахові програми.
27. Перелічіть напрями використання шахових програм.
28. Опишіть відомі вам прийоми підвищення якості шахових програм.
29. Охарактеризуйте підхід А. Семюеля до самонавчання ігрових програм.
30. Чим відрізняється підхід до аналізу гри, який використовується в ігрових програмах, від того, який здійснюється людиною?

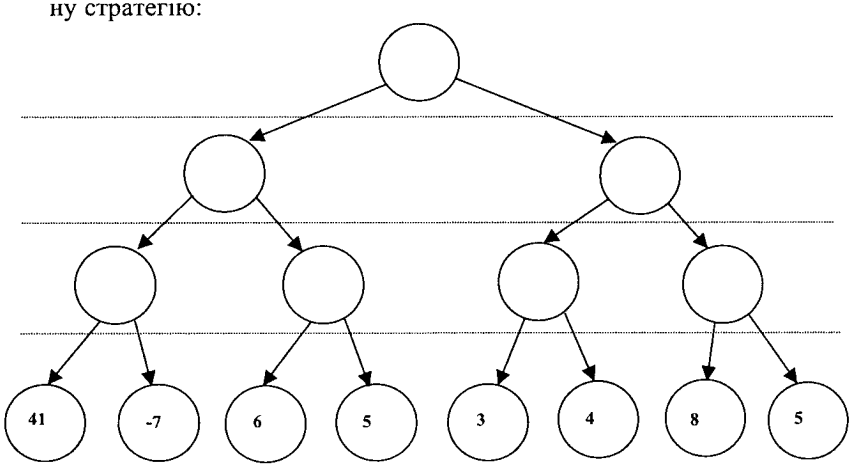
ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Наведіть власний приклад використання рефлексії в іграх.
2. Як можна описати ігрові задачі в термінах І-АБО-графів?
3. Сформулюйте деякі умови, при виконанні яких використання статичних оціночних функцій гарантує той самий результат, що й аналіз на повному дереві.
4. Наведіть власні приклади ігор, які можна аналізувати на основі графових моделей.
5. Охарактеризуйте сучасний стан ігрових програм, зокрема шахових. Які ви бачите шляхи до підвищення їх якості?
6. Як можна передати ігровим програмам здатність до аналізу, подібного до того, який здійснює людина?
7. Як можна застосувати до ігрових програм методики, характерні для роботи з недостовірними та нечіткими знаннями?

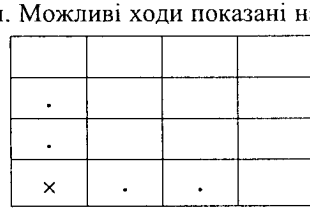
ЗАДАЧІ І ВПРАВИ

1. Як зміниться процедура знаходження оптимальної стратегії для гри, для якої заздалегідь відомо, що статичні оцінки завжди збігаються з мінімаксними (тобто отриманими в результаті мінімаксної процедури)?
2. Доведіть, що мінімаксні оцінки всіх позицій, що лежать на оптимальному варіанті, повинні збігатися між собою.
3. Доведіть, що для кожної вершини будь-якого дерева гри існує хоча б один шлях, який починається в даній вершині і закінчується в одній з листових вершин, такий, що мінімаксні оцінки всіх вершин, які лежать на цьому шляху, збігаються між собою.

4. Проаналізуйте пропоноване дерево гри та знайдіть для нього оптимальну стратегію:



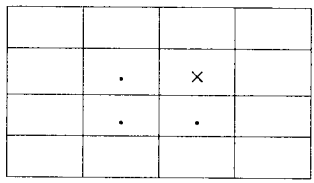
5. Грають двоє. Кожний гравець при своєму ході може покласти на стіл одну або дві монети. Всі монети мають однакову вартість. Гравці роблять ходи по черзі, один за одним. Кожний робить по два ходи, після чого гра закінчується. Якщо після останнього ходу кількість монет на столі кратна 2, виграє перший гравець, якщо кратна 3 — другий. Переможець забирає собі всі монети. За допомогою мінімаксної процедури визначте, як повинні грати гравці, якщо кожний з них прагне максимізувати свій виграш (отримати максимальну кількість монет).
6. Чи є вірним твердження *"альфа-бета-відтинання не можуть відбуватися на вершинах оптимального варіанта"*?
7. Покажіть на типовому дереві гри позиції, які аналізуються за максимальних альфа-бета-відтинань.
8. Придумайте власний приклад, який ілюструє переваги динамічного упорядкування варіантів ходів порівняно з фіксованим.
9. Гра відбувається на шахівниці 8 × 8. Гравці роблять ходи по черзі. Кожний з них має право пересунути шахову туру у напрямку правого верхнього кута (вгору по вертикалі або вправо по горизонталі), але не більше ніж на два поля. Можливі ходи показані на рисунку.



Хто перший поставить туру у правий верхній кут, той програє.

На початку гри тура стоїть у лівому нижньому кутку шахівниці. За допомогою алгоритму розстановки міток визначити, хто виграє — той, хто починає гру, чи його суперник.

10. Гра відбувається на шахівниці 8 × 8. Гравці роблять ходи по черзі. Кожний з них має право пересунути шахового короля у напрямку лівого нижнього кута (на одне сусіднє поле, але лише вліво, вниз, або вліво та вниз по діагоналі). Можливі ходи показані на рисунку.



При своєму ході гравець може зробити за своїм бажанням один, два або три ходи підряд. Хто перший поставить короля у лівий нижній кут, той виграє.

На початку гри король стоїть у правому верхньому кутку шахівниці. За допомогою алгоритму ретроспективної розстановки міток визначити, хто виграє — той, хто починає гру, чи його суперник.

Розділ 21
ЕВРИСТИЧНІ АЛГОРИТМИ

Раз отрежу, да семь раз
потом отмерю.
З пісні

21.1. Обмежуючі правила та евристики як засіб скорочення перебору

Евристичні підходи до вирішення задач людина використовувала з давніх часів. Короткий оксфордський словник англійської мови трактує це слово так: "Евристика — мистецтво знаходження істини".

При розгляді задач, що належать до класу *NP*, важливе місце посідає знаходження варіантів розв'язку цих задач для певних наборів даних (але не для всіх даних!) за поліноміальний час. Для цього досліднику бажано підказати шлях, який звільняє від необхідності, базуючись на тих особливостях, що характеризують задачу розгляду. Ці особливості і називають евристиками, а правила їх використання називають евристичними правилами. Евристичні правила можуть бути як дуже простими, так і досить складними. Прикладом простої евристики можуть служити прислів'я. Багато з них можна розглядати керівництвом до дії в повсякденному житті: "Сім разів відміряй, а потім відріж." Оскільки евристичні правила носять рекомендаційний характер, то евристичні методи не завжди приводять до бажаних результатів розв'язку задачі. Отже, бажано мати якомога більший набір евристик.

Для глибшого розуміння евристичного підходу розглянемо два типи задач: добре визначені задачі і погано визначені задачі [282].

Добре визначеною задачею називається така задача, для якої з використанням заданого можливого розв'язку можна застосувати алгоритмічний метод визначення того, чи буде він шуканим розв'язком. Прикладом такої задачі може служити задача інтегрування алгебраїчного виразу відносно однієї із змінних, що входять до нього. Процес інтегрування не є в загальному випадку алгоритмічним. Але перевірку інтегрування можна провести шляхом диференціювання по тій же змінній. Оскільки останнє є алгоритмічним процесом, маємо: інтегрування належить до добре визначених проблем.

Більшість задач повсякденного життя є погано визначеними: вибирається деяка послідовність дій, для якої немає впевненості, гарантії, що вона навіть в перспективі буде найоптимальнішою за даних обставин. Наприклад, задачу вибору ходу в шахах, незважаючи на чітко виражений кількісний характер вхідних даних, потрібно розглядати як погано визначену задачу.

Для добре визначених задач традиційно існує деякий алгоритмічний метод їх розв'язку (за винятком алгоритмічно нерозв'язних задач). Для них можна визначити *простір розв'язків*, що містить істинний розв'язок. Так, у випадку задачі інтегрування таким простором буде простір розв'язків, що містить усі можливі алгебраїчні вирази наперед визначеної довжини. Коли є можливість визначити спосіб його перегляду, кажуть, що добре визначена задача може бути в принципі вирішена алгоритмічно. Складність полягає в тому, що при повному переборі варіантів розв'язку рішення є нереальним внаслідок дуже великого простору розв'язків. Звідси отримуємо ще одне визначення евристики [55]:

> ***Евристика*** *(евристичне правило, евристичний метод) є довільним правилом, стратегією, хитрістю, спрощенням або якимось іншим засобом суттєвого зменшення, обмеження обсягу пошуку розв'язку в значних просторах розв'язку.*

Згідно з цим визначенням потрібне вірне адміністрування побудови розв'язку. Під останнім розуміється метод, який відповідно до евристичного критерію серед кількох можливих підходів до розв'язку вибирає найоптимальніший на даному кроці. Інакше кажучи, адміністративна частина алгоритму повинна бути в змозі дати оцінки складності підзадач та їх корисності, а також використовувати ці оцінки для планування процесу розв'язку задач. Отже, евристичний метод є корисним при встановленні порядку, який не обов'язково є оптимальним. Цей порядок, з великою вірогідністю, повинен виявитися набагато кращим від випадково обраного порядку.

Серед загального огляду евристик важливе місце посідає основна евристика — принцип навчання, сформульований Мінським і Селфріджем [318]. Цей принцип формулюється так: "У новій ситуації спробуйте використати методи, що найкраще працювали в аналогічних відомих ситуаціях попередньо". В такому разі важливим є вибір критерію подібності задач,

ситуацій, складною. А ця проблема є досить складною. Для визначення міри подібності задач Мінським було введено поняття евристичного зв'язку задачі [317]. Ця ідея приводить до важливих висновків відносно природи інтелекту і досить поширена при вирішенні задач штучного інтелекту.

Ми пересвідчилися, що повний перебір або перебір з поверненням дозволяє у ряді випадків прийняти деяке рішення. Але часто реалізація цих методів забирає дуже багато часу і тому є неможливою чи недоцільною. Тому за таких ситуацій вдаються до *обмежуючих правил* та *евристик* як до типових засобів скорочення перебору. Саме так у більшості випадків і діє людина при плануванні своїх дій.

Обмежуючим правилом *при плануванні цілеспрямованих дій називається правило, якому повинні бути підпорядковані альтернативні дії, що розглядаються.*

Тобто, застосування обмежуючих правил дозволяє включати до перебору не всі можливі дії, а лише ті, які не суперечать таким правилам. Це у ряді випадків дозволяє різко скоротити перебір, а інколи навіть звести задачу до поліноміальної. Зокрема, описані вище жадібні алгоритми можна розглядати як застосування обмежуючих правил.

Природа обмежуючих правил може бути різноманітною. Інколи вдається довести, що обмежуюче правило дозволяє відкинути явно безперспективні гілки і досягти того ж результату, що й повний перебір без застосування цього правила. Такі правила є *теоретично обґрунтованими* і повинні безумовно застосовуватися.

Але частіше доводиться зустрічатися з ситуацією, коли обмежуючі правила спираються на наявний апріорний досвід, але не обґрунтовані теоретично, і не гарантують або просто не дозволяють отримати оптимальне рішення. Застосування цих правил дозволяє скоротити перебір, але за рахунок втрати гарантованої оптимальності (власне, у багатьох випадках нічого кращого і не залишається). Отримуємо ще одне визначення евристики.

Евристикою при плануванні цілеспрямованих дій *називається обмежуюче правило, яке спирається на наявний досвід і не гарантує оптимальності рішення.*

Часто трапляється, що існує не одна евристика, а декілька, і вони повністю або частково суперечать одна одній. Тоді окремою проблемою стає проблема застосування евристик.

Розглянемо практичне застосування описаної методології на прикладі вирішення задачі розфарбування графу.

21.2. Задача розфарбування графу

У XIX ст. вперше постало питання про те, якою мінімальною кількістю фарб можна розфарбувати топологічну карту. Тоді з'явилася ідея розглядати карту як планарний граф, і так з'явилася проблема розфарбу-

вання графу мінімально можливою кількістю фарб [265]. Потім виявилося, що проблема розфарбування не лише планарного, а й довільного графу (звичайно, з деякими обмеженнями) є надзвичайно актуальною. Задача розфарбування графу знаходить безліч застосувань у комбінаториці, фізиці, економіці, програмуванні. До задачі розфарбування графу можна звести такі задачі, як класифікація об'єктів, розподіл ресурсів, складання розкладу, розподіл пам'яті. Розглянемо детальніше цю проблему та методи її розв'язання [8, 104, 110, 111, 154]. У своєму дослідженні розглядатимемо лише зв'язні неорієнтовані графи без петель і паралельних ребер.

Нехай маємо $G = (V, E)$ — неорієнтований граф, де $V = \{v_1 \ldots v_n\}$ — множина вершин графу. Число n далі називатимемо кількістю вершин графу. Розфарбуванням вершин графу G називатимемо відображення φ множини V на множину C таку, що $C = (c_1 \ldots c_k)$, і $k = n$. Елементи множини C називатимемо *фарбами*, а число m визначатиме *кількість фарб*, використаних при розфарбуванні графу G.

Вірним розфарбуванням графу G називатимемо таке розфарбування вершин, для якого завжди виконується $(x_i, x_j) \in E \Rightarrow \varphi(x_i) \neq \varphi(x_j)$, тобто якщо дві вершини суміжні, то вони пофарбовані різними фарбами.

Вірне m-розфарбування розбиває множину V вершин графу на m підмножин $V_1, \ldots, V_m$ (або класів), що не перетинаються та жодна з яких не містить суміжних між собою вершин. Очевидно, що ці множини будуть незалежними. Кожна з них містить вершини, розфарбовані певною фарбою. Такі множини називаються *хроматичними класами* графу G.

Мінімальним розфарбуванням називатимемо будь-яке вірне розфарбування, кількість фарб в якого не більша за будь-яке інше вірне розфарбування. З такого визначення випливає, що кількість фарб для всіх мінімальних розфарбувань є однаковою. Таку найменшу кількість фарб, за якої граф має мінімальне розфарбування, прийнято називати *хроматичним числом* графу. Воно є характеристикою графу і позначається $\chi(G)$, або ж просто χ.

Називатимемо граф *k-розфарбованим*, якщо $\chi(G) \neq k$, та *k-хроматичним*, якщо $\chi(G) = k$.

Для знаходження хроматичного числа графу недостатньо знати кількість його вершин і ребер, як і недостатньо знання розподілу степенів вершин. Однак за цими характеристиками можна знайти верхню і нижню оцінку хроматичного числа графу. Тепер постає питання, яким саме чином треба діяти, щоб отримати мінімальне розфарбування графу та визначити хроматичне число графу [154].

Отже, маємо окреслене коло проблем, пов'язаних з поняттям розфарбування графу. Задача розфарбування графу складається з двох частин. По-перше, її можна трактувати як задачу знаходження хроматичного числа графу. По-друге, задача розфарбування графу може бути поставлена як знаходження хоча б одного мінімального розфарбування графу. Власне, ці дві підзадачі і формують задачу розфарбування графу. Вони пов'язані між собою, оскільки якщо ми маємо гарантовано мінімальне розфарбування гра-

фу G, то можемо дуже просто знайти його хроматичне число $\chi(G)$. Якщо ж ми якимось чином знайшли хроматичне число графу $\chi(G)$, то, очевидно, можемо знайти правильне розфарбування графу G за допомогою $\chi(G)$ фарб. Іноді розглядають і часткові випадки задачі розфарбування графу. Наприклад, розфарбування графу у наперед визначені кольори. Для кожної вершини відомі ті кольори, якими можна її розфарбувати, і ті, якими її розфарбувати не можна. Тобто до класичної задачі розфарбування додаються певні обмеження щодо співвідношення кольорів. Але її вирішення можна природно отримати з розв'язку базової задачі.

Існує також добре відома проблема чотирьох фарб для планарних графів.

Планарний граф *— це граф, який можна зобразити на площині так, що різним вершинам відповідають різні точки площини, ребрам — дуги (без самоперетинань), і жодні два ребра не мають спільних точок, крім інцидентної їм обом вершини.*

Постає питання, якою мінімальною повинна бути кількість фарб розфарбування такого графу. Ще в XIX ст. було висунуто знамениту *гіпотезу 4-х фарб* — будь-який планарний граф можна правильно розфарбувати чотирма фарбами, тобто $\chi(G) \le 4$, якщо G — планарний граф. Але її вдалося підтвердити на практиці лише у XX ст. за допомогою ЕОМ [97].

Також іноді кажуть про *реберне розфарбування* графу. У даному разі мають справу з відображенням множини ребер на множину фарб, а вірним розфарбуванням називають розфарбування, за якого будь-які два ребра, що інцидентні одній вершині, пофарбовані різними фарбами.

Далі будуть проаналізовані точні та евристичні методи розфарбування графу.

21.3. Точні та евристичні методи розфарбування

Зауважимо лише те, що точні методи розфарбування графу дають в результаті точне хроматичне число, але вони не є ефективними. Якщо ж користуватися евристичними методами розфарбування, то не можна сподіватися на ідеально точне знаходження хроматичного числа, але ми можемо знехтувати похибкою обчислень на користь ефективнішого алгоритму.

Якщо граф G має n вершин, то, очевидно, що він має n-розфарбування, і $\chi(G)$ -розфарбування, і, крім того, має k-розфарбування для будь-якого k такого, що $\chi(G) \le k \le n$. Легко побачити, що хроматичне число повного графу з n вершинами дорівнює n. Хроматичне число парного циклу дорівнює двом, непарного циклу — трьом. Хроматичне число пустого графу (в якого множина ребер E пуста) дорівнює 1. Причому граф є 1-хроматичним тоді і тільки тоді, коли він пустий. Але ці очевидні закономірності не дають можливості оцінити, яким буде хроматичне число. Відомі теоретичні дослідження відносно різних типів графів, що мають значний інтерес [104, 110, 154].

Теорема 21.1. (теорема Кенінга). *Граф є біхроматичним (2-хроматичним) тоді і тільки тоді, коли він не містить непарних простих циклів.*

Звідси випливає, що будь-яке дерево є біхроматичним і будь-який дводольний граф є біхроматичним. Але невідомо, як характеризувати k-хроматичні графи, якщо $k = 3$. Такий критерій навіть для $k = 3$ допоміг би вирішити проблему чотирьох фарб.

Тому в теорії розфарбування графів досить важливим є питання оцінки хроматичного числа графу та наближених методів розфарбування графу.

У роботі [265] було виведено кілька верхніх оцінок хроматичного числа графу.

Теорема 21.2. (теорема Секереша-Вільфа). *Для будь-якого графу G*

$$\chi(G) \le 1 + \max_{H \subset G} \min_{v \in H} \deg_H v,$$

де максимум береться по всіх власних підграфах H графу G.

Теорема 22.3. (теорема Уелша-Пауела). *Нехай у графі G вершини впорядковані за спаданням степенів. Тоді*

$$\chi(G) \le \max_{1 \le i \le n} \min\{i, 1 + \deg v_i\}.$$

Цю оцінку буде розглянуто нижче при дослідженні наближених методів розфарбування графу.

Як і багато інших комбінаторних задач, задача розфарбування графу є *NP*-повною [97]. Це означає, що її розв'язання потребує експоненційного часу розв'язання, тобто число кроків перебору буде експоненційно зростати залежно від розмірності задачі.

До того ж розв'язання задачі розфарбування графу шляхом безпосереднього перебору варіантів передбачає побудову всіх можливих варіантів розфарбування і вибір з них мінімального правильного розфарбування. Але при цьому породжуватимуться не лише правильні розфарбування. До того ж буде породжено безліч розфарбувань, які можна перетворити одне в одне за допомогою простого перефарбування вершин, тобто, по суті, однакових. Таким чином ми бачимо, що алгоритм прямого перебору не дає нам можливості розв'язати цю задачу за реальний час при великій розмірності вхідних даних.

Проаналізувавши тривіальне рішення, дійдемо висновку, що треба шукати одразу лише правильні розфарбування графу, причому намагатися відразу ж знайти мінімальне розфарбування. Шукаючи алгоритм вирішення задачі, ми відмовимося від можливості знаходження оптимального розв'язку і будемо замість цього знаходити розв'язок, близький до оптимального, за прийнятний час. Алгоритми, засновані на такому принципі, називають евристичними, бо вони використовують різноманітні вдалі міркування без суворого обґрунтування.

Точні алгоритми розфарбування знаходять точне хроматичне число графу і мінімальне розфарбування. При цьому, звичайно, вони мають більшу складність, ніж наближені алгоритми, але меншу, ніж мав би простий перебір.

Існує багато підходів скорочення обсягів перебору, хоча при цьому і не заперечується експоненційна часова складність алгоритмів. До таких підходів належать, наприклад, метод "гілок і обмежень" або метод "неявного перебору", які полягають у побудові часткових розв'язків, представлених у вигляді дерева пошуку, і застосуванні потужних методів побудови оцінок, що дозволяють розпізнати безперспективні часткові розв'язки, в результаті чого на одному кроці від дерева пошуку відсікається ціла гілка. До інших методів належать метод динамічного програмування, розбиття графу на максимальні r-підграфи. Нижче наведений один з точних алгоритмів розфарбування.

На першому кроці визначаються нижня оцінка хроматичного числа, що дорівнює кількості вершин в одній з клік графу, і верхня оцінка хроматичного числа шляхом розфарбування вершин наближеним методом розфарбування (або ж визначається за однією з формул знаходження верхніх оцінок).

Якщо верхня оцінка дорівнює нижній, то точне розфарбування знайдено; якщо ні, то робиться спроба розфарбувати вершини графу у задану кількість кольорів $\chi_я = \chi_н + [(\chi_в - \chi_н)/2]$, де $\chi_н$ — нижня оцінка, $\chi_в$ — верхня оцінка за допомогою такого наближеного алгоритму.

Нехай k_i — номер кольору i-ї вершини, $N_n = k_1, k_2, \ldots, k_n$, де n — значне число, що відповідає розфарбуванню вершин графу. Алгоритм побудований на монотонному обчисленні N_s з N_{s-1}. Алгоритм дає або правильне розфарбування у M кольорів, або інформацію про неможливість такого розфарбування.

Якщо спроба вдалася, то змінюється значення верхньої оцінки $\chi_в = \chi_я$, інакше змінюється нижня оцінка $\chi_н = \chi_н + 1$. На кожній ітерації другого кроку алгоритму інтервал між нижньою і верхньою оцінками ділиться навпіл, і в цій точці знаходяться нові оцінки.

Враховуючи, що задача розфарбування графу є NP-повною задачею, це усуває можливість використання алгоритму прямого перебору для її розв'язання за реальний час.

Однак існують наближені алгоритми розв'язання цієї задачі, що мають не більше як квадратичну часову оцінку, але не гарантують оптимального розв'язання задачі. Кожен з таких алгоритмів базується на певній евристиці, і будь-яке додавання евристики до алгоритму може його або ускладнити, але наблизити до оптимальності, або ж зробити ефективнішим, але за рахунок пошуку рішення, близького до оптимального.

Намагаючись знайти алгоритм з не більше як квадратичною часовою оцінкою, ми доходимо висновку, що на розфарбування кожної з n вершин графу ми можемо витратити не більш як n кроків. Так ми приходимо до ідеї такої організації алгоритму, щоб не перефарбовувати вершини.

Прийнявши таке рішення, ми маємо дві альтернативи — починати з вершин або з фарб. Перший шлях — це брати певну фарбу і фарбувати нею вершини в якомусь певному порядку. Коли не залишиться вершин, які можна фарбувати цією фарбою, брати наступну, і так далі до тих пір, поки

не будуть розфарбовані всі вершини в графі. Другий шлях — це брати по черзі вершину і вибирати за якимось критерієм для неї фарбу, якою її можна розфарбувати.

Придивимося уважніше до цих варіантів. Йдучи першим шляхом, ми на кожному кроці маємо множину розфарбованих вершин V_c (на першому кроці ця множина буде пустою) і множину нерозфарбованих вершин, з яких ми і шукаємо наступну вершину — кандидата на розфарбування поточною фарбою α. Множина розфарбованих вершин має підмножину вершин $V(\alpha)$, розфарбованих фарбою α. Множина нерозфарбованих вершин, у свою чергу, містить множину тих вершин, які можна розфарбувати фарбою α (умовно позначимо її V^{∂}), і множину тих вершин, які не можна цією фарбою розфарбувати (позначимо цю множину $V^{з}$). Причому ця остання множина, очевидно, містить вершини, суміжні хоча б з однією вершиною з $V(\alpha)$. Тобто на черговому кроці ми або беремо вершину з V^{∂} і розфарбовуємо її фарбою α, або ж, якщо множина V^{∂} дозволених для розфарбування фарбою α вершин пуста, переходимо до наступної фарби. Алгоритм завершується, коли всі вершини графу будуть розфарбованими, тобто коли $V_c = V$.

Однак такий підхід має певні недоліки. Щоразу ми маємо маніпулювати із множиною $V(\alpha)$ і з усіма її околами 1-го порядку з множинами V і V_c.

При другому підході, розфарбувавши вершину v, ми вже не можемо розфарбовувати вершини з її околу першого порядку $R_1(v)$. Аналізуючи вершину, ми шукаємо фарбу, якою ми можемо розфарбувати цю вершину, і якщо не знаходимо її, то використовуємо нову фарбу. Очевидно, що, користуючись таким методом, ми маємо вибирати вершини в певному порядку. Визначення цього порядку дає нам ще кілька альтернатив організації алгоритмів.

Таким чином, ми вже маємо два різні підходи до розфарбування і можемо їх використати для розв'язання нашої задачі. Але можна піти ще далі і за допомогою подальших роздумів додати ще евристики. Уважніше поставимося до правил формування множини $V^{з}$. Вона складається з нерозфарбованих вершин, що належать околам 1-го порядку вершин з множини $V(\alpha)$. Оскільки всі вершини з $V(\alpha)$ є вже розфарбованими, то в нашому подальшому аналізі ми їх використовуємо лише для визначення суміжних з ними вершин. Якщо розглядати всі вершини з $V(\alpha)$ як одну деяку вершину w, ми матимемо $V^{з} = R_1(w)$, а це вже є значним спрощенням процедури визначення обмежень вибору наступної розфарбовуваної вершини. Таким чином, ми нібито "склеюємо" всі вершини з $V(\alpha)$. Робитимемо ми це послідовно: маючи вершину v, пофарбовану фарбою α, ми знаходимо у множині V^{∂} вершину v', і склеюємо вершини v та v', заміняючи процедуру призначення вершині певної фарби попарним склеюванням несуміжних вершин.

Склеювання двох вершин v_1 і v_2 полягатиме у вилученні з графу вершин v_1 та v_2 разом з інцидентними їм ребрами та додаванні вершини v та

інцидентних їй ребер так, щоб її окіл 1-го порядку був об'єднанням околів 1-го порядку вершин v_1 та v_2, тобто, $R_1(v) = R_1(v_1) \cup R_1(v_2)$. В результаті отримаємо граф $G' = (V', E')$, де потужність множини V' є на 1 меншою, ніж у попередньому графі, і ребер також може бути менше за рахунок взаємного поглинання ребер. Поглинання ребер отримуємо, якщо якась вершина u була суміжною з обома вершинами, що склеюються, і в результуючому графі залишиться замість двох ребер (v_1, u) та (v_2, u) одне ребро (v, u). Таке відбувається для вершин, які знаходяться одна від одної на відстані 2, тобто, коли $v_1 \in R_2(v_2)$ та $v_2 \in R_2(v_1)$. Тут під околом 2-го порядку певної вершини розумітимемо множину всіх вершин, які знаходяться від даної на відстані 2. Таку вершину u називатимемо *розділяючою* вершиною. Кількість ребер, що зникають, дорівнюватиме кількості розділяючих вершин.

Перетворення кожного чергового графу G відбуваються доти, доки в ньому не залишиться жодної пари несуміжних вершин, тобто доки граф G не виявиться деяким k-повним графом. Як уже зазначалося, k-повний граф має лише одне тривіальне розфарбування: кожній з k вершин приписується одна з k фарб, і ця фарба буде фарбою для всіх тих вершин початкового графу, які було вклеєно у дану вершину. Оскільки всі вершини, вклеєні в певну вершину результуючого повного графу, є попарно несуміжними, то отримане розфарбування є допустимим.

Тепер треба довести, що, зводячи розфарбування вершин до їх послідовного склеювання, ми не втрачаємо можливості отримати мінімальне розфарбування графу [111].

Теорема 21.4. *Для будь-якого графу з хроматичном числом χ існує послідовність попарних склеювань вершин, що приводить до χ-повного графу.*

Дві вершини графу G називають *суцвітними*, якщо існує мінімальне розфарбування графу G, за якого ці вершини пофарбовані однією фарбою.

Відомо, що в будь-якому неповному графі G існує принаймні пара суцвітних вершин і якщо граф G' отриманий з графа G склеюванням пари суцвітних вершин, то $\chi(G') = \chi(G)$.

На основі цих ідей маємо право склеювати вершини, зберігаючи хроматичне число початкового графу, і проводити таке склеювання до тих пір, поки граф не стане повним, при цьому в ньому буде $\chi(G)$ вершин.

Один з найпоширеніших алгоритмів розфарбування першого типу — алгоритм Єршова-Кожухіна [111] — базується на евристиках склеювання, розглянутих у попередньому параграфі. Для побудови алгоритму мінімального розфарбування залишається лише вирішити проблему знаходження суцвітних вершин у графі.

Теорема 21.5. (теорема Єршова-Кожухіна). *Для будь-якої вершини v графу G, для якої її окіл 1-го порядку $R_1(v)$ не збігається з множиною $X \setminus \{v\}$, в околі 2-го порядку $R_2(v)$ існує принаймні одна суцвітна вершина.*

Ця теорема дає нам евристику істотного обмеження простору пошуку суцвітних вершин, гарантуючи їхнє місцезнаходження в околі 2-го порядку вершини розгляду. Тоді алгоритм набуде такого вигляду.

Вважатимемо, що вершини графу якимось чином пронумеровані. На черговому кроці береться вершина v та в її околі 2-го порядку шукається вершина w з максимальною кількістю розділяючих вершин. Вершини v та w склеюються, отриману вершину вважають черговою. Процес продовжується доти, доки чергова вершина не стане зіркою, після чого переходять до наступної вершини-незірки, тощо. Процес склеювання вершин закінчується, коли граф стає повним.

Розглянемо два підходи побудови евристичних алгоритмів розфарбування другого типу. Згадаємо теорему 21.2. Ця теорема, надаючи верхню оцінку хроматичного числа графу через максимальний степінь вершини, наштовхує на евристику про те, що треба було б починати розфарбовувати вершини з вершин з більшими степенями. Якщо таких вершин мало, то тоді ми зможемо обійтися меншою кількістю фарб.

Якщо впорядкувати вершини графу за спаданням їх степенів і для кожної чергової вершини розфарбування обирати фарбу з найменшим можливим номером, то отримаємо ще один евристичний алгоритм розв'язання задачі мінімального розфарбування графу. Після того, як буде пофарбовано i вершин, буде витрачено максимум i кольорів, а для розфарбування решти вершин буде потрібно не більше за $1 + \max (\deg v_i)$ кольорів, звідки і отримаємо оцінку з теореми 21.4:

$$\chi(G) \le \max_{1 \le i \le n} \min\{i, 1 + \deg v_i\}.$$

Впорядкування вершин згідно зі спаданням вершин називають ПН-впорядкуванням (Перші Найбільші), а алгоритми, засновані на такому впорядкуванні,— ПН-алгоритмами. Можна відмітити, що кількість фарб, що її потребує ПН-алгоритм, може варіювати через неоднозначність ПН-упорядкування, але вона ніколи не перевищуватиме числа, визначеного за наведеною верхньою оцінкою.

Розглянемо ще одну евристику розфарбування, яка намагається зменшити кількість фарб у попередньому алгоритмі розфарбування графу. Назвемо *степенем насиченості* вершини v у частково розфарбованому графі кількість різних кольорів на суміжних з v вершинах.

На першому кроці впорядковуємо вершини за спаданням (ПН-впорядкування), і першу вершину пофарбуємо першою фарбою. Після цього на кожному кроці формуємо множину вершин K з найбільшим степенем насиченості, обираємо з неї вершину з найбільшим степенем в нерозфарбованому підграфі і розфарбовуємо її найменшим можливим кольором. Діємо так до тих пір, поки в графі не будуть пофарбовані всі вершини.

Для побудови евристичних алгоритмів мінімального розфарбування можуть використовуватися й інші впорядкування.

Для кожного евристичного алгоритму розфарбування існує клас графів, для яких отримане розфарбування не буде мінімальним. Це твердження базується на такій теоремі [97].

Теорема 21.6 (теорема Гері-Джонсона). *Для будь-якого евристичного алгоритму знайдеться такий граф G, що A (G) / χ (G) ≥ 2, де A (G) — кількість фарб, витрачених на розфарбування графу G евристичним алгоритмом, а χ (G) — хроматичне число графу G.*

У роботі [80] був введений ще один клас евристичних алгоритмів — клас окільних алгоритмів. Для алгоритмів з цього класу число затрачених фарб розфарбування будь-якого графу задовольняє оцінці Єршова-Кожухіна [111]. При практичному тестуванні вони також показали добрі результати (число фарб наближалось до хроматичного числа) на графах із класів графів, на яких попередньо розглянуті евристичні алгоритми давали погані оцінки кількості фарб.

Основною евристикою окільних алгоритмів розфарбування є розфарбування графу, а другим типом розфарбування відносно певного порядку околів, на які розбивається множина вершин графу. Задаючи певний порядок вибору цих околів і порядок розфарбування вершин околу, отримують конкретні алгоритми цього класу. Нагадаємо, що околом 1-го порядку вершини називають множину, що включає цю вершину і всі вершини, суміжні з нею.

Першим центром околу вибирається вершина з максимальним степенем у графі. Вершини околу розфарбовуються в такому порядку: щоразу з околу вибирається вершина з максимальною кількістю розфарбованих вершин. Далі центром наступного околу розфарбування вибирається нерозфарбована вершина в графі з максимальною кількістю розфарбованих суміжних вершин, і її окіл розфарбовується в порядку, зазначеному вище. Процедура повторюється, поки всі вершини в графі не буде розфарбовано. Вершина розфарбування фарбується фарбою з мінімальним незадіяним номером серед уже розфарбованих суміжних їй вершин.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте поняття обмежуючих правил та евристик. У чому полягає їх значення при плануванні цілеспрямованих дій?
2. Поясніть, чому жадібні алгоритми можна розглядати як застосування обмежуючих правил (розпишіть відповідні обмежуючі правила в явному вигляді).
3. Опишіть принцип навчання, сформульований Мінським і Селфріджем.
4. Охарактеризуйте поняття добре і погано визначених задач.
5. Для довільної предметної області наведіть приклади обмежуючих правил та евристик.
6. Опишіть задачу розфарбування графу.
7. Опишіть відомі вам евристичні алгоритми для задачі розфарбування графів.

ЧАСТИНА V
БАЗОВІ ПАРАДИГМИ ІНТЕЛЕКТУАЛЬНОЇ ДІЯЛЬНОСТІ

Розділ 22

ЗАГАЛЬНОІНТЕЛЕКТУАЛЬНІ МЕТАПРОЦЕДУРИ

Видатним досягненням сучасної науки стала програма, яка на будь-яке запитання дає будь-яку відповідь.

З першоквітневого випуску "Наркому"

22.1. Базові поняття

Очевидно, будь-яка система не стане інтелектуальнішою, якщо для неї **просто запрограмувати** один або кілька алгоритмів вирішення задач, які традиційно належать до інтелектуальних. Особливо це стосується традиційного комп'ютера, керування яким здійснюється на основі чисто алгоритмічного принципу. Наприклад, написання програми, яка вміє грати в шахи, ніяк не допоможе цьому комп'ютеру грати в шашки, хоча в основу програмування обох ігор закладені однакові принципи. У [222] зазначається, що кількісне накопичення готових алгоритмів і програм не приводить до якісних змін. Такий набір програм нагадує велику бібліотеку. У будь-якій книзі, що зберігається в бібліотеці, містяться якісь конкретні відомості, але внесення до бібліотеки нової книги жодним чином не позначається на цих відомостях і не збільшує "інтелекту" бібліотеки.

Натомість інтелектуальна система повинна мати в своєму розпорядженні загальноінтелектуальні процедури, придатні для вирішення широкого класу інтелектуальних задач. Ці процедури повинні, зокрема, породжувати нові знання на основі існуючих, а також породжувати нові алгоритми вирішення конкретних задач на основі загальних знань та аналізу аналогічних алгоритмів. Такі загальноінтелектуальні процедури прийнято називати *метапроцедурами*.

На відміну від конкретних методів та алгоритмів метапроцедур не так багато. Вони традиційно розглядалися в рамках теорії та практики *ситуаційного керування*, яке вивчає методи керування дуже складними системами [221].

22.2. Поняття про ситуаційне керування

Ситуаційне керування застосовується для вирішення задачі керування дуже складними об'єктами (живі істоти, суспільні групи, фірми та ін.). Для цих об'єктів характерні такі риси:

- практично неможливо отримати точну математичну модель об'єкта і формалізувати мету його функціонування, а відтак — отримати математичні критерії оптимального керування;
- об'єкти керування часто є унікальними, і алгоритми керування, придатні для одного об'єкта, не підходять для іншого;
- об'єкти керування постійно змінюються у часі та просторі;
- наявна інформація завжди з одного боку — неповна, а з іншого — надлишкова;
- принципово квазіалгоритмічний характер керування; керуючі впливи не обов'язково приводять до запланованих результатів; пов'язано це як з неповнотою опису, так і з тим, що об'єкти керування можуть мати свої цілі, які не обов'язково збігаються з цілями системи керування (так звана "свобода волі").

Усі ці фактори приводять до практичної неможливості отримати оптимальні результати. Керування має стати субоптимальним або просто раціональним.

У [221] описана одна з можливих схем ситуаційного керування. У найзагальніших рисах система керування постійно виконує такі операції:

- *оцінка поточної ситуації та віднесення її до одного з класів;*
- *поповнення опису ситуації на основі знань, що містяться в пам'яті системи;*
- *вироблення рішення про те, які дії слід виконати для зміни ситуації в бажаному напрямку;*
- *реалізація прийнятого рішення.*

Однією з характерних особливостей ситуаційного керування є те, що кількість можливих ситуацій набагато перевищує кількість можливих рішень.

Ключовими для класичної теорії ситуаційного керування є семіотичні моделі та мови опису ситуацій.

22.3. Семіотичні моделі та мови опису ситуацій

Семіотичні (знакові) моделі визначаються на основі логічних моделей [221]. Якщо формальна логічна модель задається четвіркою $M = <T, P, A, B>$, де T — множина базових елементів; P — синтаксичні правила; A — система аксіом формальної моделі; B — множина правил виведення, то семіотична модель задається набором $C = <M, h_T, h_P, h_A, h_B>$, де M — формальна логічна модель; h_T, h_P, h_A, h_B — правила зміни відповідно базових елементів, синтаксичних правил, аксіом та правил виведення. За допомогою семіотичних моделей можна виразити як змінність самих ситуацій, так і неоднозначність їх інтерпретації і неоднозначність слів (знаків), які описують ці ситуації.

У [221] сформульовано три гіпотези, покладені в основу мов для опису ситуацій.

Гіпотеза 22.1. *Уся інформація про об'єкт керування та засоби керування може бути виражена засобами природної мови.*

Гіпотеза 22.2. *Будь-який текст природною мовою можна перекласти на формальну мову семіотичної моделі.*

Гіпотеза 22.3. *При відображенні зовнішнього світу людина виокремлює в ньому скінченний набір відношень.*

Зрозуміло, що набори понять і дій повністю визначаються семантикою предметної області, але набори понять, квантифікаторів, оцінок і т. п. досить універсальні. У [221] їх виокремлено близько сотні. На їх основі формуються фрази **мови ситуаційного керування**, за допомогою яких описуються реальні ситуації.

22.4. Основні загальноінтелектуальні метапроцедури

Стимульно-реактивна теорія була однією з перших теорій, призначених для опису поведінки інтелектуальної системи. Вона безпосередньо пов'язана з рядом фізіологічних експериментів, які проводилися над тваринами в першій половині XX ст. В основі стимульно-реактивних моделей лежить ***біхевіористичний*** підхід, який можна в найзагальніших рисах охарактеризувати таким чином.

Поведінка системи *визначається її поточним станом, але поточний стан системи змінюється в процесі навчання під впливом заохочень та покарань, які вона отримує від зовнішнього середовища.*

По суті, в рамках даного підходу йдеться про **формування умовних рефлексів**. Система, що сприймає деякий подразник, відповідає на нього певною реакцією. Якщо ця реакція була вірною, система отримує "заохочення", яке підкріплює її впевненість у правильності цієї дії. Якщо ж дія була неправильною, система дістає "покарання".

Однією з класичних моделей, які описують поведінку системи в рамках стимульно-реактивної теорії, є *автомати з лінійною тактикою*, які будуть описані в розд. 23.

Необхідно зазначити, що стимульно-реактивний підхід не є єдиним до моделювання процесу формування умовних рефлексів. Зараз інтенсивно розвивається підхід, пов'язаний із застосуванням *нейронних мереж* (див. частину 7).

Ми вже бачили, що розв'язання багатьох задач прийняття рішень зводиться до одних і тих самих формалізмів, а саме — формалізмів евристичного пошуку в просторі задач або просторі станів. Насамперед формальна модель задачі будується як *лабіринт можливостей*, в якому слід здійснювати пошук. Таким лабіринтом можна розглядати граф станів або І-АБО-граф. Тоді згідно з [222] одну з ключових метапроцедур функціонування інтелектуальної системи можна охарактеризувати так: прийняття рішень інтеле-

ктуальною системою являє собою евристичний пошук у лабіринті можливостей. Лабіринт можливостей визначає можливі дії системи, і в процесі перебору цих можливостей інтелектуальна система знаходить оптимальний або припустимий шлях до мети.

Лабіринтна теорія передбачає попереднє задання самого лабіринту можливостей та правил його формування. Але правила формування лабіринту, який по суті являє собою деяку модель, тобто певний формалізований опис даної ситуації, можуть визначатися самою системою. Модельний опис може бути не єдиним. Часто трапляється, що задача не вирішується в одному лабіринті можливостей, але може бути легко вирішена в іншому. Тоді на перший план виходить проблема правильного вибору модельного опису. На основі цього була сформульована *модельна гіпотеза,* згідно з якою центральною метапроцедурою інтелектуальної діяльності є побудова такого лабіринту можливостей, в якому можна здійснити ефективний пошук рішення.

Можна стверджувати, що *зіставлення зі зразком* є однією з найфундаментальніших загальноінтелектуальних процедур. Воно має велике значення для міркувань за аналогією, розпізнавання зразків, інтуїтивного мислення тощо. Ми вже бачили приклад застосування зіставлення зі зразком для організації фатичного діалогу.

Спробуємо формалізувати поняття зіставлення зі зразком. Нехай M — набір зразків, що зберігаються в пам'яті інтелектуальної системи або можуть бути породжені системою, F — деяка множина функцій або алгоритмів. Тоді під зіставленням елемента q, що спостерігається інтелектуальною системою, зі зразком розумітимемо знаходження елемента $m \in M$ і алгоритму $f \in F$, таких, що $f(q) = m$ або $f(m) = q$.

Зрозуміло, що вибір елементів m і f може бути не єдиним, тобто результат зіставлення виявляється неоднозначним. Можна ввести поняття якості зіставлення, яке задаватимемо деякою функцією $r(a, b, f)$. Природно накласти одну з двох умов:

- максимізація критерію якості: $r(m, q, f) \to \max$;
- досягнення задовільного рівня якості: $r(m, q, f) > \varepsilon$, де ε — заданий поріг.

Подібна постановка задачі зіставлення є дуже загальною. Її конкретизації залежать від того, які обмеження накладаються на множини M і F та на критерій якості.

Можна виокремити два типові класи задач:

- на M та F накладаються жорсткі обмеження; система повинна відповісти на запитання, чи було зіставлення зі зразком успішним чи ні; залежно від цього виконується та чи інша дія. Така постановка задачі характерна для типових експертних систем і систем підтримки прийняття рішень;
- зіставлення обов'язково повинно бути успішним; на M та F при цьому не накладається практично жодних обмежень. Така постановка задачі має велике значення для пошуку аналогій, моделювання інтуїтивного мислення тощо.

Під *аналогією* слід розуміти схожість певних явищ або понять. Під виведенням за аналогією розуміється виведення подібних висновків, якщо є подібні передумови. Виведення за аналогією не завжди є вірним, але воно є потужним засобом планування цілеспрямованих дій та прийняття рішень, особливо у разі, якщо інтелектуальна система зустрічається з незнайомою задачею.

Можна розглядати, наприклад, таку типову постановку задачі логічного виведення за аналогією:

Якщо *A*, то *B*.

C схоже на *A*.

Що випливає з *C*?

Як приклад логічного виведення за аналогією можна навести такий [222]:

Автобус — громадський транспорт.
Трамвай — громадський транспорт.
Автобус перевозить пасажирів.
Отже, і трамвай перевозить пасажирів.

У даному конкретному випадку висновок за аналогією виявився правильним, хоча можна навести багато прикладів хибних висновків за аналогією.

Ще одну можливу постановку задачі проілюструємо на класичному прикладі [254]:

Дані геометричні фігури *A*, *B*, *C*:

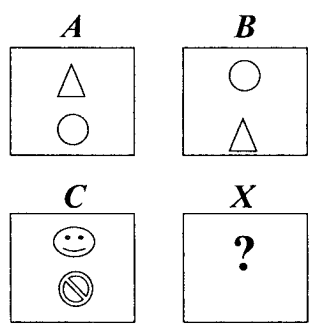


Рис. 22.1. Задача на пошук геометричних аналогій

Потрібно знайти фігуру *X*, яка співвідноситься з фігурою *C* так само, як фігура *B* — з фігурою *A*.

Якщо у нас є формалізовані описи фігур або ми можемо їх отримати і задані правила переходу від однієї фігури до іншої, ми можемо сподіватися на автоматизоване вирішення подібних задач. Однією з перших машинних програм для розв'язання подібних задач була програма Еванса, написана у 1963 р.

В основі більшості методів міркування за аналогією, які застосовуються в системах штучного інтелекту, лежить схема, яка носить назву *квадрата Лейбніца*.

Нехай для переходу від A до B потрібно виконати деяке перетворення f:

$$B = f(A),$$

а C та A пов'язані перетворенням g:

$$C = g(A).$$

Ми можемо міркувати у двох напрямах. По-перше, ми можемо спочатку отримати аналог A, а потім застосувати до нього перетворення f:

$$x_1 = f(C) = f(g(A)).$$

Друга можливість полягає в тому, що ми спочатку переходимо від A до B, а потім отримуємо аналог B:

$$x_2 = g(B) = g(f(C)).$$

Якщо $x_1 = x_2$, тобто перетворення f і g комутативні, утворюється квадрат Лейбніца (рис. 22.2). За розв'язок задачі беремо $x = x_1 = x_2$ (хоча, як зазначалося вище, цей висновок за аналогією далеко не завжди є вірним з погляду строгої логіки). Якщо ж $x_1 \neq x_2$, до розгляду доводиться залучати додаткові міркування.

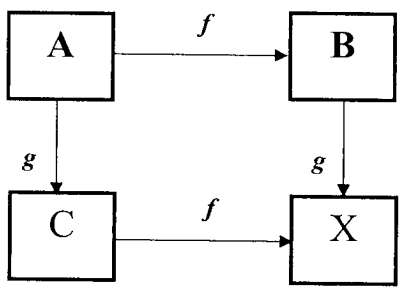


Рис. 22.2. Квадрат Лейбніца

Зрозуміло, що набір метапроцедур може бути не єдиним. Крім метапроцедур, які були розглянуті в цьому розділі, існують й інші.

Дуже важливою метапроцедурою є ***структуризація*** опису ситуації, тобто виокремлення базових елементів і характерних особливостей, на основі яких можна приймати рішення.

Характерною особливістю інтелектуального прийняття рішень є практична неможливість проаналізувати всі елементи ситуації та всі варіанти рішень за прийнятний час. Тому необхідною рисою інтелектуальної системи повинні стати метапроцедури евристичного скорочення пошуку та припинення такого пошуку, якщо його продовження видається недоцільним. Зрозуміло, що рішення при цьому не обов'язково буде оптимальним.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Що таке загальноінтелектуальна метапроцедура?
2. Охарактеризуйте поняття "ситуаційне керування".

3. Що таке семіотична модель?
4. Дайте характеристику стимульно-реактивної теорії.
5. Охарактеризуйте лабіринтну теорію. Наведіть приклади лабіринту можливостей для тих чи інших конкретних задач.
6. Чи є лабіринт можливостей єдиним?
7. Дайте характеристику модельної теорії.
8. Опишіть зіставлення зі зразком як загальноінтелектуальну метапроцедуру.
9. Охарактеризуйте задачу пошуку аналогій.
10. Що таке квадрат Лейбніца? Яким чином на його основі можна здійснювати пошук аналогій?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Чому керування дуже складною системою носить принципово квазіалгоритмічний характер?
2. Які метапроцедури, крім тих, що були описані в цьому розділі, мають значення для успішного функціонування інтелектуальної системи?
3. Наведіть приклади застосування метапроцедур структуризації та припинення пошуку.

Розділ 23
НАВЧАННЯ І САМОНАВЧАННЯ

— Ви вважаєте, що за двадцять років
можна навчити віслюка читати?
— За двадцять років хтось обов'язково помре:
або я, або замовник, або віслюк. У будь-якому разі
я не несу жодної відповідальності.

Анекдот про Ходжу Насредділа

23.1. Базові визначення

Поняття "навчання" та "самонавчання" є одними з ключових у теорії штучного інтелекту. Вони явно або неявно зустрічаються в усіх розділах підручника.

У психології під навчанням розуміється набуття умінь і навичок, які раніше були невідомими. Кажуть, що система навчилась чомусь, якщо вона стала здатною до вирішення тих чи інших задач, які до цього вона вирішувати не могла. Звичайно, якщо нова процедура просто кимось програмується, навряд чи є сенс говорити про справжнє навчання (згадайте поняття декларативного і процедурного підходів, а також дискусію про визначення інтелекту взагалі). Тому часто спеціально підкреслюється, що **при навчанні інтелектуальної системи відбувається самостійне виведення нової інформації з уже існуючої або щойно отриманої** [129].

П. Уїнстон [254] виокремлює такі *рівні навчання*: навчання шляхом програмування; навчання шляхом пояснення; навчання на прикладах; навчання шляхом відкриття. Тобто при переході від нижчих рівнів до вищих "учень" виконує дедалі більшу частину тієї роботи, яку раніше виконував "учитель".

Навчання без "учителя" називається **самонавчанням**.

23.2. Автомати з лінійною тактикою

Історично першими були моделі навчання на основі **стимульно-реактивної теорії** (див. п. 22). По суті, це — моделювання процесу утворення умовних рефлексів на основі "покарань" і "заохочень", які отримує система внаслідок своїх дій. Створенню цих моделей передував ряд фізіологічних експериментів над тваринами. Досить показовим є, наприклад, такий. Піддослідний пацюк має можливість вибрати одну з двох дій: побігти праворуч або ліворуч. Якщо він побіжить наліво, він отримує заохочення — його годують. Якщо ж пацюк побіжить направо, він отримує покарання — удар електричним струмом. Пацюк заздалегідь не знає, куди йому бігти, і він навчається цьому шляхом проб і помилок.

Для моделювання такого типу навчання виявилося зручним застосовувати *автомати з лінійною тактикою* [37]. Розглянемо основні принципи функціонування цих автоматів.

Нехай проводиться серія експериментів, і система в кожному експерименті має можливість вибрати одну з m можливих дій. Тоді автомат з лінійною тактикою, який моделює поведінку такої системи, має m груп станів, кожна з яких відповідає певній дії. Точніше, якщо автомат перебуває в будь-якому стані i-ї групи, він вибере дію, яка відповідає цій групі.

Кожна група має q станів. Параметр q називається *глибиною пам'яті автомата*. Стани кожної групи пронумеровані від 1 до q — від найменш глибокого стану до найглибшого. Позначимо через $k^{(j)}$ j-й стан k-ї групи.

Автомат з лінійною тактикою, який має три групи станів і глибину пам'яті 4, зображено на рис. 23.2. Чорною крапкою відмічено поточний стан (у нашому випадку — 1-й стан 1-ї групи). Це означає, що при черговому експерименті автомат вибере першу дію.

Навчання автомата з лінійною тактикою здійснюється шляхом переходу від одного стану до іншого. Ці переходи відбуваються за такими правилами.

Якщо автомат, перебуваючи в стані $k^{(j)}$ (j-й стан k-ї групи), в результаті вибору k-ї дії отримує заохочення, він переходить до стану $k^{(j+1)}$ (глибшого стану цієї ж групи), якщо $j < q$. Інтуїтивно це можна розуміти як "підкріплення впевненості у правильності свого вибору". Якщо ж $j = q$ (тобто якщо автомат перебуває у найглибшому стані деякої групи), він залишається у цьому ж стані.

Так, автомат, зображений на рис. 23.1, перейде до другого стану цієї ж групи (рис. 23.2).

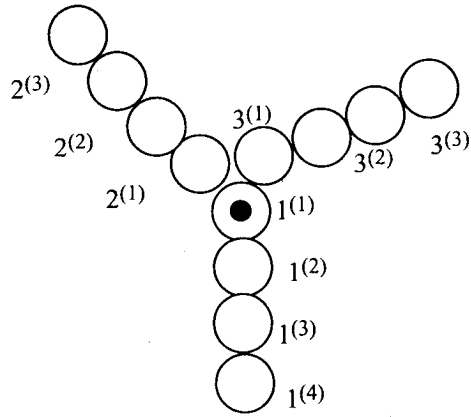


Рис. 23.1. Автомат з лінійною тактикою

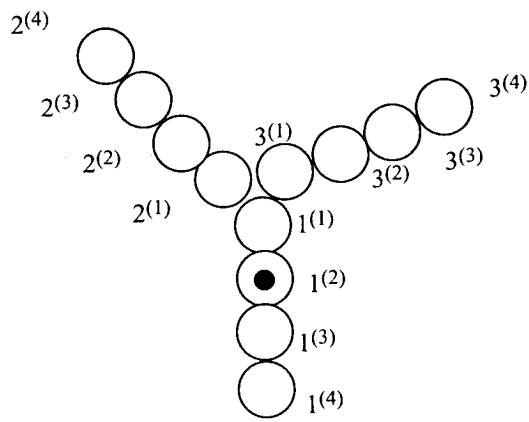


Рис. 23.2. Зміна стану в результаті заохочення

Якщо автомат, перебуваючи в стані $k^{(j)}$, в результаті вибору k-ї дії отримує покарання, він переходить до стану $k^{(j-1)}$ (менш глибокого стану цієї ж групи), якщо $j > 1$. Інтуїтивно це можна розуміти як "послаблення впевненості у правильності дії". Якщо ж $j = 1$ (тобто якщо автомат перебуває у першому стані деякої групи), він переходить до першого стану деякої іншої групи. Це означає зміну поведінки: в наступному експерименті автомат вибере іншу дію.

Так, автомат, зображений на рис. 23.1, перейде до першого стану другої групи (рис. 23.3).

Ми бачимо, що глибина пам'яті має дуже велике значення. Вона характеризує, скільки покарань у середньому повинен отримати автомат для того, щоб змінити свою поведінку.

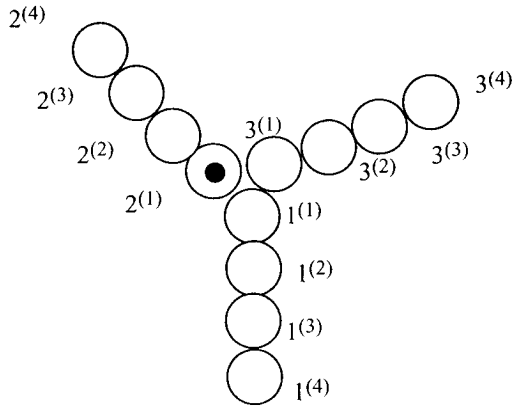


Рис. 23.3. Зміна стану в результаті покарання

У найпростішому випадку, коли кожна дія завжди пов'язана або з покаранням, або із заохоченням, автомат врешті-решт опиняється в найглибшому стані найоптимальнішої дії і залишиться там назавжди. Глибина пам'яті в даному разі повинна бути мінімальною.

Нехай тепер при виборі кожної дії покарання і заохочення відбуваються з певними ймовірностями. У такому випадку поглиблення глибини пам'яті автомата збільшує його "інертність" та зменшує вірогідність випадкових переходів від вигіднішої поведінки до менш вигідної.

Ситуація ще більше ускладнюється, якщо характеристики середовища, тобто ймовірності заохочень і покарань, не залишаються сталими, а змінюються з часом. У цьому разі автомат повинен перенавчатися відповідно до змін середовища, і глибина пам'яті, що характеризує швидкість такого перенавчання, не повинна бути ні завеликою, ні замалою.

Нарешті, важливий напрям досліджень був пов'язаний з колективною поведінкою автоматів, коли розглядається функціонування багатьох автоматів з лінійною тактикою, кожний з яких переслідує свої цілі.

23.3. Формування та засвоєння понять

Одне з найважливіших місць у процесі навчання посідає *узагальнення*, яке полягає у виокремленні спільних рис, характерних для тих чи інших явищ і понять. *Формування понять* є одним з найважливіших видів узагальнення. Проблему формування понять можна неформально сформулювати таким чином: на основі наявних спостережень виробити структурований опис поняття або ж набір правил, які дозволяють визначити, чи охоплюється той чи інший об'єкт певним поняттям. Якщо ж поняття уже сформовані і потрібно лише передати знання про ці поняття інтелектуальній системі, йдеться про *засвоєння понять*.

Однією з перших формалізацій процесу формування і засвоєння понять стала класична *схема Бонгарда* [22]. Правила, які виокремлюють одне

поняття від іншого, шукаються у вигляді диз'юнкції деяких логічних функцій. Пошук інформативних ознак, від яких повинні залежати ці функції, та комбінацій ознак, які мають значення для розпізнавання одного з понять, здійснюється шляхом перебору.

П. Уїнстон у [254] описує *метод засвоєння понять на прикладах зміни модельного опису*. При демонстрації першого прикладу система шляхом його безпосереднього аналізу формує в своїй пам'яті початкову *модель поняття* у вигляді деякої семантичної мережі. Далі описується методика корекції цієї семантичної мережі при демонстрації підкріплюючих прикладів (представників даного поняття) і контрприкладів типу *"майже те"*. *"Майже те"* — це приклад, який не є представником даного класу через певні невідповідності.

Слід згадати програми ЕВРІСКО та АМ, створені Д. Ленатом [129]. Ці програми мали можливість на основі аналізу певного набору базових математичних правил і понять встановлювати нові співвідношення і формувати нові поняття.

Параметричне навчання є, очевидно, найдослідженішим підходом до навчання систем штучного інтелекту. Воно досить поширене, зокрема в системах розпізнавання образів. Нехай, наприклад, система повинна навчитися відрізняти одне поняття від іншого. При параметричному навчанні одразу задається загальний вигляд функції, яка може бути використана для розділу понять. Невідомими залишаються лише деякі параметри, від яких залежить ця функція, і саме ці параметри повинні бути визначені в процесі навчання.

23.4. Базові поняття теорії індуктивних висновків

Слово *"індукція"* (від лат. *inductio* — проведення) означає здійснення висновку від часткового до загального. Індуктивний висновок полягає в тому, що інтелектуальна система спостерігає певний набір фактів і на основі цього формує загальне правило, що описує всі ці факти і дозволяє отримувати нові подібні факти. В іншій інтерпретації спостерігається набір об'єктів, для яких справджується певне правило, і на основі цих спостережень робиться висновок, що це правило справджується для всіх предметів. На відміну від дедуктивного, індуктивний висновок може виявитися хибним.

Слід розрізняти *індуктивну перевірку гіпотез* та *індуктивне формування гіпотез* на основі прикладів. У першому випадку перевіряється деяка уже сформована гіпотеза. Якщо при цьому не вдається здійснити повний перебір, ми ніколи не зможемо бути повністю впевнені у справедливості гіпотези, а можемо лише збільшити або зменшити ступінь нашої впевненості.

У другому ж випадку на початку дослідження гіпотези ще не існує. Вона формується на основі аналізу прикладів, і після цього її часто можна перевірити іншим шляхом.

23.5. Правила Мілля щодо формування гіпотез

Дж. Мілль запропонував чотири правила індуктивних висновків [180]. Суть їх зводиться до такого.

Правило 23.1 (метод виявлення подібностей). *Нехай ми маємо такі спостереження: (a, d, q_1); (a, d, q_2); ..., (a, d, q_n). Тут a, d — деякі факти; $q_1, ..., q_n$ — набори деяких інших фактів. Інакше кажучи, ми маємо n спостережень, в кожному з яких мало місце і a, і d. На підставі таких спостережень можна висунути гіпотезу: "a та d завжди зустрічаються разом". Якщо у нашому розпорядженні є більше інформації, на основі цієї додаткової інформації інколи можна зробити вагоміші висновки: "a є причиною d" або " d є причиною a".*

Правило 23.2 (метод виявлення відмінностей). *Це правило дозволяє зробити ті самі висновки, що й правило 1, але з більшою мірою впевненості. Точніше, відповідний індуктивний висновок "a та d завжди зустрічаються разом" робиться, якщо ми маємо n спостережень, причому в q випадках спостерігалися і a, і d, а в n – q випадках не мали місця ні a, ні d.*

Правило 23.3 (метод виявлення залишку). *Якщо у нас є певний набір спостережень, на основі яких можна дійти висновку, що поява елементів (a, b) тягне за собою появу елементів (c, d), а на основі інших спостережень встановлено, що "b є причиною c", то можна зробити індуктивний висновок, що "a є причиною d".*

Правило 23.4 (метод зв'язних змін). *Нехай $a_i \in A$, $d_i \in D$, і ми маємо групу спостережень: (a_i, ..., d_j); крапками позначені несуттєві елементи. Можна дійти висновку про те, що поява елемента з множини A завжди тягне за собою появу елемента з множини D.*

На основі правил Мілля був запропонований відомий *ДСМ-метод* автоматичного формування гіпотез [221], названий так на честь Джона Стюарта Мілля. Поширений також GUHA-*метод* [59].

Як один з класичних прикладів застосування теорії індуктивних висновків можна навести таку задачу: заданий числовий ряд, наприклад, 1, 4, 9, 16, 25. Знайти наступний член цього ряду. Розвивається формалізована математична теорія індуктивних висновків [176, 299]. Слід згадати роботу [324], що присвячена індуктивному синтезу теорій на основі фактів, або праці, в яких задача формування гіпотез розглядається як деяка оптимізаційна задача [35, 36, 271], метод зростаючих пірамідальних мереж [77, 78] та ін.

23.6. Індуктивна перевірка гіпотез і парадокс Хемпеля

З індуктивною перевіркою гіпотез пов'язаний відомий *парадокс Хемпеля* [70, 72], суть якого проілюструємо на класичному прикладі про чорну ворону.

Нехай учений бажає перевірити гіпотезу *"Всі ворони чорні"*. Природний шлях для індуктивної перевірки цієї гіпотези — дослідити якомога більшу кількість ворін і перевірити, чорні вони чи ні. Кожна виявлена не чорна ворона спростовує гіпотезу, але кожна виявлена чорна ворона збільшує впевненість у справедливості цієї гіпотези, тобто є *підкріплюючим прикладом* для неї.

Але твердження *"Всі ворони чорні"* еквівалентне твердженню *"Всі нечорні предмети не є воронами"*. Спостереження, наприклад, зеленої книги збільшує міру впевненості у справедливості останньої гіпотези, але твердження про те, що воно ж є підкріпленням для гіпотези *"Всі ворони чорні"*, викликає сумнів. Якщо це так, то зелена книга з тим самим успіхом підкріплює і гіпотезу *"Всі ворони білі"*.

До цього можна додати ще й таке міркування. Нехай деякий інопланетянин перевіряє гіпотезу *"Всі люди на Землі нижчі за 5 метрів"*. Ясно, що виявлення людини зі зростом 4 метри 99 сантиметрів, хоч і підтверджує гіпотезу, не підкріплює її, а, навпаки, послаблює.

23.7. Поняття про генетичні алгоритми

Загальновідомо, що жива природа розвивається шляхом природного еволюційного відбору. Останнім часом інтенсивно розвивається напрям досліджень, пов'язаний з набуттям системами штучного інтелекту здатності до подібного еволюційного розвитку. Для моделювання процесу еволюції були розроблені спеціальні методики, які дістали назву *генетичних алгоритмів*. Перші роботи з цього напряму [303] з'явились в середині 70-х років XX століття.

Генетичний алгоритм починає свою роботу з деякого набору даних і процедур і намагається комбінувати їх різним чином (відбувається *схрещування*). Далі з цих комбінацій відбираються найкращі за певним критерієм *(селекція)*, і саме вони беруть участь у наступних схрещуваннях. Інколи до даних і процедур, які залучені до еволюційного відбору, вносяться певні випадкові зміни (відбуваються *мутації*). Генетичні алгоритми добре зарекомендували себе для розв'язання ряду погано формалізованих та важковирішуваних задач.

Детальніше опис машинної еволюції та генетичних алгоритмів можна знайти в [121, 320].

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте класифікацію рівнів навчання за Уїнстоном.
2. Що таке автомат з лінійною тактикою? Опишіть реакцію такого автомата на заохочення і покарання.
3. Що залежить від глибини пам'яті автомата з лінійною тактикою?
4. Охарактеризуйте проблему формування та засвоєння понять.
5. Охарактеризуйте відомі вам схеми засвоєння понять.
6. Що таке параметричне навчання?

7. Охарактеризуйте поняття індуктивного висновку.
8. Чи завжди індуктивний висновок є істинним? Якщо ні, наведіть приклади.
9. У чому полягають індуктивне формування та індуктивна перевірка гіпотез?
10. Опишіть правила Мілля формування гіпотез.
11. У чому полягає парадокс Хемпеля, пов'язаний з індуктивною перевіркою гіпотез?
12. Дайте характеристику генетичних алгоритмів.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Наведіть приклад конкретної задачі, яку можна було б дослідити на основі автоматів з лінійною тактикою.
2. Проаналізуйте парадокс Хемпеля.

Розділ 24

РОЗВ'ЯЗОК ЗАДАЧ ЗА ДОПОМОГОЮ МОДЕЛЮВАННЯ

Тренуйся краще на кішках.
(З к/ф "Операція "И" та інші пригоди Шурика")

24.1. Основні визначення

Розв'язуючи прикладні задачі, необхідно вміти абстрагуватись, виокремлювати суттєві і несуттєві характеристики об'єкта, системи. Потрібно будувати модель системи дослідження.

Модель представляє об'єкт, систему чи поняття (ідею) в деякій формі, що відрізняється від її реального існування. Вона є засобом, що допомагає в поясненні чи вдосконаленні системи. Модель будь-якого об'єкта може бути або його точною копією (хоча виконана з іншого матеріалу і в іншому масштабі), або відображати деякі характерні властивості об'єкта в абстрактній формі [242].

При розробці моделі насамперед потрібно прагнути до розумного балансу її точності (підвищення якої, зазвичай, досягається ускладненням формулювань) і простоти оптимізації. Один зі способів досягнення такого балансу — "покрокове" уточнення моделі, коли одна за одною будуються всі поступово ускладнені версії. Цей прийом особливо корисний при конструюванні моделей, що містять багато складних взаємозв'язаних блоків.

Моделювання застосовується в багатьох сферах. Наприклад, мініатюрні моделі кораблів та літаків часто перевіряються в імітованих водоймах та

аеротрубах до стадії випробування справжніх екземплярів. Створюються моделі будинків для визначення функціональних та естетичних факторів, пов'язаних з будівництвом. За допомогою обчислювальних машин моделюються природні явища. Випадкові числа допомагають наблизити модель до реальності. У багатьох випадках моделювання дозволяє значно спростити планування та виконання експериментів. Тому воно інтенсивно використовується у різних галузях науки і техніки.

Першими моделями можна назвати мовні знаки, що виникали за потреби обмінюватися певними предметами в первісних общинах. Предмети обміну складали в 2 ряди і цим досягалася безпосередня однозначна відповідність (O'—O'') матеріальних об'єктів одного роду O' об'єктам іншого роду O''. Потім зрозуміли, що відповідності між об'єктами O' і O'' можна досягти за допомогою об'єктів третього роду (O) як посередників (O'—O—O''). Цими замінниками спочатку були природні об'єкти (наприклад, пальці рук або ніг). Пізніше з'явились штучні об'єкти (спеціальні палички). Це були перші логічні умовні моделі, які поступово привели до формування поняття числа та грошей [242].

Наступною сходинкою в розвитку моделювання була поява спеціальних знаків і чисел. Вони відображали природний процес розвитку людства, його потребу в абстрагуванні логічних роздумів. Як відомо, для підрахунків спочатку використовувалися зарубки, потім почали з'являтися цифри, математичні символи та ін. Тому перші знаки відображали структуру зарубок.

Подальший етап розвитку логічних знакових моделей полягає у виникненні писемності та математичної символіки. Наприклад, Евклідом було побудовано вчення про подібність, а Гіппократ для вивчення людського ока скористався фізично аналогічною моделлю — оком бика.

Людство продовжувало еволюціонувати, а разом з тим розвивалася виробнича сфера. Стало зрозумілим, що для прогнозування властивостей об'єктів великих розмірів недостатньо використовувати геометричну подібність об'єктів менших розмірів. Перший крок у цьому напрямі був зроблений Ньютоном, що сформулював умови подібності механічних явищ. Були сформульовані теореми, що дозволяли встановити умови подібності явищ будь-якої фізичної природи. Відтоді метод подібності став основним методом екстраполяції характеристик моделі в характеристики оригіналу при фізичному моделюванні.

Поряд з розвитком фізичного моделювання прогресувало логічне моделювання в знаковій формі: логарифми, диференційне числення, числові методи розв'язку різних задач. Для прискорення і полегшення цих дій були розроблені різні обчислювальні засоби моделювання основних математичних операцій: рахівниця, логарифмічна лінійка, планіметр, лічильник Беббіджа, арифмометр, електричні аналогові та обчислювальні пристрої, ЕОМ.

Будь-які два об'єкти навколишнього світу завжди в чомусь схожі і чимось відмінні.

Основним механізмом аналізу цих об'єктів є ***порівняння****. Процес заміни одного об'єкта іншим з метою вивчення або фіксації його важливих властивостей називається* ***моделюванням****.*

Замінений (той, що моделюється) об'єкт називається *оригіналом* або *натурою*, замінюючий (той, що моделює) — *моделлю*.

Модель *— це замінник оригіналу, що дозволяє вивчати або фіксувати деякі властивості оригіналу.*

Фіксація властивостей оригіналу є частковою задачею моделювання. Більш загальним та важливим є дослідження оригіналу за допомогою моделі, що завжди супроводжується фіксацією його важливих властивостей.

Приклад 24.1. При створенні рухомих об'єктів потрібно встановити залежність лобового опору від швидкості. Вирахувати цю залежність аналітично можна, але складно. Це питання вирішується розглядом геометрично подібної моделі менших розмірів в аеротрубі. Заміна оригіналу в даному випадку забезпечує фіксацію важливих властивостей оригіналу і можливість його дослідження.

Приклад 24.2. При проектуванні будівельних об'єктів будується кілька його макетів, кожний з яких відповідно спрямований на виявлення майбутніх естетичних, функціональних та інших властивостей споруди.

Згідно з [126, 242], процес моделювання складається з таких основних етапів:

1. *Постановка задачі і визначення властивостей оригіналу, який потрібно досліджувати.*
2. *Констатація труднощів або неможливості дослідження оригіналу в натурі.*
3. *Вибір моделі, що достатньо добре фіксує істотні властивості оригіналу і може легко досліджуватися.*
4. *Дослідження моделі відповідно до поставленої задачі.*
5. *Перенесення результатів дослідження моделі на оригінал.*
6. *Перевірка цих результатів.*

Найважливішим у процесі моделювання є вибір моделі і перенесення результатів дослідження моделі на оригінал. Для розв'язку цих задач існують як загальні, так і спеціальні методи.

24.2. Класифікація моделей

Класифікація моделей значно спрощує роботу і допомагає систематизувати об'єкти. Будь-яка класифікація має базуватися на найсуттєвіших ознаках об'єктів: *законі функціонування і характерних особливостях вираження властивостей і відношень оригіналу; основах для перетворення властивостей і відношень моделі у властивості і відношення оригіналу.*

Загальна класифікація сучасних моделей [126] показана на рис. 24.1.

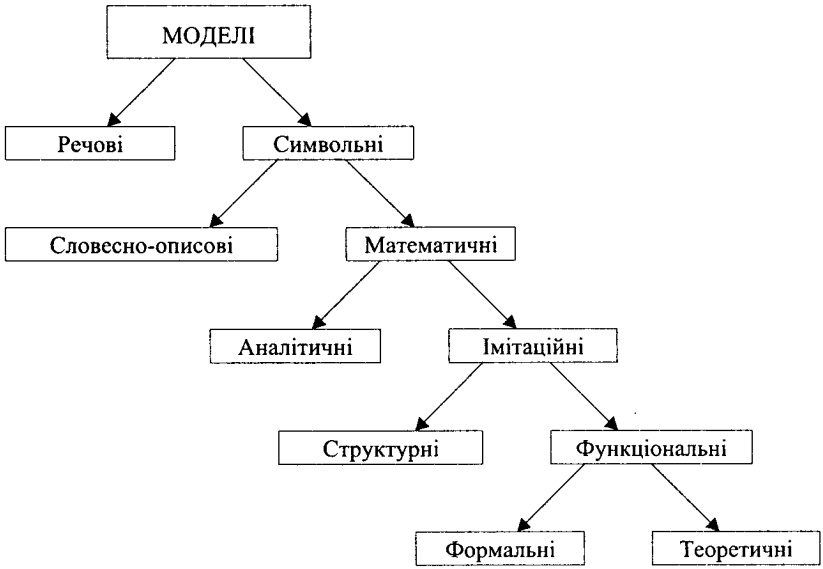


Рис. 24.1. Загальна класифікація сучасних моделей

До *речових* моделей належать моделі конкретних пристроїв (авіамоделі), полігони з відповідними макетами для випробування машин та ін.

У *символьних* моделях фіксація, побудова, опис об'єкта або явища подаються за допомогою якоїсь мови. Серед символьних моделей розрізняють *словесно-описові* та *математичні*. До *словесно-описових* моделей належать моделі, що описуються розмовною мовою. За їх допомогою можна досить повно описати об'єкт або ситуацію, але вони погано формалізуються і тому переважно є проміжним ланцюгом побудови кінцевої моделі для аналізу процесів формальним шляхом за допомогою ЕОМ. Для цього їх перетворюють на математичні.

24.3. Математичне моделювання

Математичними моделями *називаються комплекси математичних залежностей і знаково-логічних виразів, що відображають суттєві характеристики явища, що вивчається.*

Оскільки математичні моделі є найбільш абстрактними, вони досить поширені у системних дослідженнях. Розробляючи математичну модель, потрібно уважно перевірити, чи вона адекватно описує об'єкт.

Математичні моделі можуть бути *аналітичними* або *імітаційними.*

В *аналітичних* моделях процеси функціонування складної системи записуються у вигляді деяких функціональних відношень (алгебраїчних, інтегральних, диференціальних і т. п.) або логічних умов. Аналітичну модель можна досліджувати або аналітично (в загальному вигляді отримують явні залежності для шуканих величин), або чисельно (не маючи рішень

в загальному вигляді, застосовується обчислювальна техніка, щоб отримати числові результати за конкретних числових даних), або якісно (не маючи розв'язку в явному вигляді, можна знайти його деякі властивості, наприклад, оцінити стійкість розв'язку). В аналітичному моделюванні забезпечується подібність характеристик об'єкта і моделі.

Імітаційне моделювання нагадує фізичний експеримент, в якому забезпечується подібність процесів, що проходять як у моделі, так і в реальному об'єкті, а також є можливість отримання наочних результатів моделювання. Перевагою імітаційних моделей є можливість моделювання навіть у тих випадках, коли аналітичні моделі або відсутні, або через складність системи не дають корисних з практичного погляду результатів. Імітаційне моделювання також дозволяє врахувати вплив великого числа випадкових і детермінованих факторів, а також складних залежностей при введенні в модель відповідних елементів та операцій. Іноді застосовуються імітаційні моделі, що відображають тільки *структурно-топологічні* властивості об'єкта (тобто об'єкт задається за допомогою матриці, графу та ін.) або аналізуються тільки *функціональні* залежності.

Якщо знання про об'єкт отримують на основі вивчення фізичних закономірностей функціонування об'єкта дослідження, а структура і параметри моделей мають певне фізичне трактування, тоді таку модель називають *теоретичною. Формальні* моделі отримують на основі порівняння властивостей об'єкта, що моделюється, в середовищі дослідження. У більшості випадків теоретичний підхід дозволяє отримати універсальніші математичні моделі. Формальні моделі описують точніше ту частину предметної області дослідження, де вони визначалися, і менш точні далі від неї [162].

При моделюванні складної системи звичайно використовується сукупність кількох моделей з числа всіх різновидів, що найповніше відображають функціональні зв'язки між підсистемами системи та ієрархію її побудови. При цьому відбувається поступове ускладнення — прості моделі замінюються складнішими. Для того, щоб цей процес був зручним і природним, усі математичні моделі повинні задовольняти трьом вимогам: точність, економічність і універсальність.

Точність моделі — це властивість, що відображає ступінь збігу прогнозованих з її допомогою значень параметрів об'єкта з істинними значеннями. Істинні значення параметрів об'єкта ототожнюють з експериментально отриманими.

Економічність математичної моделі визначається передусім затратами або машинного часу, або пам'яті ЕОМ на програмну реалізацію моделі. Показником економічності математичної моделі може також бути кількість використаних внутрішніх параметрів.

Ступінь універсальності математичної моделі визначається її застосованістю до аналізу групи однотипних об'єктів в одному або багатьох режимах функціонування.

Кількісна та якісна перевага при застосуванні математичного моделювання полягає в такому. Повністю або частково відпадає необхідність у

тривалому і копіткому етапі виготовлення лабороторного макета або напівпромислової установки, в затратах на вимірювальні прилади і обладнання для випробування системи. Значно скорочуються час визначення характеристик і час випробувань. Математична модель робить зрозумілою загальну структуру і розкриває важливі причинно-наслідкові зв'язки та допомагає уникнути неточностей у складних поняттях, можливих при словесному описі. Може бути використана ЕОМ. Вона дає можливість прорахувати всі граничні випадки, провести пошук оптимальних рішень, аналіз і прогноз поведінки об'єктів, що моделюються.

Математичне моделювання *— це заміщення оригіналу математичною моделлю, що забезпечує фіксацію і дослідження властивостей і відношень оригіналу, а також перехід до оригіналу з допомогою математичних методів [177].*

Формальне визначення *математичної моделі*, як правило, будується на теоретико-множинній мові: система $S \le X \times Y$ називається *математичною моделлю*, якщо задано набір задач D_x, $x \in X$ з множиною розв'язків X; для будь-якого елемента $x \in X$ і $y \in Y$ пара (x, y) належить системі S тоді і тільки тоді, коли існує елемент $y \in Y$, який є розв'язком задачі D_x.

Математичне моделювання включає такі етапи:

1) *розробка математичного опису процесу;*

2) *створення алгоритму, що моделює даний процес;*

3) *перевірка адекватності моделі даному процесу;*

4) *використання моделі.*

Існує багато математичних описів, але найпоширенішими є *детерміновані* і *статистичні*. Детерміновані описи (і відповідно моделі) будуються на основі фундаментальних теоретичних законів і закономірностей. Статистичний опис базується на обробці експериментальних даних. Найважче (без урахування економічної ефективності) обрати вектор, що задає цільову функцію або вихідні параметри. При плануванні експерименту збираються дані для визначення коефіцієнта залежності між вхідними та вихідними даними процесу. Існує багато варіантів встановлення такої залежності на основі статистичного аналізу і кожний з них дає можливість побудувати модель, ізоморфну об'єкту дослідження [242].

Тому при виборі структури моделі і критерію адекватності моделі процесу, а також її параметрів, необхідно враховувати мету моделювання і задачу, що ставиться при розробці моделі.

Другою проблемою, що постає перед розробником при використанні моделювання в процесі проектування, є підготовка математичної моделі. Тобто модель зводиться до якоїсь стандартної структурної схеми дискретного процесу, а система рівнянь — до дискретної форми, що дозволяє використовувати ЕОМ.

Математичний опис конкретного об'єкта (його розрахункова модель) може мати різні форми [152]. У найпростішому вигляді це — явна функція, що відображає змінну через її аргументи:

X_i: $Y = f(X_1, ..., X_i, \ldots, X_n)$ або скорочено $Y = f(X_i)$, $i = 1, 2, \ldots n$.

У деяких складніших випадках це — скінченне рівняння:

$F(Y, X_1, ..., X_i, \ldots, X_n) = 0$ або скорочено $F(Y, X_i) = 0$, $i = 1, 2, \ldots n$,

що виражає залежність $Y = f(X_i)$ в неявній формі.

У ще складнішому випадку це звичайне диференційне рівняння:

$$F(Y, Y', Y'', \ldots, Y^{(n)}, X_i, X_i', X_i'', X_i^{(m)}, t) = 0,$$

що зв'язує незалежну змінну t, відомі функції $X_i = X_i(t)$, невідому функцію $Y = Y(t)$ і похідні функцій X_i, Y.

Якщо ввести оператор диференціювання $D = d/dt$, то в символьній формі:

$$F(Y, D_y, D_y^2, \ldots, D_y^n; X_i, D_{x_i}, D_{x_i}^2, \ldots, D_{x_i}^m; t) = 0$$

або скорочено

$$F(Y, X_i, t, D) = 0,$$

математичним описом також може бути диференційне рівняння в часткових похідних:

$$F(Y_i, X_i; t_1, \ldots, t_j, \ldots, t_k; D_1, \ldots, D_j, \ldots, D_k) = 0$$

або скорочено

$$F(Y, X_i, t_j, D_j, A_s) = 0,$$

де $D_j = d/dt_j$ і взяті до уваги постійні коефіцієнти A_s.

Цей процес можна продовжувати, нарощувати і т. д. В цілому під F можна розуміти будь-який оператор, що символізує сукупність певних дій, які виконуються над Y, X_i, t_j, D_j.

Традиційно цей етап моделювання закінчується математичним описом технологічних процесів і структурною схемою всієї системи моделювання.

Перевірку адекватності математичної моделі і процесу дослідження потрібно проводити, оскільки будь-яка модель є лише наближеним відображенням реального процесу внаслідок припущень, що завжди приймаються при створенні математичної моделі. На цьому етапі встановлюється наскільки дані припущення вірні, тобто визначається, чи можна цю модель застосовувати. У разі необхідності математична модель коригується. Для цього використовуються результати вимірювань на самому об'єкті чи на його фізичній моделі, які в невеликих масштабах описують основні фізичні закономірності об'єкта моделювання. Цей етап мусить бути проведений на кожному ієрархічному рівні ускладнення моделі.

Від якості підготовки моделі залежить якість отриманих результатів. Процес формування математичної задачі та алгоритму її розв'язування є досить складним. Його можна уявити у вигляді таких етапів [90].

Вивчення об'єкта. На цьому етапі об'єкт вивчається, досліджуються всі його властивості, всі особливості функціонування. Потрібно чітко визначити фактори, що впливають на його функціонування, їх кількість і ступінь впливу, вибрати критерій оптимізації, що відображає мету задачі розгляду.

Описове моделювання. Встановлюють і словесно фіксують основні зв'язки і залежності між характеристиками процесу або явища з точки зору критерію оптимізації.

Математичне моделювання. Переводять описову модель на формальну описову мову. Всі умови записують у вигляді відповідної системи рівнянь і нерівностей, а критерій оптимізації — у вигляді функції. Після того як задача записана в математичній формі, її конкретний зміст перестає нас цікавити аж до моменту проведення змістовного аналізу отриманого результату. Справа в тому, що часто різні за змістом задачі можуть привести до одного математичного запису.

Вибір або створення методу розв'язку. Головну увагу звертають на отриману математичну структуру задачі (постановку задачі). Виходячи з неї, вибирають або вже відомий метод вирішення, або якусь модифікацію відомого методу, або розробляють новий.

Вибір або написання програми для розв'язку задачі на ЕОМ. Задачі, що мають цільову функцію та умови-обмеження, і ті, що описують поведінку реальних об'єктів, як правило, мають велику кількість змінних і залежностей (рівнянь зв'язку) між ними. Тому вони можуть бути вирішені у розумний термін лише за допомогою ЕОМ. Програма для ЕОМ реалізує вибраний метод розв'язку задачі.

Розв'язок задачі на ЕОМ. Необхідну інформацію для розв'язку задачі вводять в пам'ять ЕОМ разом з програмою. ЕОМ обробляє введену числову інформацію, отримує потрібні результати (розв'язок) і видає його користувачу в заданій формі.

Аналіз отриманого розв'язку. Він буває *формальним* (математичним) і *змістовним.* При формальному аналізі перевіряють відповідність отриманого розв'язку побудованій математичній моделі, тобто перевіряють правильність введення вхідних даних, функціонування програми, ЕОМ і т. п. При змістовному аналізі перевіряють відповідність отриманого розв'язку тому реальному об'єкту, який моделювали. В результаті змістовного аналізу модель (словесна і математична) може бути змінена, після чого процес розгляду повторюється. Тільки після повного закінчення аналізу (і формального, і змістовного) модель може бути використана для розрахунків.

Потрібно мати на увазі, що в математичному програмуванні виокремлюють два типи розв'язку: *допустимий розв'язок* і власне *розв'язок задачі.* Під *допустимим* слід розуміти такий набір значень шуканих величин (змінних), який задовольняє умови-обмеження задачі. *Розв'язком задачі* з множини допустимих розв'язків буде той, за якого цільова функція досягає свого найбільшого (найменшого) розв'язку.

Для того щоб підкреслити важливість змістовного аналізу, можна навести такий приклад. Коли вперше розв'язували задачу про дієту, за фактор оптимізації взяли мінімум затрат, а в умови-обмеження включили тільки вимогу щодо калорійності їжі. Розв'язок задачі був такий: харчуватися потрібно оцтом (він належав до можливих продуктів харчування): і калорійність забезпечена, і вартість мінімальна.

24.4. Задача розміщення

Розглянемо дослідження задачі розміщення за допомогою моделювання за [23, 90].

Постановка задачі. Припустимо, що у нас є система з n населених пунктів і доріг, що з'єднують їх. Розмірами населених пунктів можна знехтувати, зображуючи їх точками; між населеними пунктами задані відстані (по дорозі). Потрібно оптимально розмістити в цій системі школу.

Вивчення властивостей. Спершу треба визначити, що означає оптимальне розміщення. Очевидно, що початкової інформації для розв'язку задачі недостатньо — потрібно ще знати, скільки учнів живе в кожному пункті. Нехай відомі такі дані: p_i $i = 1, 2, \ldots, n$ — число учнів в i-му пункті. Припустимо, що існує всього два пункти: селище, де живе 1000 учнів, та віддалений хутір, де живе 1 учень. Мабуть, зсувати школу в бік хутора було б невірно, потрібно мінімізувати суму учне-кілометрів.

Повернемося до постановки з n пунктами. Уявімо собі, що проектувальник майбутньої школи рухається з пункту 1 в пункт n і до нього прив'язані резинки: a — до спини і b — до грудей. Тут a — кількість учнів, що залишилися за спиною, b — які живуть попереду, $a + b = s$. Після того як проектувальник залишить перший пункт, p_1 резинок тягнутиме його назад, а $s - p_i$ — вперед; якщо $p_1 < s - p_i$, то рівнодійно тягнутиме вперед. Тобто, після кожного кроку розміщувача p_1 учням доведеться ранком робити на один крок більше і $s - p_i$ учням — на один крок менше.

Можна сформулювати проміжне твердження.

Твердження 24.1. Школу потрібно розміщувати в населеному пункті.

Доведення. Припустимо, що точка зупинки розміщувача сталася на дорозі між двома сусідніми пунктами з кількістю учнів c і d відповідно. Позаду точки зупинки живе a учнів, $a = \alpha + c$, а попереду — b учнів, $b = d + \beta$; $a = b$. Нехай розміщувач зробить крок назад. Внаслідок цього a учням доведеться робити на крок менше, а b — на крок більше, але, враховуючи, що $a = b$, можемо дійти висновку — кількість учне-кілометрів не зміниться. Отже, розміщувач може зробити ще крок назад і т. д., поки він не ввійде в попереднє місто. Тут розміщувача назад тягнуть α резинок, c резинок не натягнуті, а b тягнуть вперед. Оскільки $b > \alpha$, розміщувач піде вперед, але тоді $\alpha + c = a$ резинок почнуть тягнути назад, і оскільки $a = b$, розміщувач зупиниться. Тому згідно з обраними правилами руху розміщувач не зможе вийти з пункту на дорогу.

Твердження доведено.

Для формальнішого розв'язку цієї задачі використаємо теорію графів: населені пункти вважатимуться вершинами графу, ребра — дорогами, що з'єднують міста.

З попередніх роздумів зрозуміло, що школу потрібно розташовувати у вершині графу. А це означає, що ми повинні вибрати місце для школи не з нескінченної кількості точок, а з n точок, що робить повний перебір легкоздійсненним.

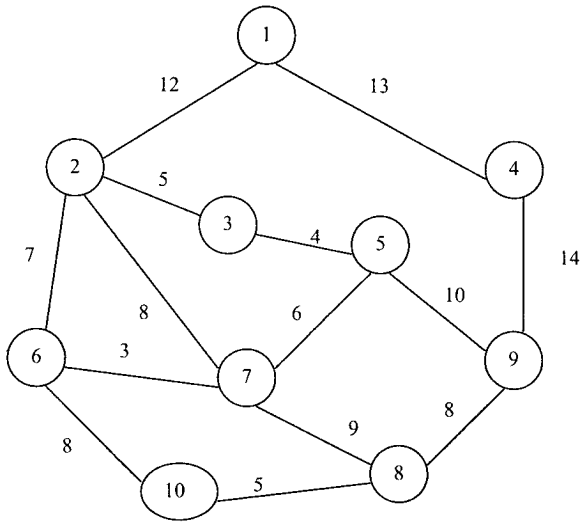


Рис. 24.2. Граф-модель задачі

Граф системи продемонстровано на рис. 24.2. У середині кола, що зображає вершину, стоїть номер населеного пункту. Кількість учнів у вершинах $i = 1, \ldots, 10$ задано відповідно числами $p = (80, 40, 65, 100, 74, 90, 56, 34, 120, 23)$. Ребра позначені числами, що характеризують відстані між відповідними населеними пунктами.

Розв'язок. Насамперед знайдемо найкоротші ланцюжки з кожної вершини в кожну іншу вершину за допомогою якогось із відомих алгоритмів. Отримаємо результат, записаний в матриці. Нехай d_{ij} позначає мінімальний шлях між вершинами i та j.

$$\begin{pmatrix}
0 & 12 & 17 & 13 & 21 & 19 & 20 & 29 & 27 & 27 \\
12 & 0 & 5 & 25 & 9 & 7 & 8 & 17 & 19 & 15 \\
17 & 5 & 0 & 28 & 4 & 12 & 10 & 19 & 14 & 20 \\
13 & 25 & 28 & 0 & 24 & 32 & 30 & 22 & 14 & 27 \\
21 & 9 & 4 & 24 & 0 & 9 & 6 & 15 & 10 & 17 \\
19 & 7 & 12 & 32 & 9 & 0 & 3 & 12 & 19 & 8 \\
20 & 8 & 10 & 30 & 6 & 3 & 0 & 9 & 16 & 11 \\
29 & 17 & 19 & 22 & 15 & 12 & 9 & 0 & 8 & 5 \\
27 & 19 & 14 & 14 & 10 & 19 & 16 & 8 & 0 & 13 \\
27 & 15 & 20 & 27 & 17 & 8 & 11 & 5 & 13 & 0
\end{pmatrix}$$

Якщо поставити школу у вершині i, то загальна кількість учне-кілометрів дорівнюватиме сумі добутків i-го рядка матриці на відповідні числа масиву p (скалярному добутку першого рядка матриці на вектор p). Пере-

бравши всі вершини, знайдемо мінімум. У цій вершині мінімуму і потрібно розмістити школу.

Тепер, коли, по суті, все зрозуміло, задачу можна поставити формально:

Задано граф G з n вершинами, яким відповідає вага $p_1, p_2, \ldots, p_n$. Знайти точку u на мережі (в вершині або на ребрі), таку щоб:

$$f = \sum_{i=1}^{n} d_{iu}\, p_i \Rightarrow \min.$$

У результаті доведеного твердження мінімізуюча функціонал f точка розміщується в одній з вершин. Вершина, що мінімізує поданий вище вираз, знаходиться перебором [23].

Подивимось на задачу розміщення з іншого погляду [119], погляду програміста.

Постановка задачі. Припустимо, що одна сторона міської площі прямокутної форми має довжину, що дорівнює розмірам 20 однакових автомобілей, розташованих один за одним. Спеціальних позначок для можливих місць зупинки автомобілів немає. Автомобілі можуть стояти вздовж тротуару випадково. Скільки автомобілів можна розмістити вздовж тротуару?

Після вивчення постановки задачі видно, що розв'язок може набувати значення від 10 до 20. Найбільше значення буде, якщо автомобілі встановлюватимуть безпосередньо один за одним. Зрозуміло, що для деякого $X > 0$ найбільша відстань між будь-якими двома автомобілями дорівнює $1 - X$. Отже, кожен автомобіль займає ділянку, одна з сторін якої має відстань $2 - X$ (рис. 24.3).

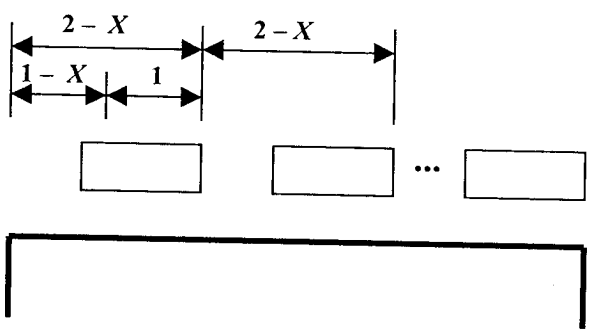


Рис. 24.3. Можливе розміщення автомобілів

Дану задачу можна вирішити аналітично, але зараз спробуємо скласти її програмну модель, тобто розробимо модель процесу розташування автомобілів та підрахуємо кількість розташованих автомобілів.

Один з можливих розв'язків задачі полягає в проведенні експерименту зі справжніми автомобілями і в реальних міських умовах. Знайдемо міську площу, що задовольняла б специфікаціям задачі, і підрахуємо кількість розташованих на ній автомобілів. Оскільки ця кількість змінна, необхідно проводити такий експеримент впродовж кількох днів і визначити середню кількість автомобілів за деякий час.

Але такий розв'язок має два принципових недоліки:

1) щодня вдається отримати лише одне значення розміщених автомобілів. Отже, для отримання розв'язку з потрібною точністю необхідно буде аналізувати багато днів;

2) автомобілі різного типу мають різну довжину. Тому в реальному світі не існує умов, які б повністю відповідали специфікаціям задачі.

Є також інша можливість — побудувати модель міської площі і автомобілів фіксованої довжини, а потім провести експеримент. Але ми спробуємо побудувати програмну модель і розв'язати задачу, використовуючи моделювання. В цій задачі нас цікавлять тільки ті ознаки, які стосуються розміщення автомобілів вздовж тротуару. Інші властивості (колір, ширина або висота автомобіля, природні умови тощо) до уваги братись не будуть [119].

Не має значення і абсолютна довжина автомобіля. Істотна тільки його відносна довжина (стосовно довжини тротуару). Припустимо, що довжина кожного автомобіля дорівнює 1, а довжина тротуару — 20. Якщо відомо розміщення передньої частини автомобіля, то відомо все місце, що займає автомобіль вздовж тротуару. Отже, якщо автомобіль розміщений, починаючи з точки A, то наступний автомобіль можна поставити, починаючи з точки $A + 1$.

Для того щоб нашу модель максимально наблизити до реальних умов, генеруватимемо число, що з рівною ймовірністю розподіляється між 0 і 20. Потім перевірятимемо, чи можна встановити автомобіль на місце, що відповідає числу. Якщо це можливо, то автомобіль розміститься у вказаному місці. Цей процес продовжуватимемо доти, поки існуватиме місце, придатне для розміщення хоча б одного автомобіля. Для генерування послідовності псевдовипадкових чисел використаємо функцію X = LRANDOM (X), яка генерує випадкові числа від 0 до 1.

При моделюванні даної ситуації необхідно вирішувати дві різні задачі. Потрібно слідкувати за розміщенням автомобілів і визначати, чи можна встановити новий автомобіль.

Необхідно генерувати випадкові числа, намагатися вставити машини на визначені цими числами місця і у відповідний час припинити виконання програми.

Тому програму можна розділити на дві частини: головну програму, що керує процесом моделювання, і допоміжну частину, що формалізує функції керування розміщенням автомобілів.

З постановки задачі випливає, що в головній програмі варто використати чотири функції, пов'язані з процесом розміщення автомобілів. Булева функція CANPARK (X) визначає можливість розміщення автомобіля за точкою X. Функція PARK (X) розміщує автомобіль за точкою X, а функція FINDMAX визначає найбільшу довжину частини тротуару, де є ще вільне місце. Оскільки для отримання середнього значення програму необхідно виконати кілька разів, потрібна функція INITIALIZE, яка видаляє зі стоянки всі розташовані автомобілі.

Враховуючи використання поданих вище функцій, структура програми може бути записана таким чином [119]:

```
begin
    Встановлення початкового стану;
    while (є місце для стоянки автомобілів) do
    begin
        Знайти наступний автомобіль для розміщення;
        Поставити автомобіль;
    end
    Вивести кількість встановлених автомобілів;
end;
```

Для реалізації головної програми необхідно мати також засоби реєстрації місцерозташування автомобілів у нашій моделі. Розміщення деякого автомобіля можна охарактеризувати положенням його передньої частини щодо тротуару. Тоді для реєстрації положення автомобіля можна скористатися елементом масиву CARS [BLOCKLENGTH + 2].

Для реалізації функції INITIALIZE видалити всі автомобілі можна шляхом присвоєння змінній NUMBERPACKED нульового значення.

У функції CANPARK перевіряється, чи є достатньо місця для розміщення автомобіля в заданій поточній точці розміщення. Для цього можна використати змінну LOCATION як точку бажаного розміщення, і оскільки координати точок розміщення автомобілів впорядковані в масиві CARS, необхідно виконати такі дії:

1. Знайти в масиві CARS точку розміщення автомобіля, яка передує точці розміщення нового автомобіля, і точку розміщення автомобіля, наступну після точки розміщення нового автомобіля.

2. Перевірити, чи можна встановити новий автомобіль між двома знайденими автомобілями.

Бажано також встановити фіктивні автомобілі в кожній кінцевій точці тротуару START і BLOCKLENGTH, що усуває необхідність розгляду спеціальних випадків: коли автомобіль, що встановлюється, становлять перед першим автомобілем; коли автомобіль, що встановлюється, потрібно поставити після останнього автомобіля; розміщення першого автомобіля. Тому семантичну частину цієї функції можна описати так:

```
begin
    Знайти попередній і наступний автомобілі;
    FOLLOW = наступний автомобіль;
    PRECEDE = попередній автомобіль;
    if (X встановлюється між PRECEDE і FOLLOW)
    then canpark = true
    else canpark = false
end;
```

Знаходження попереднього і наступного автомобілів забезпечується за допомогою простого циклу, в якому перебуває єдина перша точка, що лежить праворуч від точки LOCATION. Відповідно до попередніх роздумів і

з урахуванням внесених змін в функцію INITIALIZE можна стверджувати, що така точка буде єдиною. Тоді фрагмент функції, яка реалізує пошук автомобілів, матиме такий вигляд:

```
begin
    i = 1;
    while (LOCATION > CARS [i]) do
    i = i + 1;
    FOLLOW = CARS [i];
    PRECEDE = CARS [i – 1];
end;
```

Автомобіль може бути встановлений у точці розміщення LOCATION, якщо вона знаходиться на відстані щонайменше одиниці від точки PRECEDE (враховується довжина нового автомобіля), а точка FOLLOW — на відстані щонайменше одиниці від точки LOCATION (враховується довжина автомобіля, що знаходиться в точці FOLLOW). Тоді оператор if у функції CANPARK матиме такий вигляд:

```
if ((LOCATION – PRECEDE >= 1) and (FOLLOW – LOCATION >= 1))
then canpark = true
else canpark = false;
```

Функція PARK встановлює автомобіль на задане місце. Якщо припустити, що змінна *j* є індексом точки розміщення FOLLOW (як і у функції CANPARK, тоді розміщення автомобіля в точці LOCATION виконується з допомогою таких операторів:

```
j = NUMBERPARKED + 1;
while j >= i do
begin
    CARS [j + 1] = CARS [j];
    j := j – 1
end;
```

За допомогою функції FINDMAX знаходять найбільший проміжок, що є між двома послідовно розміщеними автомобілями. Тіло функції виглядає так:

```
MAX = 0;
i = START + 1;
while i < NUMBERPARKED + 2 do
begin
    temp = CARS [i] – CARS [i – 1] – 1;
    if (temp > MAX) then MAX = temp;
    i = i + 1;
end;
```

Для отримання середнього значення програма повинна виконуватися кілька разів. Отже, ми дістали потрібний розв'язок. На жаль, він, хоча й коректний, але неефективний за рахунок основного циклу головної функції. Пошук потрібного числа може призвести до дуже великої кількості вико-

нань циклу. Звичайно, цикл врешті-решт закінчиться, але очікуваний час виконання може виявитися тривалим. Подібний недолік можна усунути. Подумайте, яким чином?

24.5. Мережі Петрі та їх використання

Для побудови складних систем обробки інформації та моделювання асинхронних інформаційних потоків потрібна математична модель, зручна в описі керування роботою таких систем. Історично першою для такого моделювання використовувалася теорія автоматів [11]. Автомати застосовують для моделювання послідовних алгоритмічних систем — коли система (автомат) послідовно переходить зі стану в стан відповідно до заданої функції переходу і здійснює наступний крок алгоритму.

Але існують і неалгоритмічні паралельні системи з недетермінованою поведінкою, в якій компоненти функціонують незалежно і взаємодіють час від часу. Прикладами таких систем є *багатопроцесорні обчислювальні машини, паралельні програми, що моделюють паралельні дискретні системи, мультипрограмні операційні системи.* Такі системи не описуються адекватно в термінах класичної теорії автоматів. Наприклад, неможливо описати ці системи за допомогою таких термінів, як стан автомату, глобальна функція переходу. Один з методів опису подібних систем запропонував К. Петрі, де для моделювання асинхронних інформаційних потоків у системах обробки даних використовується мережна модель — мережа Петрі. Їй притаманні такі риси:

1. Мережа Петрі використовується для опису змодельованої системи, і це може бути застосовано для специфікацій (для побудови систем) або опису системи.
2. Поведінку мереж Петрі можна проаналізувати як моделюванням (що еквівалентно виконанню програми та її налагодженню), так і формальнішими методами аналізу (що відповідає програмній перевірці).
3. Процес створення опису та виконання аналізу допомагає краще зрозуміти модельовану систему самому моделювальнику.

Розглянемо формальне визначення мережі Петрі [213].

Мережею називається трійка $S = (P, T, F)$, де P — не пуста множина елементів мережі, які називають місцями; T — не пуста множина переходів; $F \subseteq P \times T \cup T \times P$ — відношення інцидентності. До того ж для S повинні виконуватись такі умови:

1) $P \cap T = \varnothing$;

2) довільне місце в мережі зв'язано з деяким переходом і довільний перехід зв'язаний з деяким місцем у мережі;

3) якщо для довільного елемента мережі x позначити через $^{\wedge}x$ множину його вхідних елементів $\{y \mid yFx\}$, а через $x^{\wedge}$ — множину його вихідних елементів $\{y \mid xFy\}$, тоді $\forall p_1, p_2 \in P$: $(^{\wedge}p_1 = {}^{\wedge}p_2)\ \&\ (p_1{}^{\wedge} = p_2{}^{\wedge}) \Rightarrow (p_1 = p_2)$.

Мережа Петрі *— це набір* $S' = (P, T, F, W, M_0)$, *де* (P, T, F) *— скінченна мережа, а* $W: F \to N\backslash\{0\}$, *і* $M_0 : P \to N$ *відповідно функції кратності дуг і початкової розмітки* (N *— множина натуральних чисел з нулем*).

Функціонування мережі Петрі описують формально за допомогою множини послідовних спрацювань переходів і множини розміток, що можуть бути досягнуті в мережі. Множину розміток $M: P \to N$, враховуючи впорядкованість місць мережі S', можна задавати вектором чисел $m = (m_1, m_2, \ldots, m_n)$, де для $\forall i$, $1 \le i \le n$, $m_i = M(p_i)$.

Функція інцидентності F: $P \times T \cup T \times P \to N$ визначається так:

$$F(x, y) = \begin{cases} n, & \text{якщо } xFy \ \& \ (w(x, y)) = n, \\ 0, & \text{якщо } \neg(xFy). \end{cases}$$

Перехід t може спрацювати при деякій розмітці M мережі S', якщо $\forall p \in t^\wedge$: $M(p) \ge F(p, t)$. Спрацювання переходу t при розмітці M породжує розмітку M' згідно з правилами:

$\forall p \in P$: $M'(p) = M(p) - F(p, t) + F(t, p)$.

Приклад 24.3. Припустимо, що в нас є мережа Петрі, зображена на рис. 24.4.

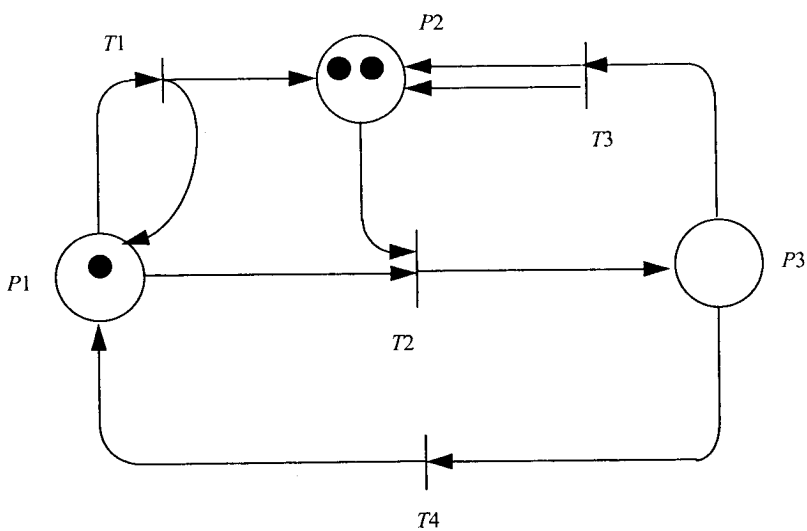


Рис. 24.4. Графічне зображення мережі Петрі

Початкова розмітка її $M_0 = (1, 2, 0)$. При такій розмітці за правилами функціонування мережі може спрацювати або перехід $T1$ (до переходу входить одна дуга з місця $P1$, яка вже має одну фішку), або перехід $T2$ (до переходу входять дві дуги з місць $P1$ і $P2$, які мають відповідно 1 і 2 фішки). Система недетерміновано вибирає спрацювання одного з переходів. Розглянемо можливі варіанти.

Якщо спрацьовує перехід $T1$, тоді наступна розмітка буде $M_1 = (1, 3, 0)$. Вона отримується так. З усіх місць, які мають дуги, що входять до переходу $T1$, забираються фішки, кількість яких визначається кількістю дуг, що виходять з місця і входять в перехід $T1$. Після цього розмітка $P1$ стане дорівнювати 0. Але після спрацьовування переходу $T1$ в усі місця, в які ведуть дуги з $T1$, додається кількість фішок з дуги; із $T1$ — кількість фішок, яка дорівнює кількості відповідних дуг. Тому після спрацьовування $T2$: $P1 = 0 + 1 = 1$, а $P2 = 2 + 1 = 3$. Місце $P3$ залишається без змін, бо в нього не входять дуги з переходу, який спрацював.

Після спрацьовування переходу $T1$ можливе подальше спрацьовування знов $T1$ або $T2$. Якщо недетерміновано вибереться $T2$, тоді $M_1 = (0, 1, 1)$. Далі може спрацювати або перехід $T3$, або ж перехід $T4$.

Процес роботи мережі Петрі може закінчитись або при досягненні ситуації, в якій не може спрацювати жодний перехід, або при досягненні якоїсь певної, попередньо визначеної розмітки.

Розглянемо перший випадок. Якщо спрацює перехід $T3$, тоді розмітка матиме вигляд $M_3 = (0 + 0, 1 + 2, 1 - 1) = (0, 3, 0)$, і жодний з переходів не може спрацювати. Для характеристики другого випадку закінчення роботи мережі Петрі візьмемо за фінальну розмітку $M_ф = (1, 1, 0)$.

Якщо при розмітці $M_2 = (0, 1, 1)$ спрацює перехід $T4$, тоді ми отримаємо $M_ф = (1, 1, 0)$. У даному разі $M_ф = M_3$ і мережа закінчить роботу.

Розглянемо *будову моделі системи.* Компоненти моделі уявляють у вигляді абстрактних *подій.* Подія — виконання оператором програми, переривання в операційній системі. Подія може реалізуватись один раз, кілька разів, породжуючи *дії*, або не реалізуватись взагалі. Сукупність дій, що виникають як реалізації подій при функціонуванні системи, утворює *процес*, породжуваний системою. Тобто, система в однакових умовах функціонує по-різному, породжуючи множину процесів, має недетерміновану поведінку.

Для *асинхронних моделей* Петрі замінив послідовні зміни станів, як це відбувається у послідовних алгоритмічних системах, на *причинно-наслідкові зв'язки.* Події в асинхронних моделях можуть бути як *елементарними* (неподільними), так і *складними структурованими* (складатися з "підподій"). Описати взаємодії в системах можливо, якщо замість самих зв'язків між ними описувати ситуації, коли ця подія може реалізуватись. Глобальні ситуації в системі формуються локальними операціями — *умовами* реалізації подій.

Кожна умова має *ємність*: умова не виконана — ємність дорівнює 0; умова виконана — ємність дорівнює 1; умова виконана з n-кратним запасом — ємність дорівнює n, $n > 0$, $n \in N$. Приклади умов: наявність даного для операції в програмі, стан регістра в ЕОМ. Сукупність умов спричиняє подію *(передумови події)* або реалізація події змінює деякі умови *(постумови).*

У мережах Петрі події та умови — це абстрактні символи з 2-х непересетних алфавітів: множини *переходів* і множини *місць*. Умови-місця та події-переходи пов'язані відношенням залежності (причинно-наслідкового зв'язку) — *дугами*. Кожна дуга має кратність — невід'ємне ціле число, яке відповідає необхідній для спрацювання переходу кількості *маркерів (фішок)*

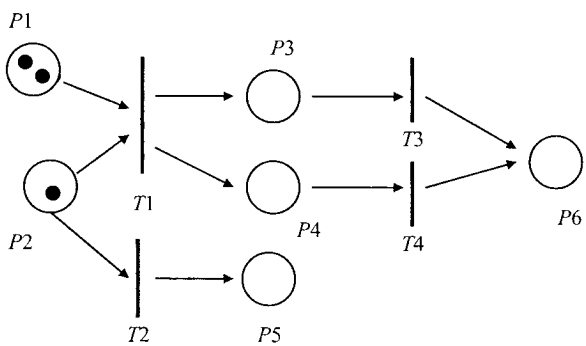
у місці (або ємності умови), з яким ця дуга з'єднує перехід. Маркери — це кількість ресурсів, необхідна для спрацювання події-переходу.

Місця, з яких виходять дуги в даний перехід, називаються *вхідними місцями*; місця, в які входять дуги з переходу — *вихідними місцями*. Перехід спрацьовує, якщо ємність кожного його вхідного місця не менша від кратності відповідної дуги. В результаті переходу ємність кожної передумови або кількість маркерів вхідних місць цього переходу зменшується на кратність відповідної дуги, а ємність кожної постумови збільшується на кратність дуг. В автоматній мережі Петрі для кожного переходу існує лише одне вхідне і лише одне вихідне місця.

Роботу побудованої нами мережі уявимо як сукупність *локальних дій — спрацювання переходів — відповідно реалізація подій*, що приводить до зміни розмітки місць (до локальної зміни умов у системі). Перехід спрацьовує, якщо виконуються всі умови реалізації відповідної події (для одинарної системи Петрі — якщо всі вхідні місця переходу мають хоча б один маркер).

Якщо кілька переходів може спрацювати і вони мають однакове вхідне місце, то спрацьовує один перехід, при цьому це може призвести до того, що інші переходи вже не зможуть спрацювати — *модель конфлікту між подіями*. Мережа зупиняється, якщо жоден перехід не може спрацювати.

Приклад 24.4. Розглянемо ще один приклад мережі Петрі (рис. 24.5), акцентуючи нашу увагу на зазначених особливостях.



а

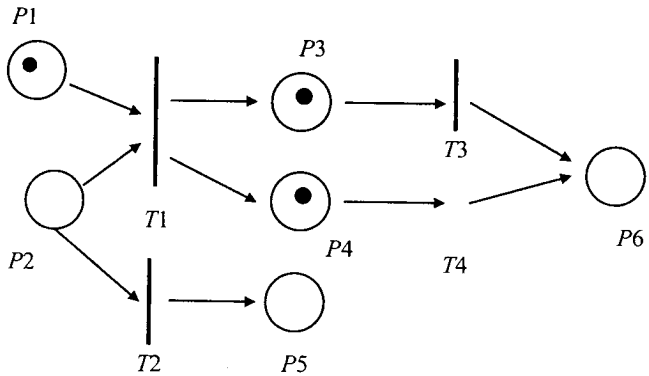
Рис. 24.5. Функціонування мережі Петрі:

а — початкова розмітка мережі Петрі. Для переходу $T1$ вхідні місця — $P1$ та $P2$; вихідні місця — $P3$ і $P4$. Місце $P2$ є також вхідним місцем для переходу $T2$;

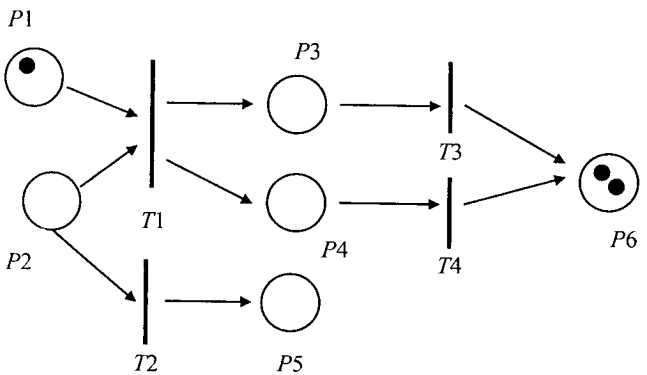
б — робота мережі Петрі. Спрацювання переходу $T1$ спричинило те, що перехід $T2$ вже не зможе спрацювати — він заблокований;

в — робота мережі Петрі. Спрацювали переходи $T3$ і $T4$. В результаті жоден з переходів вже не зможе спрацювати, вони всі в тупику. У такому разі кажуть, що вся система тупикова. Мережа зупиняється;

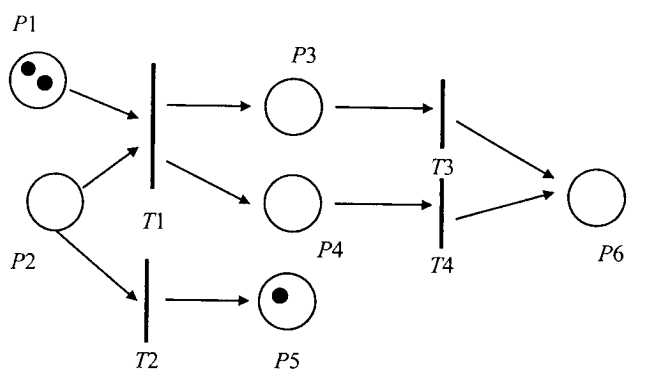
г — робота мережі Петрі. Спрацювання переходу $T2$ спричинило те, що перехід $T1$ вже не зможе спрацювати (він заблокований) — модель конфлікту між подіями. Всі інші переходи також стають заблокованими, мережа зупиняється.


P1
P3
T3
T1
P2
P4
T4
P6
T2
P5

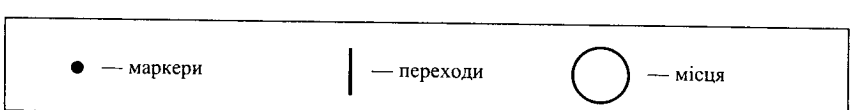
б


P1
P3
T3
T1
P2
P4
T4
P6
T2
P5

в


P1
P3
T3
T1
P2
P4
T4
P6
T2
P5

г


— маркери
— переходи
— місця

Приклад системи, яку можна змоделювати за допомогою мережі Петрі — операційна система з паралельними циклами процесів, що взаємодіють між собою. Мережа Петрі описує функціональну схему системи, а робота мережі моделює *процес*, що виникає при функціонуванні системи.

24.6. Властивості мереж Петрі

Вище ми розглядали мережі Петрі, що складаються з умов та подій, при цьому всі маркери (умови) однакові. Це так звані неінтерпретовані орієнтовані мережі Петрі. Існує ще й інший вид мереж Петрі — *розфарбовані мережі Петрі*. На практиці набагато частіше використовують саме цей вид мереж. Відмінність їх від звичайних мереж полягає у тому, що маркери-умови в них не однакові, а "розфарбовані". Що це означає? Просто такі маркери мають різні властивості, і дуги "пропускають" лише якісь певні кольори маркерів. Тобто, для спрацювання переходу потрібна не просто наявність у вхідних місцях кількості маркерів, рівної кратності відповідних з'єднуючих дуг, але ці маркери мають бути ще й певних кольорів. Ця умова ускладнює саме визначення мережі та аналіз її роботи.

На практиці майже завжди маємо ситуацію обмеженої ємності реальних умов реалізації. Для реалізації події, що моделюється переходом, потрібно, щоб кожне вхідне місце-умова мало кількість маркерів, яка дорівнює кратності дуги, що з'єднує його з переходом. Але при роботі мережі деякі її місця можуть накопичувати необмежену кількість маркерів. Якщо ці місця інтерпретувати як буфери даних, сигналів у системах, то можливе переповнення. Тому потрібно дати визначення обмеженого місця. Місце P в мережі $N = (P, T, F, W, M_0)$ — *обмежене*, якщо існує таке n, що для "досяжної в мережі розмітки M'' виконується $M(P) \le n$.

Серед мереж Петрі виокремлюють також консервативні мережі. *Консервативна мережа* — мережа Петрі, сума маркерів в усіх місцях якої залишається незмінною у процесі роботи мережі.

Для деякого переходу умова його спрацювання може ніколи не виконатись. Тоді такий перехід зайвий, його можна видалити. Також може статися, що після певної послідовності спрацювання переходів деякі переходи стають такими, що вже ніколи не зможуть спрацювати. Таким чином моделюються ситуації, коли в системах трапляються випадки неспроможності спрацювання деяких процесів. В операційній системі це називається *взаємним блокуванням процесів* (deadlocks).

Розрізняють [213] такі особливості мереж Петрі:

1) мають графічне подання. З графічною формою інтуїтивно легше зрозуміти роботу мережі навіть тим, хто раніше не мав й уяви про такі моделі;

2) мають чітко виражену семантику, що однозначно визначає поведінку мережі. Семантика дозволяє здійснювати імітацію для мереж Петрі, і це надає підстави для формальних методів аналізу;

3) дуже загальні, тому за допомогою них можна описати досить велику кількість різноманітних систем. Застосування мереж Петрі — від не-

формальних (наприклад, опис процесів) до формальних систем (наприклад, протоколи зв'язку);

4) мають досить мало базисних елементів, але їх виразні можливості дуже потужні. Визначення мереж Петрі досить коротке і побудоване на стандартних поняттях, які вже відомі з математики та мов програмування. До того ж, невелика кількість елементів означає, що можна легко розробляти потужні методи аналізу;

5) мають явний опис як станів, так і дії. Більшість же системних мов опису описують або стани, або дії;

6) мають семантику, що будується на чистому паралелізмі;

7) пропонують ієрархічний опис. Це означає, що ми можемо створити велику мережу Петрі за допомогою з'єднання менших мереж. Створення ієрархії мереж Петрі відіграє роль, подібну до підпрограм, процедур, функцій і модулів мов програмування;

8) об'єднують опис управління та синхронізацію з описом обробки даних. Багато інших графічних мов опису працюють з графами, які лише описують конфігурацію системи, коли визначено детально поведінку;

9) можуть бути розширеними часовими характеристиками. Це означає, що можна використовувати мову моделювання для специфікації, підтвердження вірності функціональних і логічних властивостей (як, наприклад, відсутність тупиків) і характеристик (приклад — середній час очікування). Основна ідея розширення поняттям часу — впровадження глобального часу та дозвіл кожній фішці-умові мати позначку часу, коли вона може бути використана переходом;

10) стабільні до незначних змін у змодельованій системі. Це доведено багатьма практичними експериментами і означає, що невеликі модифікації модельованої системи не змінюють повністю структуру мережі Петрі. Наприклад, можна додати новий послідовний процес, не змінюючи структуру мережі, що являє собою існуючий процес;

11) пропонують діалогове моделювання, де результати представлені на самій мережній діаграмі. Моделювання дозволяє відлагоджувати велику модель, поки вона створюється — аналогічно до хорошого програміста, який налагоджує окремі частини програми після їх написання. Значення даних у маркерах (фішках) можна перевірити під час їх руху по мережі;

12) мають багато методів аналізу, за допомогою яких властивості мереж Петрі можуть бути формально доведеними. Два найважливіших методи аналізу — граф подій і мова розміток.

24.7. Застосування мереж Петрі в машинній побудові зв'язних тестів

Задача обробки природної мови належить до основних проблем створення систем штучного інтелекту. При вирішенні цієї задачі за допомогою ЕОМ основним питанням є питання задання знань, тобто як сприймати, зберігати і обробляти великі обсяги даних з розвинутою внутрішньою

структурою, що потребує значних обчислень. У розподіленій базі знань кожний процесор комплексу можна зв'язати з окремою базою знань, і вони могли б працювати незалежно, паралельно.

Серед спеціалізованих систем продукцій можна виокремити розподілені системи продукцій. Для таких систем характерним є поділ початкової бази знань на окремі компоненти, які можна було б обробляти незалежно. Оскільки застосування правил продукцій до кожної складової у даному разі проходить незалежно і порядок їх застосування не має значення, тому можлива і бажана паралельна обробка подібних систем. Але за такого підходу ми повинні вміти виражати глобальну термінальну умову у вигляді кон'юнкції термінальних умов кожної складової розподіленої бази знань. Для вирішення цієї проблеми в розподілених базах знань можна використовувати інтерпретовані розфарбовані мережі Петрі.

Вони отримуються з класичних мереж Петрі завдяки такій модифікації. З кожною позицією зв'язується дискретний перетворювач, який описує функціонування однієї з компонент розподіленої бази знань [11]. Фішки і дуги розфарбовуються кольорами. З окремим кольором пов'язується конкретна семантична інтерпретація. Дуга, розфарбована певним кольором, може пропустити тільки фішки відповідного кольору.

Традиційно бази знань мають величезні обсяги інформації, які належать до деякої предметної області. Для вилучення інформації за конкретним запитом необхідно провести дедуктивні роздуми з фактами з бази знань. Виникає проблема побудови логічно зв'язаного виведення знань (текстів) згідно з отриманими запитами природною мовою. Знання задаються фреймами.

Розглянемо задачу машинного синтезу зв'язаних текстів. Загальна схема формування зв'язаних текстів включає таку послідовність кроків: задається вхідний алфавіт; із символів алфавіту випадковим чином будуються послідовності символів довільної, досить великої довжини, і з допомогою спеціальних фільтрів з них виокремлюють змістовні фрагменти.

Значна кількість варіантів, мала вірогідність породження змістовного фрагмента показує неможливість застосування цього методу на практиці. Подальша модифікація даного методу можлива за рахунок таких факторів:

- *урахування статичних зв'язків між елементами початкового алфавіту;*
- *укрупнення початкових елементів;*
- *скорочення початкового словника шляхом введення додаткових евристик.*

Д. Поспєлов запропонував розглядати текст [56, 218] у вигляді послідовності структурних одиниць і синтезувати з них структуру всього тексту, а потім наповнювати її семантичним змістом і робити синтаксичне узгодження як між виділеними структурами, так і всередині них.

Розглянемо подібний підхід на прикладі вирішення задачі автоматизованого породження "чарівних казок". Більшість праць у сфері машинного створення "чарівних казок" базуються на ідеях В. Проппа. Він, проаналі-

зувавши структури "чарівних казок", створив функціональну модель такої казки [228]. Зупинимось на цьому докладніше.

Структура "чарівних казок" у більшості випадків постійна і включає такі типові ситуації: початкова ситуація, від'їзд батьків з дому, заборона якоїсь дії, порушення заборони і т. д. Кожна категорія персонажів має свою форму появи і до кожної категорії застосовуються особливі способи включення персонажа в хід дії казки. За В. Проппом, морфологією "чарівної казки" може бути названо будь-який розвиток від нанесення шкоди (*Ш*) або недостачі (*А*) через різні проміжні функції до кінцевої цілі. Кінцевими функціями є нагорода (*Z*), здобич або ж взагалі ліквідація горя (*Л*), порятунок від переслідування (*Сп*) і т. п. Але враховуючи паралелізм системи і неоднозначний характер функції діючих героїв (наприклад, функція "ліквідації горя" може бути реалізована кількома стандартними способами: об'єкт пошуку викрадається із застосуванням хитрощів або ж сили, або ж ...) — казка може мати кілька ходів. Для отримання множини казок визначаються варіанти з'єднання різних ходів.

Враховуючи той факт, що "чарівні казки" мають стандартну структуру, можна запропонувати таку кінцеву зв'язку функцій казок:

Б П

А В С / Д Г *Z R* {К} Л \ Пр — Сп Х Ф У О Т Н С × (*)

З Р

Деякі функції, наприклад А В С / Д Г R, присутні майже в кожній казці, а деякі — тільки в окремих випадках.

Кодування розглянутої схеми детальніше можна знайти у В. Проппа [228].

Припустимо, що кожну ситуацію в казці описує відповідний дискретний перетворювач інформації — перетворювач текстів. Він являє собою набір штампів. Штампи — фрагменти текстів, які описують конкретні ситуації при заданні деякої функції і мають нетермінальні символи. Прикладом штампів можуть бути фрази типу:

1 *дарує* # 2 # 3.

3 *переносить* # 2 *в заморське царство.*

При підстановці конкретних персонажів можемо отримувати різні фрагменти казок:

1. *Чаклун дарує царевичу вовка. Вовк переносить царевича в заморське царство.*

2. *Цар дарує Іванові коня. Кінь переносить Івана в заморське царство.*

Якщо в функцію перетворення додати і підпрограму обробки закінчень слів, тоді отримуватимемо кращі тексти. За кожний нетермінал відповідає конкретний вхід дискретного перетворювача. Наприклад, за # 1 закріплюється перший вхід (аргумент), за # 2 — другий і т. д.

Виходи дискретного перетворювача можна розділити на два типи. Перший вихід забезпечує керування друку отриманого фрагмента, інші виходи — передачу вихідних персонажів з даного фрагмента.

Повернемось до розфарбованих мереж Петрі (РМП). Якщо з позиціями ми зв'яжемо дискретні перетворювачі, а з переходами і розфарбованими стрілками — зовнішнє керування (опис загальної схеми казки (*)): аналіз вхідних параметрів (вхідних героїв), фільтрацію типів персонажів, вибір потрібних фрагментів казки, тоді отримаємо апарат побудови тексту казки. Оскільки РМП мають паралельний характер, ми отримаємо кілька варіантів казки, які відображають один сюжет.

Можлива й така реалізація нашого підходу.

Виберемо одну позицію в РМП для задання конкретних персонажів казки. Згідно з В. Проппом "чарівна казка" має фіксовану кількість типів персонажів. Припустимо, їх сім. Тоді наша РМП матиме сім типів (кольорів) фішок і сім типів дуг (кольорів дуг). Нагадаємо, що дуга даного кольору може пропускати тільки фішку відповідного кольору, що забезпечує фільтрацію і передачу потрібних персонажів між позиціями мережі. Наприклад, # 1 позначає героя, # 2 — відправника, # 3 — злодія, # 4 — дарувальника, # 5 — хибного героя, # 6 — помічника, # 7 — шуканий персонаж. Тоді в першій позиції іде призначення імен даним персонажам. Наприклад, # 1 — Іван-дурник, # 2 — цар-батечко і т. д. Тоді загальну схему казки можна описати РМП з різними варіантами опису базових ситуацій. Подібний підхід до синтезу машинних казок був реалізований у [79]. Дана мережа була змодельована програмою в мові Паскаль. Отримані варіанти казок засвідчили хороше практичне застосування запропонованого підходу.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте поняття моделі.
2. Охарактеризуйте основні етапи процесу моделювання.
3. Наведіть відому вам класифікацію моделей.
4. Що таке математична модель?
5. Перелічіть відомі вам вимоги до математичних моделей.
6. Які ви знаєте типи математичних моделей?
7. Охарактеризуйте поняття імітаційного моделювання.
8. Дайте визначення мережі Петрі.
9. Яким чином мережі Петрі пов'язані з моделюванням причинно-наслідкових зв'язків?
10. Перелічіть відомі вам властивості мереж Петрі.
11. Що таке розфарбовані мережі Петрі?
12. Охарактеризуйте застосування мереж Петрі до генерації зв'язних текстів.

ЧАСТИНА VI | РОЗПІЗНАВАННЯ ОБРАЗІВ

Розділ 25
ОСНОВНІ ПРИНЦИПИ РОЗПІЗНАВАННЯ

— Тату, а що означає "двоїться в очах"?
— Бачиш, синку, он дві берези на пагорбі.
Коли двоїться в очах, здається, що їх чотири.
— Тату, але там лише одна береза...

З анекдоту

25.1. Основні постановки задач розпізнавання

Розпізнавання образів є однією з найфундаментальніших проблем теорії інтелектуальних систем. З іншого боку, задача розпізнавання образів має величезне практичне значення [38, 39, 46, 252, 260, 264].

Замість терміна "розпізнавання" часто вживається інший — "класифікація". Ці два терміни у багатьох випадках розглядаються як синоніми, але не є повністю взаємозамінюваними. Кожний з них має свої сфери застосування, і інтерпретація обох термінів часто залежить від специфіки конкретної задачі. Ми вважатимемо обидва терміни синонімами, якщо не буде явно обумовлене щось інше.

Наведемо деякі типові постановки задач класифікації.

1. Задача ***ідентифікації***, яка полягає в тому, щоб вирізнити певний конкретний об'єкт серед йому подібних (наприклад, впізнати серед інших людей свою дружину).
2. ***Віднесення об'єкта до того чи іншого класу***. Це може бути, наприклад, задача розпізнавання літер або прийняття рішення про наявність дефекту у деякій технічній деталі. Віднесення об'єкта до певного класу відображає найтиповішу проблему класифікації, і, коли говорять про розпізнавання образів, найчастіше мають на увазі саме цю проблему. Тому ми розглядатимемо її першочергово.
3. ***Кластерний аналіз***, який полягає в поділі заданого набору об'єктів на класи — групи об'єктів, схожі між собою за тим чи іншим критерієм. Цю задачу часто називають ***класифікацією без учителя***, оскільки, на відміну від задачі 2, класи апріорно не задані.

Як уже зазначалося, проблеми розпізнавання легко вирішуються людьми, як правило, підсвідомо. Спроби ж побудувати штучні системи розпізнавання не настільки переконливі. Основна проблема полягає в тому, що

часто неможливо адекватно визначити ознаки, на основі яких слід здійснювати розпізнавання. Для задач, для яких такі ознаки вдається виокремити, штучні системи розпізнавання набули значного поширення.

Методи розпізнавання образів і технічні системи, що реалізують ці методи, широко використовуються на практиці. Наведемо деякі з них.

Технічна діагностика. На виробництві часто виникає потреба автоматизувати контроль якості деталей. Задача полягає в тому, щоб виявити, чи є деталь дефектною, чи ні. Якщо ж з'ясовується, що деталь має дефект, часто потрібно визначити тип цього дефекту.

Медична діагностика. Системи розпізнавання часто використовуються і в медичній практиці. Найтиповіша ситуація полягає в тому, що ті чи інші захворювання діагностуються на основі аналізу кардіограм, рентгенівських знімків і т. п.

Розпізнавання літер. Окрім усього іншого, ця проблема має велике значення для власне комп'ютерних технологій. Системи розпізнавання літер працюють разом зі *сканерами* — пристроями, які використовуються для введення до комп'ютера друкованих зображень і текстів. При введенні друкованого тексту сканер формує лише графічне зображення; для того щоб створити текстовий документ, з яким може працювати текстовий редактор, необхідно впізнати на цьому зображенні окремі літери. Аналогічно розпізнавання літер є необхідним для підтримки *пристроїв рукописного введення*. Цими пристроями, зовні схожими на звичайну авторучку, часто комплектуються надпортативні комп'ютери (персональні помічники). Основна мета цих пристроїв — замінити введення з клавіатури, що є незручним для багатьох користувачів.

Розпізнавання мови. Сьогодні інтенсивно розвиваються технології, пов'язані, по-перше, з голосовим керуванням комп'ютером, а по-друге — з введенням текстів з голосу.

Робототехніка. Застосування методів розпізнавання в робототехніці є абсолютно природним і необхідним, оскільки роботи повинні безпосередньо сприймати зовнішній світ і, відповідно, мати пристрої машинного зору.

Охоронні системи. Застосування методів розпізнавання в охоронних системах пов'язано в першу чергу з проблемою ідентифікації. Наприклад, потрібно ідентифікувати певну особу, щоб визначити, чи має вона право входити на територію, що охороняється. Розвиваються також системи, які вирішують проблему ідентифікації відбитків пальців і т. п.

25.2. Класи та їх властивості

Одне з найвдаліших формулювань ключової парадигми теорії розпізнавання можна знайти в [264]: **будь-який об'єкт у природі є унікальним, всі об'єкти є типізованими.**

Зміст цієї парадигми такий. Кожний об'єкт характеризується тими чи іншими властивостями. Наявність чи відсутність таких властивостей, а також якісні та кількісні характеристики цих властивостей розглядаються як

ознаки об'єкта. Унікальність будь-якого об'єкта означає те, що в природі не існує двох різних об'єктів, для яких збігаються абсолютно всі ознаки, і це дозволяє, принаймні теоретично, відрізнити один об'єкт від іншого. Але деякі ознаки різних об'єктів можуть збігатися, і це дає підстави говорити про те, що ці об'єкти належать до одного типу або класу.

Фундаментальні поняття "клас" та "об'єкт" неможливо повністю формалізувати. Спробуємо навести їх неформальні визначення.

***Об'єктом** у теорії розпізнавання прийнято називати будь-яку сутність, що існує або могла б існувати в реальному світі, а також будь-яке явище або процес.*

Це дуже широке визначення, подальші уточнення можуть бути пов'язані з тим чи іншим звуженням нашого розуміння про те, що саме слід вважати об'єктом. Так, реально існуюча Ейфелева башта буде вважатися об'єктом практично у будь-якій інтерпретації, а "розвиток наукових досліджень про етнокультурний стан Ефіопії та Антарктиди за період з липня 1898 року по січень 1927 року" — можливо, ні. Тут усе залежить від специфіки конкретної задачі і від мети, з якою вона вирішується.

***Класом** у теорії розпізнавання образів прийнято називати сукупність об'єктів, які мають ті чи інші спільні ознаки.*

Клас може об'єднувати фізично існуючі сутності (наприклад, людина, яблуко) або бути абстрактним поняттям (горе, економічний розвал і т. п.).

Ознаки, що дають можливість відрізнити представників одного класу від іншого, прийнято називати *інформативними ознаками*.

Ознаки, спільні для всіх представників класу, називатимемо *інваріантами класу*.

У [38] наведено дещо формальніше визначення класу: класом називається сукупність об'єктів, пов'язаних між собою деяким відношенням еквівалентності, або, в крайньому разі, толерантності. Нагадаємо, що відношенням еквівалентності називається відношення, яке є симетричним, рефлексивним і транзитивним (наприклад, відношення "дорівнювати"). Для відношення ж толерантності властивість транзитивності в цілому не виконується (наприклад, відношення "бути схожим").

Набори інформативних та інваріантних ознак можуть збігатися, але це зовсім не обов'язково (наведіть відповідні приклади).

Інколи, щоб уникнути непорозумінь, ми називатимемо об'єкти і класи реального світу відповідно *Р-об'єктами* і *Р-класами*. Зрозуміло, що *Р*-об'єкт може належати будь-якій кількості *Р*-класів.

Р-об'єкти часто називаються *реалізаціями*, або *зразками Р*-класів.

Можна виокремити такі основні властивості класів [38]:

1. Усі представники класу мають **певний набір спільних ознак** (випливає з визначення).
2. **Змінюваність реалізацій** класів. По-перше, різні об'єкти, що належать одному класу, можуть бути не схожими між собою. Вони

повинні мати спільні інваріантні ознаки, але всі інші ознаки можуть як завгодно варіювати. По-друге, один і той самий об'єкт може змінюватися з часом і навіть поступово переходити від одного класу до іншого (наприклад, перетворення пуголовка на жабу). Усе це свідчить про те, що розпізнати чіткі межі класу часто неможливо.

3. **Ознайомлення з деякою скінченною кількістю представників даного класу дає можливість впізнавати інших представників цього класу.** Взагалі кажучи, ця властивість може не виконуватися. Але якщо ця властивість виконується (принаймні, теоретично), це є дуже сильним і важливим твердженням. Воно по суті означає можливість навчатися на прикладах, тобто на основі спостереження певної кількості прикладів (можливо, разом з контрприкладами), сформулювати *правило розпізнавання*, яке дає змогу відрізняти представників даного класу від представників іншого (можливо, з певною достовірністю, тобто з певним процентом помилок). У деяких випадках правилом розпізнавання може бути предикат, який залежить від інформативних або інваріантних ознак, інколи правило розпізнавання реалізується у вигляді деякої складної процедури.

Якщо навчання на прикладах неможливе або неефективне, правило розпізнавання інколи можна задати явно. Якщо цього не вдається, єдиною можливістю для надійного розпізнавання залишається запам'ятовування всіх можливих представників даного класу. Цей випадок не становить інтересу з теоретичної точки зору, і його можна реалізувати лише, якщо кількість можливих представників не є надто великою.

25.3. Модельні описи класів. Розпізнавання як зіставлення

Інтелектуальна система не в змозі сприймати реальні об'єкти (*P*-об'єкти) безпосередньо. Вона сприймає їх за допомогою органів чуття, фізичних датчиків, різноманітних технічних пристроїв і т. п. При цьому формуються первинні описи об'єктів, які ми називатимемо *С-об'єктами*.

С-об'єкт *— первинний опис Р-об'єкта у тому вигляді, в якому він сприймається інтелектуальною системою.*

Ми вважаємо, що інтелектуальна система може сприймати лише окремих представників класу, але не весь клас в цілому. Відповідно до цього поняття "*С*-класу" ми не розглядаємо.

С-об'єкти можна було б називати образами, але в літературі термін "образ" часто вживається як синонім терміна "клас". *С*-об'єкти часто (хоча й не завжди) є функціями зміни тих чи інших фізичних величин. Наприклад, при розпізнаванні зображень первинним описом може бути функція яскравості зображення в різних точках.

Процедури сприйняття можуть послаблювати або посилювати природну змінюваність *Р*-класів і *Р*-об'єктів. *С*-об'єкти як результати спостереження *Р*-об'єктів істотно залежать від умов спостереження. Одному і тому

самому *Р*-об'єкту можуть відповідати різні *С*-об'єкти. Так, деякий предмет може сприйматися по-різному, якщо його спостерігати під різними кутами і з різної відстані. На формування *С*-об'єктів справляють значний вплив зовнішні шуми (наприклад, прекрасна музика П. Чайковського спотворюється неякісною апаратурою або ревом реактивного двигуна).

З іншого боку, і різним *Р*-об'єктам може відповідати один і той самий *С*-об'єкт. Таке можливо, наприклад, якщо два *Р*-об'єкти належать одному класу, і з високою мірою точності вимірюються лише інваріантні ознаки цього класу. Тоді *С*-об'єктами є набори таких ознак, і природно, що вони збігаються.

Ми знаємо, що інтелектуальна система завжди зіставляє первинні описи зі своїми знаннями. Відповідно до цього в основі розпізнавання лежить метапроцедура зіставлення зі зразком. В результаті такого зіставлення формується вторинний опис об'єкта, який ми називатимемо *М-об'єктом* (модельним описом, моделлю об'єкта). Модельним описом може бути просто набір інформативних ознак або складніша структура, наприклад, семантична мережа.

У пам'яті системи зберігаються також модельні описи різних класів, і ці модельні описи класів будемо називати *М-класами*. *М*-класи багато в чому аналогічні фреймам, а *М*-об'єкти — екземплярам фреймів (тут доречно нагадати, що в термінології об'єктно-орієнтованого програмування фрейми називаються саме класами).

Тепер можна сформулювати загальний принцип розпізнавання.

С-об'єкти, що спостерігаються системою, або відповідні їм М-об'єкти (модельні описи об'єктів), що формуються в процесі роботи, зіставляються з М-класами (модельними описами класів), і система приймає рішення про віднесення об'єкта до того класу, зіставлення з модельним описом якого було найбільш успішним.

Безумовно, цей загальний принцип вимагає уточнення і конкретизації.

25.4. Постановка задачі і основні режими розпізнавання

Тут і далі ми розглядатимемо розпізнавання як задачу віднесення об'єкта до того чи іншого класу. Сформулюємо постановку цієї задачі жорсткіше.

Нехай є *m* класів. Задача розпізнавання полягає у віднесенні об'єкта, який спостерігається системою, до одного з них.

Виокремлюють три режими функціонування системи розпізнавання: робочий режим, навчання та екзамен.

У *робочому режимі* система розпізнавання повинна прийняти рішення про віднесення об'єкта, який спостерігається системою, до того чи іншого класу. Критерії, які використовуються для прийняття такого рішення, і власне процедура прийняття рішення утворюють правило розпізнавання.

Загальну схему розпізнавання в робочому режимі можна зобразити таким чином:

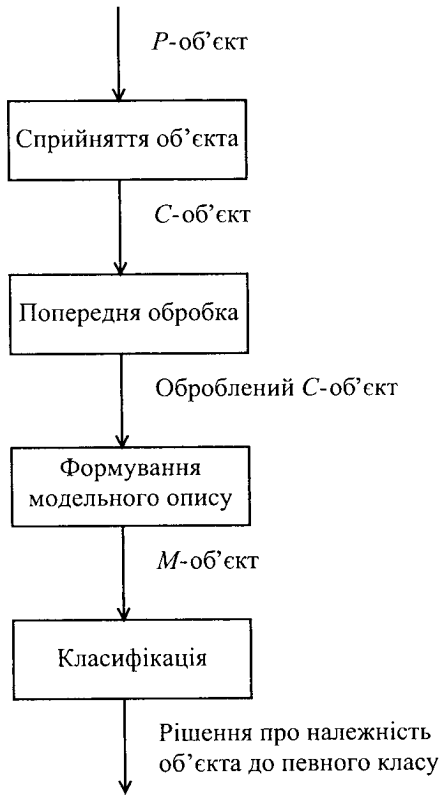


Рис. 25.1. Типова схема розпізнавання

Система розпізнавання повинна насамперед сприйняти реальний об'єкт. Люди і тварини сприймають об'єкти реального світу за допомогою органів чуття. Технічні системи використовують для цього датчики фізичних величин. Уже зазначалося, що зображення можна описати як двовимірну функцію, що задає зміну яскравості, і при аналізі зображень вимірюється яскравість зображення в різних точках. Аналогічно кардіограма являє собою функцію, яка залежить від часу. Найпростіші датчики просто здійснюють *дискретизацію* (див. розд. 28), але за допомогою спеціально сконструйованих датчиків можуть вимірюватися й інші характерні ознаки.

Блок попередньої обробки здійснює необхідну попередню обробку первинного опису. Основна його задача — зменшення рівня шумів, що виникають внаслідок вимірювальних похибок, впливу сторонніх факторів, недостатньої якості вимірювальної апаратури і т. п.

Блок формування модельного опису формує модельний опис об'єкта. Як уже зазначалося, модельним описом може бути просто набір інформативних ознак або ж структура об'єкта.

Нарешті, блок класифікації (класифікатор) застосовує до модельного опису правило розпізнавання і на основі цього виробляє рішення про належність об'єкта до того чи іншого класу.

Перед функціонуванням системи розпізнавання в робочому режимі часто буває доцільно здійснити ***навчання*** цієї системи. До основних цілей навчання можна віднести формування правила розпізнавання, а також вибір множини інформативних ознак.

Навчання здійснюється на основі навчальної вибірки, якою називається деяка множина об'єктів, причому про кожний з них відомо, до якого класу він належить. Як правило, до навчальної вибірки включають достатню кількість представників кожного класу.

Після завершення навчання доцільно провести ***екзамен***, метою якого є оцінка якості навчання, передусім оцінка надійності сформованого правила розпізнавання. Екзамен здійснюється на основі екзаменаційної вибірки (певний набір об'єктів, про які відомо, до якого класу вони належать). Здійснюється розпізнавання об'єктів екзаменаційної вибірки, і на основі порівняння відповідей з правильними робиться висновок про якість навчання і можливість переходу до робочого режиму.

Можна виокремити такі типові схеми роботи системи розпізнавання:

- *схема без навчання*, за якої навчання не здійснюється, а правило розпізнавання жорстко задається під час проектування і розробки системи;
- *схема з одноразовим навчанням*, за якої правило розпізнавання виробляється один раз і надалі не змінюється;
- *схема з періодичним перенавчанням*, за якої навчання періодично повторюється. При цьому навчальна вибірка може поповнюватися за рахунок об'єктів, які розпізнавалися в робочому режимі.

25.5. Розпізнавання як прийняття рішень

Задачу розпізнавання образів можна уявляти як задачу прийняття рішень. Справді, розпізнавання і є прийняття рішення про те, до якого класу належить той чи інший об'єкт.

Насамперед розглянемо розпізнавання як деяку "гру з природою". Нехай є m класів, і потрібно здійснити розпізнавання деякого об'єкта. Система розпізнавання має вибір між m стратегіями: i-та стратегія означає прийняття рішення про те, що об'єкт відноситься до i-го класу. Природа також має одну з m стратегій: j-та стратегія означає те, що насправді об'єкт належить до j-го класу.

Маємо матрицю ризиків $R = (r_{ij}, i, j = 1, \ldots, m)$; r_{ij} — втрати від віднесення об'єкта до i-го класу, тоді як насправді він належить до j-го.

Очевидно, **метою системи розпізнавання є мінімізація середнього ризику.**

Інтуїтивно зрозумілими є такі властивості матриці ризиків:

- $r_{ij} \le 0$ при $i = j$; $r_{ij} \ge 0$ при $i \ne j$;
- матриця ризиків, як правило, асиметрична. Дійсно, розглянемо такий приклад. Нехай на виробництві діє система автоматизованого контролю якості вузлів літака. Є два класи: деталі без дефектів та дефектні. Якщо деталь приймається за справну, вона використовується, якщо ж ні — вона викидається. Очевидно, що використати в літаку дефектну деталь може бути значно гіршим, ніж викинути справжню.

25.6. Класифікація основних методів розпізнавання

Перш за все слід згадати прості спеціалізовані методи розпізнавання на основі фіксованих наборів еталонних ознак. Самі об'єкти при цьому вважаються відносно незмінними. Кожний з цих методів розрахований на окрему специфічну задачу.

Існує декілька груп таких методів [38]. Для прикладу згадаємо один з них — ***метод зіставлення з еталоном***. Основний принцип полягає в порівнянні зображення з набором попередньо сформованих еталонів. Порівняння здійснюється з використанням спеціальних технічних пристроїв. Можуть застосовуватися, зокрема, фотомаски, матриці електричних елементів і т. п. Наприклад, при використанні фотомасок зображення безпосередньо накладається на еталонні маски, набір яких охоплює всю множину еталонів. Міра збігу визначається за допомогою фотоелемента, розташованого за маскою.

Змінюваність реалізацій класів вимагає розвитку універсальніших методів. Основна відмінність між різними групами методів розпізнавання полягає в тому, яким чином формуються модельні описи об'єктів.

Найбільш вживаною є класифікація, згідно з якою методи розпізнавання образів розділяються на *дискримінантні* (часто вони називаються розпізнаванням у просторі ознак) і *структурні*. При використанні дискримінантних методів розпізнавання кожний об'єкт описується точкою у просторі ознак, тобто координатами цієї точки виступають значення ознак об'єкта. Відповідно до цього основною метою навчання є поділ усього простору на зони, кожна з яких відповідає певному класу.

Дискримінантні методи розпізнавання, описані в розд. 26, у свою чергу поділяються на *ймовірнісні* та *детерміністські*.

При використанні ймовірнісних методів кожний об'єкт описується вектором, який розглядається як реалізація деякого випадкового вектора. Якщо відомі ймовірнісні характеристики розподілу випадкових векторів, то можна визначити ймовірності, з якими об'єкт може належати до того чи іншого класу, і на основі цього прийняти рішення про класифікацію. В основі ймовірнісних методів розпізнавання лежить прийняття рішень на основі байєсівського підходу.

При використанні детерміністських методів жодних імовірнісних припущень не робиться, а рішення приймається на основі аналізу геометричних мір близькості між різними точками.

Структурні методи залучають до розгляду структуру об'єктів, які розпізнаються. Вони поділяються на *синтаксичні* (інша назва — *лінгвістичні*) і *логічні*.

При використанні синтаксичних методів (розд. 27) кожний клас описується формальною мовою, породженою деякою формальною граматикою. Об'єкт, що розпізнається, описується певною фразою, і головний принцип розпізнавання полягає у тому, щоб визначити, до якої мови належить отримана фраза.

Нарешті, логічні методи розпізнавання ґрунтуються на аналізі комбінацій характерних ознак. Кожний клас описується деяким предикатом, і об'єкт належить до класу, якщо відповідний предикат приймає значення "істина". Використання логічних методів розпізнавання — це, по суті, реалізація деякої експертної системи.

Можна розглядати й іншу класифікацію методів розпізнавання, яка враховує, чи є помилки на навчальній вибірці. Використання деяких методів гарантує, що в робочому режимі всі об'єкти навчальної вибірки розпізнаватимуться безпомилково. Для інших методів це не так.

25.7. Поняття про допустимі перетворення

Якість розпізнавання може бути істотно підвищена, якщо залучити до розгляду описи модифікацій, яких може зазнавати об'єкт, залишаючись при цьому в межах класу. Так, при розпізнаванні літер одна з них може бути повернута на деякий градус, стиснута або розтягнута; вона може бути дещо деформованою і т. п. Такі модифікації, які залишають будь-який M-об'єкт в межах одного й того самого класу, прийнято називати ***допустимими***, або ***інваріантними***, перетвореннями.

Нехай система розпізнавання сприйняла деякий об'єкт і сформувала його модельний опис M. Нехай у навчальній вибірці існує еталонний об'єкт M^*, який однозначно належить до класу K. Основна ідея *методу допустимих перетворень* полягає в такому. Якщо система розпізнавання знаходить інваріантне стосовно класу K перетворення, яке переводить M^* до M, то система приймає рішення про віднесення M до K.

Використання припустимих перетворень є, очевидно, характернішим для синтаксичних і логічних методів розпізнавання, хоча його можна застосовувати і для дискримінантних методів. Апарат допустимих перетворень широко використовується при розпізнаванні зображень, особливо літер, і при розпізнаванні мови. Значний вклад у розвиток методу припустимих перетворень внесли В. Ковалевський і Т. Вінцюк.

Детальніший опис методу припустимих перетворень можна знайти у довіднику [38].

Сьогодні ведуться роботи в напрямі створення загальної теорії розпізнавання. З даного погляду слід відмітити **теорію комбінаторної регулярності Гренандера** [91], яка розглядає правила побудови складних об'єктів з

простіших; **алгебраїчний підхід до задач розпізнавання та класифікації**, що був розвинений у працях Ю. Журавльова [123] та ін.

Опис математичної теорії розпізнавання образів можна знайти в [129]. При цьому значна увага приділяється побудові алгоритмів, що є оптимальними за тими чи іншими критеріями. Розвивається також теорія, яка вивчає міри складності алгоритмів розпізнавання [240].

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Опишіть відомі вам постановки задачі класифікації.
2. Перелічіть відомі вам практичні способи застосування методів класифікації.
3. Наведіть ключову парадигму теорії розпізнавання, сформульовану Хантом.
4. Що таке клас у теорії розпізнавання?
5. Що таке інваріанти класу?
6. Що таке інформативні ознаки?
7. Перелічіть основні властивості класів.
8. Які фактори визначають змінюваність реалізацій класів?
9. Охарактеризуйте роль, яку відіграє в задачі розпізнавання образів навчання на прикладах.
10. Охарактеризуйте процес розпізнавання з точки зору зіставлення зі зразком.
11. Сформулюйте задачу розпізнавання як задачу віднесення об'єкта до певного класу.
12. Охарактеризуйте типову схему розпізнавання в робочому режимі.
13. Що таке датчики?
14. У чому полягає етап попередньої обробки при розпізнаванні?
15. Охарактеризуйте етап навчання, який передує розпізнаванню в робочому режимі. Які основні задачі вирішуються на цьому етапі?
16. Що таке навчальна вибірка?
17. Охарактеризуйте схеми без навчання, з одноразовим та періодичним перенавчанням.
18. У чому полягає режим "екзамену"?
19. Охарактеризуйте задачу розпізнавання як задачу прийняття рішень.
20. Наведіть класифікацію основних методів розпізнавання.
21. Дайте загальну характеристику дискримінантних методів.
22. Дайте загальну характеристику структурних методів.
23. Охарактеризуйте застосування методу припустимих перетворень у розпізнаванні образів.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. У чому полягають основні проблеми при створенні штучних систем розпізнавання?
2. Наведіть власний приклад матриці ризиків при розпізнаванні об'єктів з довільної предметної області.

Розділ 26

РОЗПІЗНАВАННЯ В ПРОСТОРІ ОЗНАК

Усе французьке повинно бути
на французькій стороні, а все
італійське — на італійській.

З кінофільму "Закон є закон"

26.1. Загальна характеристика дискримінантних методів розпізнавання

Як уже зазначалося, для дискримінантного розпізнавання, або для розпізнавання в просторі ознак, характерним є те, що кожний об'єкт зображається окремою точкою в деякому просторі. Координатними осями цього простору виступають ознаки, за якими здійснюється розпізнавання. Таким чином, координатами об'єктів виступають значення відповідних ознак. Далі розпізнавання здійснюється на основі аналізу мір близькості між об'єктами.

Існує величезна кількість дискримінантних методів розпізнавання. Тут ми обмежимося розглядом лише найпростіших детерміністських методів. Повніший виклад проблеми можна знайти, наприклад, в [38, 46, 107, 252, 260, 263].

26.2. Типи ознак, міри відстаней

Виокремлюють різні типи ознак: *дихотомічні* (ознака може бути присутня або відсутня; наприклад, є крила або немає крил); *номінальні* (наприклад, колір: червоний, синій, зелений і т. п.); *порядкові* (наприклад, великий — середній — маленький); *кількісні*. Для кожного типу ознак можна вводити свої міри відстані між об'єктами.

Ми обмежимося лише кількісними ознаками, які можуть мати числові значення. Для таких ознак, як міри близькості між об'єктами, найчастіше беруть звичайні *евклідові відстані* або їх квадрати. Нагадаємо, що евклідова відстань в n-вимірному метричному просторі між векторами $a = (a_1, \ldots, a_n)$ та $b = (b_1, \ldots, b_n)$ обчислюється за формулою:

$$D(a,b) = \sqrt{\sum_{i=1}^{n} (a_i - b_i)^2}.$$

Якщо різні ознаки повинні враховуватися різною мірою, кожній ознаці приписується вага w_i, і використовується *зважена евклідова відстань*:

$$D(a,b) = \sqrt{\sum_{i=1}^{n} w_i (a_i - b_i)^2}.$$

Існують спеціальні методики, які дозволяють здійснювати перехід від некількісних типів ознак до кількісних.

26.3. Вектори та матриці ознак

Будь-який об'єкт при застосуванні дискримінантних методів розпізнавання, як ми пересвідчилися, зображається вектором ознак у деякому n-вимірному просторі, де n — кількість ознак. Таким чином, будь-якому об'єкту відповідає n-вектор:

$$x = (x_j; j = 1, \ldots, n);$$

де x_j — значення j-ї ознаки.

Навчальну вибірку прийнято позначати у вигляді спеціальної матриці, яка часто називається матрицею даних:

$$Y = (y_{ij}; i = 1, \ldots, q; j = 1, \ldots, n + 1).$$

Тут q — загальна кількість представників усіх класів, n — кількість ознак; y_{ij} $(j = 1, \ldots, n)$ — значення j-ї ознаки для i-го об'єкта; $y_{i,\,n+1}$ — клас, до якого належить i-й об'єкт.

26.4. Гіпотеза компактності

Усі дискримінантні методи так чи інакше спираються на ***гіпотезу компактності*** [38]. Відповідно до цієї гіпотези класу відповідає компактна множина точок у деякому просторі ознак.

Слово *"компактний"* тут вживається у дещо незвичному значенні. Компактною у даному розумінні називається множина точок, для якої:

- число граничних точок мале порівняно із загальним числом точок;
- будь-які дві внутрішні точки можуть бути з'єднані плавною лінією так, щоб ця лінія проходила лише через точки цієї ж множини;
- майже будь-яка внутрішня точка має в достатньо великому околі лише точки цієї ж множини.

Важливість належного вибору простору ознак, в якому класам відповідають компактні множини точок, ілюструють такі приклади. На усіх рисунках представники одного класу позначені кружечками, а іншого — зірочками.

Приклад 26.1 (лінійна роздільність).

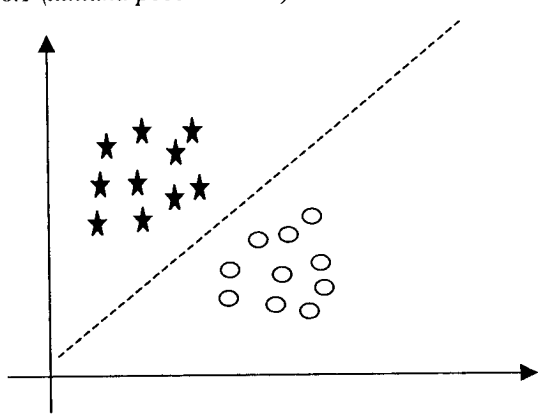


Рис. 26.1. Лінійна роздільність класів

У даному випадку ознаки обрано дуже вдало. Класи добре відокремлені один від одного. Більше того, можна провести пряму лінію так, що всі представники першого класу будуть розміщуватися по один бік цієї лінії, а всі представники другого класу — по інший бік. У такому разі кажуть, що класи є *лінійно роздільними*.

Лінійна роздільність дуже спрощує задачу розпізнавання. Правило розпізнавання на основі лінійних роздільних функцій є дуже простим і не вимагає великих затрат часу і пам'яті. Роздільну пряму лінію дуже легко побудувати (деякі алгоритми обговорюватимуться в п. 26.6).

Приклад 26.2 (роздільність класів).

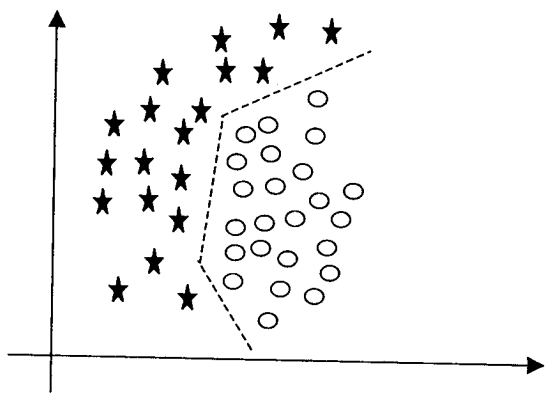


Рис. 26.2. Ілюстрація роздільності класів

У цьому прикладі лінійної роздільності немає. Але все-таки можна побудувати лінію, хоча й непряму, але яка відокремлює один клас від іншого. Процедури навчання і розпізнавання в даному разі суттєво ускладнюються.

Приклад 26.3 (часткове перекриття класів).

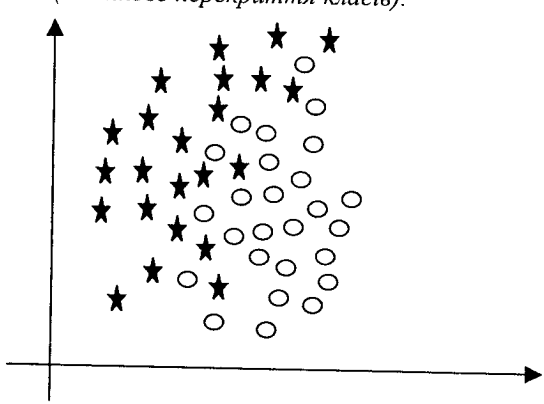


Рис. 26.3. Класи частково перекриваються

У цьому випадку надійно відокремити один клас від іншого не вдається, і це значно збільшує кількість помилок при розпізнаванні.

Приклад 26.4 (повне перекриття класів).

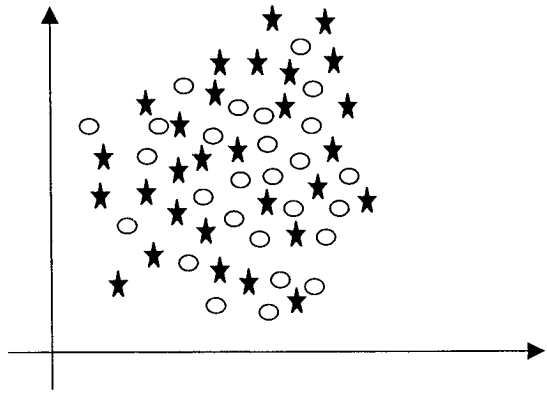


Рис. 26.4. Класи повністю перекриваються

Ні про яке розпізнавання в такому просторі ознак говорити, очевидно, не доводиться.

26.5. Типова схема розпізнавання в просторі ознак

Схема розпізнавання в просторі ознак повністю відповідає загальній схемі, наведеній в п. 25.4. В даному випадку "модельним описом" об'єкта є його вектор ознак, і тому "формування модельного опису об'єкта" є не що інше, як виокремлення ознак, отримання їх числових значень.

Типова схема розпізнавання в просторі ознак наведена на рис. 26.5.

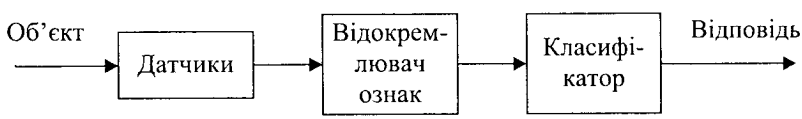


Рис. 26.5. Розпізнавання в просторі ознак

Виокремлення ознак не слід змішувати з іншою задачею — задачею *вибору інформативних ознак*, за якими слід здійснювати класифікацію. Такий вибір повинен відбуватися на етапі навчання. На етапі розпізнавання просто визначаються конкретні значення ознак для всіх об'єктів, що розпізнаються.

Легко зрозуміти, що всі ці етапи тісно пов'язані між собою. Уже з даного прикладу видно, що ідеальний класифікатор не потребує виокремлення ознак, і навпаки, при ідеальному виокремленні ознак робота класифікатора максимально спрощується. Основну проблему можна сформулювати таким чином: реальні методи отримання первинної інформації та доступні алгоритми виокремлення ознак часто не дозволяють отримати саме ті ознаки, які насправді відрізняють представників одного класу від представників іншого.

26.6. Роздільні функції. Лінійні роздільні функції

Для простоти розглядатимемо випадок двох класів; методологія, що буде описана, легко поширюється на загальніший випадок.

Отже, нехай маємо два класи: K_1 і K_2. Нехай довільний вектор x описується своїми координатами: $x = (x_1, \ldots, x_n)$. Кажуть, що функція $g(x)$ ***розділяє*** ці класи, або є ***роздільною функцією***, якщо для всіх векторів класу K_1 $g(x) > 0$, а для всіх векторів класу K_2 $g(x) < 0$.

Побудова і використання роздільних функцій є основою багатьох методів розпізнавання. Насправді в робочому режимі ми, як правило, не можемо знати заздалегідь, до якого класу належить даний об'єкт (саме це і потрібно визначити). Натомість, якщо у нас є деяка роздільна функція, що є наближенням до справжньої роздільної функції, ми можемо на основі її аналізу прийняти певне рішення, а саме: рішення про належність вектора x до класу K_1, якщо $g(x) > 0$, і класу K_2, якщо $g(x) < 0$. Це рішення може виявитися помилковим. Таким чином, метою навчання має бути отримання справжньої роздільної функції, якщо вона існує, або побудова деякої наближеної роздільної функції в іншому випадку. Саме такий зміст ми надалі вкладатимемо в поняття "роздільна функція".

Як уже зазначалося, особливо важливим є випадок, коли роздільна функція є *лінійною*, тобто є рівнянням гіперплощини і має вигляд:

$$g(x) = w_1 x_1 + \ldots + w_n x_n + w_{n+1},$$

коефіцієнти w_i повинні бути визначені в процесі навчання.

Одним з найпростіших методів підбору параметрів є *алгоритм персептрона*, названий так через свій тісний зв'язок з навчанням персептронів (див. розд. 31).

Алгоритм полягає в такому. Задається початковий вектор параметрів $w^{(0)}$, який уточнюється по мірі розпізнавання елементів навчальної вибірки. Правила уточнення такі:

якщо черговий елемент навчальної вибірки $x^{(i)}$ належить K_1, а $g(x) \le 0$, то

$$w^{(i+1)} := w^{(i)} + cx^{(i)};\ c > 0;$$

якщо черговий елемент навчальної вибірки $x^{(i)}$ належить K_2, а $g(x) \ge 0$, то

$$w^{(i+1)} := w^{(i)} - cx^{(i)};\ c > 0.$$

У разі лінійної роздільності алгоритм збігається.

Кусково-лінійні роздільні функції є складнішими, ніж лінійні, і побудова їх також непроста. З іншого боку, кусково-лінійна роздільна функція може з достатньою мірою точності апроксимувати практично будь-яку роздільну функцію. Одна з процедур побудови кусково-лінійних роздільних функцій була запропонована А. Аркадьєвим і Е. Браверманом [260].

26.7. Метод найближчого сусіда

Метод найближчого сусіда є одним з найвідоміших і класичних дискримінантних методів розпізнавання. Його можна сформулювати таким чином:

Об'єкт відноситься системою розпізнавання до того класу, до якого належить його найближчий сусід з навчальної вибірки.

Метод не дає помилок на об'єктах навчальної вибірки. Його застосування фактично еквівалентне використанню деякої кусково-лінійної роздільної функції, хоч ця функція в явному вигляді не отримується. Тому метод не вимагає попереднього формування розв'язувального правила. Але його недоліком є те, що в пам'яті доводиться зберігати всю навчальну вибірку і знаходити відстані до кожного її елемента. Це призводить до великих затрат як пам'яті, так і часу.

У разі застосування правила найближчого сусіда існує можливість провести попереднє навчання, в процесі якого скоротити повну навчальну вибірку. Для цього можна використовувати так зване ***стиснене правило найближчого сусіда***, запропоноване Е. Хартом. Далі наводиться алгоритм цього правила, описаний на основі [263].

Для роботи алгоритму потрібні дві робочі множини — ПАМ'ЯТЬ і ВІДСІВ. Алгоритм виконує такі дії:

1. У ПАМ'ЯТЬ заноситься по одному представнику кожного класу.
2. Кожний наступний об'єкт навчальної вибірки класифікується за допомогою правила найближчого сусіда, але при цьому використовуються лише ті об'єкти, які знаходяться в множині ПАМ'ЯТЬ. Якщо даний об'єкт класифікується правильно, він викидається у ВІДСІВ, інакше він заноситься в ПАМ'ЯТЬ.
3. Після перегляду всіх об'єктів описана процедура повторюється знову, але лише для елементів множини ВІДСІВ. Процедура завершується або при повному вичерпанні ВІДСІВ, або якщо при черговому проході жодний об'єкт з ВІДСІВ не перейшов до ПАМ'ЯТЬ.
4. Після завершення роботи алгоритму як скорочена навчальна вибірка використовується множина ПАМ'ЯТЬ.

Модифікацією методу найближчого сусіда є ***правило k найближчих сусідів***. Об'єкт відноситься системою розпізнавання до того класу, до якого належить більшість з k його найближчих сусідів з навчальної вибірки.

26.8. Байєсівські методи розпізнавання

Байєсівські методи належать до ймовірнісних методів розпізнавання. Вони залучають до розгляду ймовірнісні характеристики класів, і врахування такої інформації дозволяє суттєво підвищити якість розпізнавання.

Основну ідею байєсівських методів можна неформально охарактеризувати таким чином. Нехай відомі ймовірнісні характеристики того, що об'єкт, який належить до певного класу, описується деяким вектором ознак. Якщо ми отримали вектор ознак, який потрібно розпізнати, ми можемо на основі правил Байєса розрахувати ***апостеріорні ймовірності*** належності цього вектора до кожного класу. Далі об'єкт відноситься системою розпізнавання до того класу, для якого ця апостеріорна ймовірність найбільша (інакше кажучи, до найвірогіднішого класу).

Дамо формалізованіший опис байєсівських методів.

Нехай ми маємо m класів $K_1, \ldots, K_m$. З кожним класом пов'язана апріорна ймовірність $P(K_i)$ — ймовірність появи об'єкта i-го класу. Нагадаємо, що ймовірності тісно пов'язані з частотою, і тому ймовірності $P(K_i)$, по суті, характеризують, наскільки часто зустрічаються об'єкти відповідних класів.

Основною ймовірнісною характеристикою, яка встановлює розподіл векторів певного класу в просторі ознак, є ***умовна густина розподілу***. Дамо необхідні визначення.

Функцією розподілу одновимірної неперервної випадкової величини η називається функція

$$F_\eta(x) = P(\eta \le x).$$

Аналогічно вводиться поняття функції розподілу від векторної випадкової величини.

Густиною розподілу називається похідна від функції розподілу.

Умовною густиною розподілу $p(X_i / K_j)$ називається густина розподілу вектора X_i за умови, що він належить до класу K_j.

Для ґрунтовнішого ознайомлення з відповідним математичним апаратом читачеві рекомендується звернутися до будь-якого підручника з теорії ймовірностей.

Тепер можна описати байєсівське розв'язувальне правило.

Нехай ми маємо вектор Y. Для класифікації цього вектора слід для кожного класу розрахувати величини

$$p(K_i/Y) = \frac{P(K_i)p(Y/K_i)}{p(Y)},$$

які являють собою апостеріорні ймовірності належності даного вектора ознак для кожного класу. Слід обрати той клас, для якого ця ймовірність максимальна.

Далі у знаменнику фігурує величина $p(Y)$, яка є спільною для всіх класів і яку можна скоротити. Остаточно маємо таке розв'язувальне правило: **вектор Y слід віднести до того класу, для якого величина є максимальною.**

Байєсівські методи можна застосовувати і у випадку, коли необхідно враховувати ризики, пов'язані з неправильною класифікацією; відповідні формули можна знайти, наприклад, у [107, 260].

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте дискримінантні методи розпізнавання.
2. Що таке простір ознак?
3. Перелічіть відомі вам типи ознак.
4. Які ви знаєте міри близькості між об'єктами, якщо ознаки є кількісними?
5. Наведіть формулу евклідової відстані між векторами.
6. Що таке зважені евклідові відстані? В якому випадку їх доцільно використовувати?
7. Що таке матриця даних?
8. Охарактеризуйте гіпотезу компактності.
9. Опишіть типову схему розпізнавання в просторі ознак.
10. Дайте визначення роздільної функції.
11. Дайте визначення лінійної роздільності та лінійної роздільної функції.
12. Опишіть алгоритм персептрона для побудови лінійної роздільної функції.
13. Сформулюйте метод найближчого сусіда.
14. Чи вимагає метод найближчого сусіда попереднього вироблення розв'язувального правила?
15. Охарактеризуйте основні переваги і недоліки методу найближчого сусіда.
16. Опишіть стиснене правило найближчого сусіда. З чим пов'язана доцільність його застосування?
17. Охарактеризуйте байєсівські методи розпізнавання образів.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Чи для будь-якої предметної області можна використовувати розпізнавання в просторі ознак? Наведіть приклади.
2. Наведіть власні приклади дихотомічних, номінальних і порядкових ознак.

ЗАДАЧІ І ВПРАВИ

1. Наведіть приклад навчальної вибірки, для якої існує лінійна роздільна функція. Побудуйте цю функцію за допомогою алгоритму персептрона.
2. Дана навчальна вибірка, яка складається з трьох представників першого класу з координатами (0; 0); (0; 1) і (1; 0) та двох представників другого класу з координатами (1; 1) та (0.5; 0.5). Класифікуйте вектор з координатами (0.2; 1) за методом найближчого сусіда.
3. Наведіть приклад навчальної вибірки X і об'єкта y, що складається з п'яти представників першого класу і п'яти представників другого, таких, що всі представники другого класу знаходяться ближче до y, ніж будь-який представник першого класу, крім одного, але система, яка здійснює розпізнавання за методом найближчого сусіда, відносить y до першого класу.

Розділ 27

СИНТАКСИЧНІ МЕТОДИ РОЗПІЗНАВАННЯ

Точка, точка, запятая,
Минус — рожица кривая.
Ручки, ножки, огуречик —
Вот и вышел человечек.

З дитячої пісні

27.1. Загальна характеристика синтаксичних методів розпізнавання

Синтаксичні (інша назва — ***лінгвістичні***) методи розпізнавання [25, 46, 260, 262] беруть до уваги структуру об'єктів, які розпізнаються. Вважається, що опис кожного об'єкта складається з простих атомарних елементів, які складають алфавіт певної формальної мови. Ці прості елементи можуть по-різному поєднуватися між собою, при цьому утворюються фрази, або речення мови. Отже, кожному об'єкту відповідає деяке речення формальної мови.

Кожний клас описується певною формальною мовою. Інакше кажучи, всі об'єкти даного класу можна описати за допомогою деякої формальної граматики, що породжує дану мову. Тоді синтаксичне розпізнавання полягає в отриманні речення, що описує даний об'єкт, і у визначенні того, якою граматикою це речення породжується.

Типова схема системи, яка реалізує синтаксичне розпізнавання, складається з таких основних елементів:

- *блок попередньої обробки*, призначений для зменшення шумів і виконання інших необхідних операцій;
- *блок формування опису*, задача якого полягає у виокремленні атомарних елементів і зв'язків між ними, тобто у формуванні речення, що описує даний об'єкт;
- *блок синтаксичного аналізу*, який здійснює граматичний розбір речення і з'ясовує, якою граматикою воно породжується. Якщо таку граматику вдається знайти, система приймає рішення до віднесення об'єкта до відповідного класу.

Основною задачею навчання є формування граматик, які описують класи, що розпізнаються.

У принципі синтаксичні методи можна використовувати для розпізнавання об'єктів будь-якої природи. Але найчастіше вони застосовуються для розпізнавання сигналів і зображень.

Далі в цьому розділі буде наведено приклад формального опису простих зображень. Але спочатку необхідно дати основні відомості з теорії формальних граматик і мов.

27.2. Формальні граматики і мови

Формальною граматикою називається четвірка $G = < T, N, P, S >$, де T — скінченна множина *основних*, або *термінальних*, символів; N — скінченна множина *допоміжних*, або *нетермінальних*, символів. $T \cap N = \varnothing$ — множини основних і допоміжних символів не мають спільних елементів. Позначимо $V = T \cup N$; ε — пустий символ; Q^* — множина всіх ланцюжків, які утворюються символами з алфавіту Q; Q^+ — множина всіх непустих ланцюжків з Q; P — множина *правил підстановки*; кожне правило підстановки має вигляд $\alpha \to \beta$; тут α та β — деякі ланцюжки символів, при цьому ліва частина α містить хоча б один нетермінальний символ; дія правила підстановки полягає в тому, що в деякій фразі ланцюжок α замінюється на ланцюжок β. Наприклад, якщо маємо ланцюжок *XYZ* і правило підстановки $Y \to ab$, то ми можемо за допомогою цього правила підстановки отримати ланцюжок *XabZ*; S — *початковий символ*; $S \in N$, тобто початковий символ є нетермінальним.

***Мовою L (G)**, що породжується граматикою G, називається сукупність тих і тільки тих ланцюжків, які складаються з основних символів і можуть бути утворені з початкового символа шляхом скінченного числа застосувань правил підстановки даної такого G.*

Наведемо *приклад 27.1.* Побудуємо граматику, яка породжує речення, подібні до такого:

Маленька дівчина нагодувала цуцика.

Ця граматика може бути, наприклад, такою:

***T* = {маленька, дівчина, нагодувала, цуцика}**

***N* = {<Речення>, <Група підмета>, <Група присудка>, <Означення>, <Підмет>, <Присудок>, <Доповнення>}**

***S* = <Речення>**

<Речення> → <Група підмета> <Група присудка>

<Група підмета> → <Означення> <Підмет>

<Група присудка> → <Присудок> <Доповнення>

<Означення> → маленька

<Підмет> → дівчина

<Присудок> → нагодувала

<Доповнення> → цуцика

27.3. Класифікація граматик за Хомським

Одна з найвідоміших класифікацій формальних граматик була запропонована С. Хомським. Він поділив граматики на чотири класи. Тут ми позначатимемо термінальні символи малими літерами латинського алфавіту, нетермінальні — великими літерами, а ланцюжки символів — літерами грецького алфавіту.

Найширшим класом є ***граматики типу 0 (необмежені граматики)***. Правила підстановки таких граматик можуть бути будь-якими — на ці правила не накладається жодних обмежень.

Правила підстановки ***граматик безпосередньо складових*** мають вигляд:

$$\alpha_1 A \alpha_2 \to \alpha_1 \beta \alpha_2,$$

де $A \in N$, $\alpha_1, \alpha_2, \beta \in V^*$, $\beta \neq \varepsilon$.

Це правило можна також прочитати так: нетермінальний символ A можна замінити на непустий ланцюжок β у контексті α_1, α_2. Тому граматики безпосередньо складових інколи також називають *контекстно-залежними*.

Правила підстановки *контекстно-вільної* (або *безконтекстної*) граматики мають вигляд:

$$A \to \beta,$$

де $A \in N$, $\beta \in V^+$.

Це означає, що нетермінальний символ A можна замінити на непустий ланцюжок β незалежно від контексту, в якому він зустрічається.

Так, граматика, наведена в п. 27.2, являє собою типовий приклад контекстно-вільної граматики.

Правила підстановки ***автоматної*** (або ***регулярної***) граматики мають вигляд

$$A \to aB,$$

або

$$A \to b,$$

де $A, B \in N$, $b \in T$.

Наприклад, мова $L(G) = \{a^n b,\ n = 1, 2, \ldots\}$ породжується граматикою

$$S \to aA,$$
$$A \to aA,$$
$$A \to b.$$

Описані вище класи формальних граматик **строго включаються** один до одного. Це означає, що будь-яка автоматна граматика є безконтекстною (зворотне невірно), будь-яка безконтекстна граматика є граматикою безпосередньо складових і т. д.

Класифікація формальних мов відповідає класифікації формальних граматик. Слід тільки мати на увазі таке. Одна й та сама мова може бути породжена граматиками різних типів. Наприклад, граматика, наведена в п. 27.2, є безконтекстною, хоча існує й автоматна граматика, що породжує цю саму мову.

В цілому мова є регулярною, якщо існує хоча б одна регулярна граматика, що її породжує; мова є безконтекстною, якщо для неї існує хоча б одна безконтекстна граматика, і т. д.

Зверніть увагу ще на один класичний приклад. Мова $L_1 = \{0^n 1^m;\ n, m = 1, 2, \ldots\}$ є автоматною мовою, оскільки для неї існує проста регулярна граматика. Водночас мова $L_2 = \{0^n 1^n;\ n = 1, 2, \ldots\}$ не є автоматною.

27.4. Приклад опису зображень на основі формальних граматик

Розглянемо прості геометричні фігури, які являють собою чотири форми прямого кута. Побудуємо одну з можливих граматик для опису таких фігур [46].

Нехай маємо основний алфавіт $T = \{a, b, c, d\}$, символи a, b, c, d відповідають одиничним векторам:

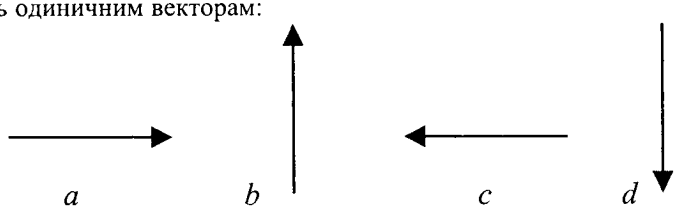


Рис. 27.1. Чотири атомарні елементи

Далі $N = \{S, A, B, C, D\}$; правила підстановки мають вигляд:

$S \to aA$,
$S \to bB$,
$S \to cC$,
$S \to dD$,
$A \to b$,
$B \to c$,
$C \to d$,
$D \to a$.

Тоді за допомогою розглянутої мови можна описати такі зображення:

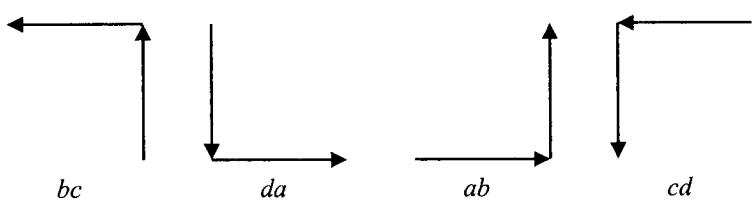


Рис. 27.2. Зображення, породжені граматикою

27.5. Основні методи граматичного розбору

Нагадаємо (п. 27.1), що після формування опису об'єкта, що розпізнається, необхідно провести граматичний розбір, метою якого є встановлення того, чи може утворена фраза бути породжена граматикою, що відповідає тому чи іншому класу. Інакше кажучи, необхідно встановити, чи належить дана фраза тій чи іншій формальній мові і які правила підстановки слід застосовувати для того, щоб отримати дану фразу з початкового символу.

Найбільшого поширення набули методи аналізу найпростіших типів граматик: автоматних і контекстно-вільних. Для останніх традиційним є застосування конструкції, яка дістала назву ***дерева граматичного розбору*** (або ***дерева виведення***).

Дерево розбору фрази $F = x_1 x_2 \ldots x_n$ на основі контекстно-вільної граматики $G = <T, N, P, S>$ повинно задовольняти таким вимогам:

1) кожна вершина має за мітку деякий символ з алфавіту $V = T \cup N$;

2) коренем дерева є початковий символ S;

3) листки дерева, якщо їх читати зліва направо, утворюють фразу F (порядок листків має велике значення!);

4) будь-яка нелистова вершина має за мітку нетермінальний символ, і навпаки;

5) якщо вершина A має синів $B_1\ B_2\ \ldots B_n$ (якщо відповідні мітки читати зліва направо), то множині P повинно належати правило підстановки $A \to B_1 B_2 \ldots B_n$.

Так, для прикладу з п. 27.1 *(Маленька дівчина нагодувала цуцика)* дерево розбору має вигляд:

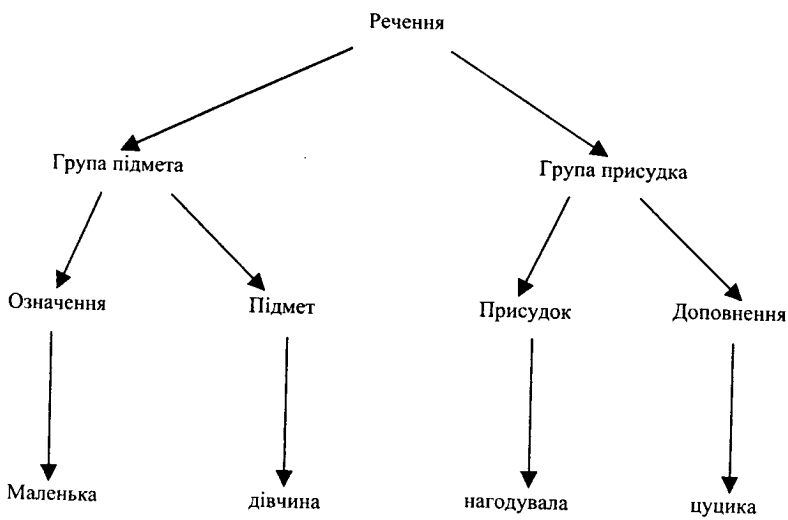


Рис. 27.3. Приклад граматичного розбору фрази

Існують дві основні стратегії граматичного розбору: згори донизу і знизу нагору.

Аналіз згори донизу починається з кореня дерева. Послідовно застосовуються підстановки, поки не буде отримана фраза, яка аналізується.

Аналіз знизу нагору починається з фрази, яка розбирається. Знайдені підланцюжки замінюються на ліві частини відповідних правил підстановки, поки не буде отриманий початковий символ.

Обидві стратегії граматичного розбору є перебірними процедурами. Перебір можна скоротити за рахунок використання ***апріорних тестів***, які дозволяють відкинути явно безперспективні правила підстановки. Деякі приклади таких тестів можна знайти в [262].

Існують методи аналізу, специфічні для автоматних граматик. Ці методи, які ґрунтуються на аналізі графу переходів відповідного скінченного автомата, будуть проілюстровані нижче.

27.6. Засоби опису складніших зображень

Традиційні граматики для утворення фраз використовують ланцюгову операцію конкатенації (зчеплення), і це обмежує їх застосування при описі зображень. Тому були запропоновані складніші граматики [262]: ***графові граматики***, ***мова опису зображень PDL***, ***веб-граматики***, ***плекс-граматики*** та ін. Розглянемо підхід, заснований на використанні плекс-граматик.

Плекс-граматики були розроблені Федером на основі ідеї Нарасімхана, яка полягає у формалізації опису з'єднань між елементами, що можуть мати довільну кількість точок під'єднання. Символ з N точками під'єднання називається *N-конфігурацією* (NAPE). Структури, які утворюються взаємозв'язаними NAPE, називаються *плекс-структурами*. Множини плекс-структур утворюють *плекс-мови*, граматики для опису плекс-мов називаються *плекс-граматиками*.

Плекс-граматика формально описується як шістка $<V_N, V_T, P, S, Q, q_0>$, де V_N — скінченна непуста множина допоміжних (нетермінальних) NAPE; $V_т$ — скінченна непуста множина основних (термінальних) NAPE; V_T та V_N не мають спільних елементів; P — скінченна множина правил підстановки; S — початковий символ, що називається початковою NAPE; при цьому $S \in V_N$; Q — скінченна множина ідентифікаторів, що відповідають точкам примикання; q_0 — спеціальний ідентифікатор, який називається пустим ідентифікатором.

Правило підстановки плекс-граматики у загальнішому вигляді записується як $\Psi\Gamma_{\psi\Delta\Psi} \rightarrow \omega\,\Gamma_{\omega\Delta\omega}$ де ψ — список замінюваних конфігурацій; ω — список конфігурацій заміщення; Γ_Ψ — список замінюваних точок з'єднань; Γ_ω — список точок з'єднань, які замінюють; $\Delta\Psi$ — список точок зчеплень замінюваної частини; $\Delta\omega$ — список точок зчеплень частини заміщення.

Списки конфігурацій замінюваної і замінюваних частин — це ланцюжки вигляду $\Psi = a_1a_2\ldots a_m$; $\omega = b_1b_2\ldots b_n$, де a_i та b_i — NAPE.

Списки точок з'єднання визначають, яким чином NAPE у списках конфігурацій зв'язані між собою. Вони діляться на поля. Поле — це список вигляду $(q_{i1}q_{i2}\ldots q_{ik})$, що визначає, які точки примикання яких NAPE беруть участь у даному з'єднанні. Одне поле відповідає одному з'єднанню. Входження q_i в деяке поле на j-му місці означає, що j-та компонента списку

конфігурацій бере участь у даному з'єднанні і з'єднується за допомогою i-ї точки примикання. Якщо j-та компонента не включена до з'єднання, в j-й позиції відповідного поля повинен стояти ідентифікатор q_0.

Списки точок зчеплень встановлюють відповідність між зовнішніми зв'язками частин заміщення. Кожній точці зчеплення, що стоїть на p-му місці у списку точок зчеплення замінюваної частини, повинна відповідати точка зчеплення частини, яка її заміщує.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте синтаксичні методи розпізнавання.
2. Які функції виконують блок формування опису та блок синтаксичного аналізу?
3. Дайте визначення формальної граматики.
4. Що таке формальна мова?
5. Опишіть класифікацію формальних граматик, що була дана С. Хомським.
6. Що таке контекстно-вільна граматика?
7. Що таке регулярна граматика?
8. Чи будь-яка автоматна граматика є безконтекстною? Покажіть, що зворотне невірно.
9. Дайте визначення дерева граматичного розбору.
10. Охарактеризуйте стратегії розбору "знизу нагору" та "згори донизу".
11. Що таке плекс-граматики? Для чого вони використовуються?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Чи можуть правила підстановок формальних граматик розглядатися як продукції деякої продукційної системи?
2. Як пов'язані стратегії граматичного розбору з бектрекінговими алгоритмами?

ЗАДАЧІ І ВПРАВИ

1. Наведіть власний приклад опису простих зображень на основі деякої граматики.
2. Дані дві мови:

   $L_1 = \{0^n1^m;\ n, m = 1, 2, \ldots\}$;

   $L_2 = \{0^n1^n;\ n = 1, 2, \ldots\}$.

   а) Наведіть приклад регулярної граматики, яка породжує мову L_1.

   б) Наведіть приклад безконтекстної, але не регулярної граматики, яка породжує мову L_1.

   в) Наведіть приклад безконтекстної граматики, яка породжує мову L_2.
3. Наведіть приклад плекс-граматики, за допомогою якої можна описати кілька довільних літер українського алфавіту.

Розділ 28

ОСНОВНІ МЕТОДИ ПОПЕРЕДНЬОЇ ОБРОБКИ СИГНАЛІВ І ЗОБРАЖЕНЬ

Подделывал так чисто сторублёвки,
Что запросто на рынке их сбывал...
С. Михалков

28.1. Суть попередньої обробки сигналів і зображень

Як уже зазначалося, обробка експериментальних сигналів і зображень має велике значення для теорії і практики розпізнавання. Нагадаємо, що експериментальні сигнали описуються функцією однієї змінної (найчастіше — функцією, яка задає зміну деякої фізичної величини з часом). Зображення ж описуються функцією двох змінних, яка задає значення яскравості в різних точках зображення.

На етапі попередньої обробки вирішується ряд важливих задач, насамперед таких:

- отримання первинних ознак;
- зменшення шумів та підвищення якості зображення (наприклад, підвищення чіткості контурів фігур);
- сегментація (поділ зображення на зони, які будуть оброблятися).

Існує величезна кількість задач, пов'язаних з обробкою сигналів і зображень. Деякі з цих задач, а також основні методи їх вирішення описані, наприклад, у [25, 46, 209].

28.2. Отримання первинних ознак на основі дискретизації

Найтиповішим і найчастіше вживаним способом отримання первинних описів об'єктів, які задаються неперервними величинами, є ***дискретизація*** цих величин (інші назви — *оцифровка*, *цифрування*). Дискретизація є аналогово-цифровим перетворенням і полягає в тому, що аналогова величина реєструється у вигляді набору своїх значень у дискретних точках.

Нехай неперервна функція $f(t)$ спостерігається на проміжку $[a, b]$. Зафіксуємо n дискретних точок, в яких вимірюватимуться значення функції: $t_1 = a < t_2 < \ldots < t_{n-1} < t_n = b$. Ці точки часто називаються *вузлами*. Тоді дискретизація функції $f(t)$ полягає у представленні її вектором $y = (y_1, \ldots, y_n)$, де $y_j = f(t_j)$ (тобто кожна координата вектора являє собою значення функції в черговому вузлі).

Дискретні значення $y_j = f(t_j)$ часто називають первинними ознаками сигналу. Найчастіше застосовується *рівномірна дискретизація*, за якої відстань між будь-якими сусідніми вузлами є однаковою. У такому разі величина $h = t_{j+1} - t_j$, однакова для всіх j, називається *кроком дискретизації*.

Легко переконатися в тому, що крок рівномірної дискретизації може бути обчислений за формулою

$$h = \frac{b-a}{n-1}.$$

Величина $q = 1/h$ називається *частотою дискретизації*, що характеризує кількість дискретних вузлів на одиничному інтервалі.

Приклад 28.1. Отримаємо набір дискретних значень функції $f(t) = 2t + 1$ на проміжку [0, 1] з кроком 0.2.

Маємо $n = 6$; $t_1 = a = 0$; $t_2 = 0.2$; $t_3 = 0.4$; $t_4 = 0.6$; $t_5 = 0.8$; $t_6 = b = 1$. Тоді вектор дискретних значень $y = (1; 1.4; 1.8; 2.2; 2.6; 3)$.

Звичайно, дискретизовану функцію можна задавати і в табличному вигляді:

t	0	0.2	0.4	0.6	0.8	1
$f(t)$	1	1.4	1.8	2.2	2.6	3

Частоту дискретизації не можна вибирати довільно. Надто висока частота дискретизації (і, відповідно, надто малий крок дискретизації) приводить до завеликого обсягу інформації. Це, в свою чергу, спричиняє надмірні витрати на зберігання і обробку такої інформації. У зв'язку з цим виникає проблема стиснення інформації, яка в даному контексті означає зберігання оцифрованої інформації в економнішому вигляді.

З іншого боку, зменшення частоти дискретизації приводить, як правило, до зменшення точності. Фундаментальна ***теорема Котельникова*** (аналогічний результат пізніше був отриманий Шенноном) встановлює, якою повинна бути мінімальна частота дискретизації, за якої інформація не втрачається, тобто за якої неперервна функція ще може бути точно відновлена за дискретними даними [208].

Аналогічно здійснюється дискретизація зображень. На зображення накладається двовимірна сітка, і тоді зображення можна закодувати матрицею, елементи якої дорівнюють сумарній яскравості зображення в окремому квадратику сітки. Часто застосовується і простіший спосіб кодування, за якого елемент матриці дорівнює 1, якщо у відповідному квадратику є фрагмент зображення, і 0 — якщо немає.

Приклад 28.2. Нехай сканується зображення, яке являє собою літеру А. Закодуємо це зображення за допомогою сітки розміром 8 × 8.

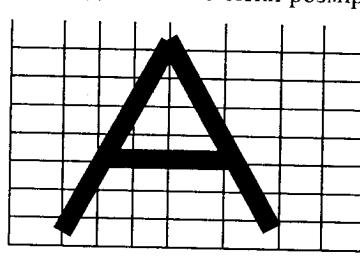


Рис. 28.1. Приклад дискретизації зображення

Отримаємо матрицю

$$\begin{pmatrix} 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \end{pmatrix}$$

Слід звернути увагу на обставини, які ускладнюють обробку подібних зображень:

1. Форма відсканованої літери дещо спотворюється; це стає особливо помітним при великому розмірі клітинок сітки (при малому кроці дискретизації).
2. Сканер працює не бездоганно, і в результаті на відсканованому зображенні можуть з'являтися зайві лінії, яких не було на оригіналі, і навпаки — зникати ті, які були. В результаті, наприклад, літера **О** може бути сприйнята як літера **С**.
3. Результат кодування є дуже чутливим до позиціонування тексту відносно сканера. Якщо літеру на рисунку трошки зсунути вбік, зображення літери потрапить в інші квадрати, і матриця істотно зміниться.
4. Одна й та сама літера може бути написана дуже по-різному.

Усі ці обставини зумовлюють добре відомі факти: якість розпізнавання друкованого тексту істотно перевищує якість розпізнавання тексту рукописного; якість розпізнавання знижується при недостатній якості друку, а також при низькій роздільній здатності сканера; багато реальних систем розпізнавання орієнтовані на певний стиль написання літер, і користувач повинен попередньо засвоїти цей стиль (зокрема, це стосується ряду сучасних персональних помічників).

Крім того, методи розпізнавання в просторі ознак є дуже чутливими до перелічених вище факторів, і тому для розпізнавання літер насамперед використовуються структурні, в першу чергу синтаксичні, методи.

28.3. Ланцюговий код Фрімена

Для опису ліній і контурів зображень широко застосовуються ***ланцюгові коди***, основна ідея яких полягає в такому. Відрізки ліній апроксимуються векторами з певного фіксованого набору, кожний з яких, у свою чергу, кодується символом деякого алфавіту.

Найпростішим і вживанішим є ***код Фрімена***, за якого лінії кодуються за допомогою восьми можливих векторів, пронумерованих числами від 0 до 7:

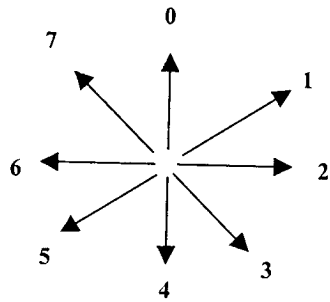


Рис. 28.2. Ланцюговий код Фрімена

Типова схема кодування полягає в такому. Вузли сітки, що є найближчими до контуру, з'єднуються прямими лініями (векторами), які, в свою чергу, кодуються відповідними числовими значеннями.

28.4. Параметризація неперервних функцій на основі лінійних ортогональних перетворень

Потужним засобом отримання вторинних ознак сигналів і зображень є використання ***ортогональних перетворень***. Надалі розглядатимемо лише одновимірні сигнали.

З курсу математичного аналізу добре відомо, що неперервна функція $\eta(t)$, яка спостерігається на відрізку $[a, b]$, може бути розкладена в ряд:

$$\eta(t) = \sum_{k=1}^{\infty} c_k \varphi_k(t). \qquad (28.1)$$

При цьому координатні функції $\{\varphi_k(t)\}$ можуть бути вибрані *ортонормованими*. Це означає, що скалярний добуток

$$\int_a^b \varphi_k(t)\,\varphi_l(t)\,dt$$

дорівнює 1, якщо $k = l$ і 0, якщо $k \neq l$.

Якщо координатні функції ортонормовані, коефіцієнт розкладу (28.1) обчислюється за формулою:

$$c_k = \int_a^b \varphi_k(t)\eta(t)dt, \quad k = \overline{1,\infty}. \qquad (28.2)$$

Формула (28.2) і являє собою формулу *ортогонального перетворення*.

Для функцій, які можуть бути фізично реалізовані, коефіцієнти c_k прямують до нуля, і, більш того, ряд (28.1) збігається. На практиці його можна апроксимувати скінченним рядом

$$\eta(t) = \sum_{k=1}^{r} c_k \varphi_k(t) \qquad (28.3)$$

з певною похибкою.

Тоді спектральні коефіцієнти c_k можна розглядати як ознаки функції $\eta(t)$, а вектор $(c_1, \ldots, c_r)$ — як уявлення цієї функції у відповідному просторі ознак.

Існує чимало літератури, що стосується розпізнавання в просторі спектральних коефіцієнтів, наприклад [207].

Найбільш відомим ортогональним перетворенням є *перетворення Фур'є*, формулу якого можна записати у вигляді:

$$c_k = \frac{1}{2\pi}\int_{-\pi}^{\pi} \eta(t) e^{-ikt}\, dt, \quad k = \overline{1, \infty}. \tag{28.4}$$

Існує значна кількість перетворень, тісно пов'язаних з перетворенням Фур'є, в першу чергу синусне та косинусне. Так, якщо врахувати, що

$$e^{-ix} = \cos x - i \cdot \sin x,$$

ряд Фур'є можна переписати таким чином.

Нехай неперервна функція $f(x)$ задана на відрізку $[-\pi, \pi]$. До загальнішого випадку ми можемо перейти за допомогою заміни змінних. Ряд Фур'є такої функції записується у вигляді:

$$f(x) = \frac{a_0}{2} + \sum_{n=1}^{\infty} \left(a_n \cos nx + b_n \sin nx\right),$$

де коефіцієнти розкладу обчислюються за формулами:

$$a_0 = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x)\, dx,$$

$$a_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x) \cos nx\, dx,$$

$$b_n = \frac{1}{\pi}\int_{-\pi}^{\pi} f(x) \sin nx\, dx.$$

Безпосередня реалізація формули (28.2) та її дискретних апроксимацій пов'язана зі значними часовими затратами. Але було помічено, що через періодичність синусів і косинусів доданки, які входять до розрахункових формул, часто повторюються; крім того, виявилося, що розрахунковий алгоритм можна записати в рекурсивній формі. Ці міркування стали основою швидких алгоритмів реалізації перетворення Фур'є, які широко застосовуються на практиці. Ці алгоритми відомі під загальною назвою *швидке перетворення Фур'є*. Детальніше з ними можна ознайомитись в [117, 208].

28.5. Інтегральне перетворення Карунена—Лоева

Інтегральне перетворення Карунена—Лоева є в певному розумінні оптимальним ортогональним перетворенням. Його оптимальність полягає в тому, що воно мінімізує похибку апроксимації ряду (28.1) при фіксованому числі членів усіченого ряду r, або мінімізує r у межах заданої похибки. В цьому розумінні можна сказати, що перетворення Карунена—Лоева забезпечує оптимальне стиснення даних [263].

Координатні функції перетворення Карунена—Лоева обчислюються як власні функції рівняння

$$\int_a^b R(t,s)\varphi_k(s)\,ds = \sigma_k \varphi_k(t), \tag{28.5}$$

що відповідають власним функціям σ_k, впорядкованим за спаданням.

Тут $R(t, s)$ — коваріаційна функція породжуючого випадкового процесу. Вона в цілому є невідомою, але може бути оцінена за визначеним набором реалізацій цього процесу V. Як правило, для такого оцінювання використовується формула

$$\hat{R}^{(V)}(t,s) = \sum \alpha_i \xi_i(t)\,\xi_i(s), \tag{28.6}$$

де сума береться по всіх функціях $\xi_i(t)$, що входять до набору V; α_i — задані коефіцієнти, сума яких дорівнює 1. Множину V називатимемо базовою вибіркою, позначимо через q число її елементів.

Основним недоліком перетворення Карунена—Лоева є відсутність для нього швидких перетворень. Незважаючи на це, роль перетворення Карунена—Лоева зростає в таких випадках:

- у задачах, в яких швидкість обробки не має суттєвого значення, а на перший план виходить саме міра стиску інформації;
- при аналізі нестаціонарних процесів, оскільки їх спектр Фур'є залежить від двох змінних і практичне використання такого спектра ускладнюється;
- у ряді задач, в яких першочергове значення має некорельованість спектральних коефіцієнтів, яка в загальному випадку забезпечується лише перетворенням Карунена—Лоева. Це, наприклад, задача відновлення сигналів і зображень, спотворених неідеальною вимірювальною апаратурою [44, 45, 204, 205, 206, 250].

Алгоритми комп'ютерної реалізації власне інтегрального перетворення Карунена—Лоева сьогодні розвинуті недостатньо. Їх ефективність суттєво залежить від способу апроксимації інтегралів. Одну з можливих обчислювальних схем було запропоновано в [204, 205]. Будемо називати цю схему *базовою квадратурною схемою* інтегрального перетворення Карунена—Лоева. При розробці цієї схеми було враховано необхідність отримання простих векторно-матричних формул, зручних для програмування.

Базова квадратурна схема вимагає дискретизації реалізацій випадкового процесу в точках $t_j, j = 1, \ldots, n$.

Нехай

$X = (x_{ij}, i = 1, \ldots, q; j = 1, \ldots, n)$; $x_{ij} = \xi_i(t_j)$ — матриця розміром $q \times n$, яка задає базову вибірку V;

$y = (y_j, j = 1, \ldots, n)$; $y_j = \eta(t_j)$ — n-вектор, який задає довільну функцію з деякої множини U;

$\Phi = (\varphi_{jk}, j = 1, \ldots, n; k = 1, \ldots, r)$; $\varphi_{jk} = \varphi_k(t_j)$;

$c = (c_k, k = 1, \ldots, r)$ — вектор спектральних коефіцієнтів;

$\Lambda = \text{diag}\,(\sigma_k)$;

$A = \text{diag}\,(\alpha_i)$; можна взяти $\alpha_i = 1/q$ для всіх i.

$B = \text{diag}\,(\beta_k)$; β_k — коефіцієнти квадратурної формули, що використовуються для апроксимації інтегралів.

До складу базової квадратурної схеми входять два алгоритми: ***прямий квадратурний алгоритм***, який доцільно застосовувати при $n \le q$, та ***двоїстий квадратурний алгоритм*** для випадку $n > q$. Обчислювальні формули наведено в табл. 28.1.

Таблиця 28.1

№ кроку	Крок	Прямий квадратурний алгоритм	Двоїстий квадратурний алгоритм
1	Отримання опорної матриці K	$K^{(\text{пр})} = B^{1/2} X^T A X B^{1/2}$	$K^{(\text{дв})} = A^{1/2} X B X^T A^{1/2}$
2	Вирішення алгебраїчної задачі на власні значення для опорної матриці	$K^{(\text{пр})} \Psi^{(\text{пр})} = \Psi^{(\text{пр})} \Lambda$	$K^{(\text{дв})} \Psi^{(\text{дв})} = \Psi^{(\text{дв})} \Lambda$
3	Отримання матриці дискретизованих власних функцій Ф	$\Phi = B^{-1/2} \Psi^{(\text{пр})}$	$\Phi = X^T A^{1/2} \Psi^{(\text{дв})} \Lambda^{-1/2}$
4	Отримання матриці перетворення J	$J = B\Phi$	$J = B\Phi$
5	Перетворення базової вибірки	$W = XJ$	$W = XJ = A^{-1/2} \Psi^{(\text{дв})} \Lambda^{1/2}$
6	Перетворення довільної функції	$c = yJ$	$c = yJ$

Слід відмітити, що на практиці замість інтегрального перетворення Карунена—Лоева використовується його ***дискретний варіант***, який, по суті, є відомим у математичній статистиці методом головних компонент [5]. Можна показати, що цей дискретний варіант утворюється з прямого квадратурного алгоритму, якщо A та B є одиничними матрицями.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Охарактеризуйте основні задачі, які вирішуються на етапі попередньої обробки сигналів і зображень.
2. У чому полягає процес дискретизації?
3. Що таке рівномірна дискретизація?
4. Що таке ланцюговий код Фрімена?
5. Що таке ортонормована система функцій?
6. Що таке ортогональне перетворення?
7. Яким чином можна отримати ознаки сигналу на основі ортогональних перетворень?

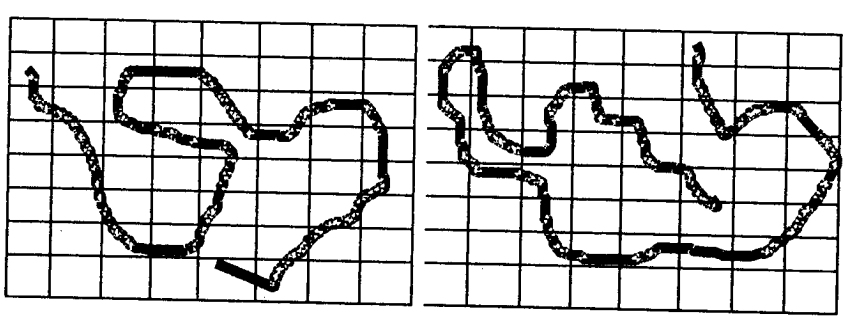
8. Наведіть формули перетворення Фур'є.
9. Що таке швидке перетворення Фур'є?
10. Охарактеризуйте інтегральне перетворення Карунена—Лоева. У чому полягають його переваги та недоліки?
11. Опишіть базову квадратурну схему інтегрального перетворення Карунена—Лоева.
12. В якому випадку вигідніше застосовувати прямий квадратурний алгоритм, а в якому двоїстий?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Охарактеризуйте відомі Вам методи стиснення даних, які сьогодні застосовуються на практиці. У чому полягають їх переваги і недоліки?
2. Чому для розпізнавання літер і мови доцільніше використовувати синтаксичні методи?
3. Чому рукописні літери важче розпізнавати, ніж друковані?

ЗАДАЧІ І ВПРАВИ

1. Написати програму, яка виділяє ознаки неперервних функцій у вигляді:
   а) набору дискретних значень;
   б) диференціального коду;
   в) ланцюгового коду Фрімена.
2. Отримайте первинні ознаки шляхом дискретизації функції $f(t)$, яка вимірюється на інтервалі [1, 2], якщо крок дискретизації (відстань між сусідніми замірами) становить 0.2:
   а) $f(t) = 2t - 1$;
   б) $f(t) = 3 - t$.
3. Закодуйте графік функції
   $y = 2x^2 + 1$
   на проміжку [0, 1] ланцюговим кодом Фрімена. Взяти по обох координатах однаковий крок, який дорівнює 0.1.
4. Подайте за допомогою ланцюгового коду Фрімена фігуру, зображену на рисунку:



а *б*

ЧАСТИНА VII НЕЙРОННІ МЕРЕЖІ

Розділ 29

ЗАГАЛЬНА ХАРАКТЕРИСТИКА КОНЕКЦІОНІСТСЬКОГО ПІДХОДУ ДО ПОБУДОВИ СИСТЕМ ШТУЧНОГО ІНТЕЛЕКТУ

Гуртом і батька легше бити.
Народне прислів'я

29.1. Конекціоністський підхід як спроба моделювання людського мозку

Конекціоністський підхід до побудови систем штучного інтелекту розвинувся на противагу символьному, що є характерним для сучасних моделей знань. В основі конекціоністського підходу лежить спроба безпосереднього моделювання розумової діяльності людського мозку. Відомо, що мозок людини складається з величезної кількості нервових клітин (нейронів), що взаємодіють, тобто являє собою природну ***нейронну мережу*** [9, 155, 255]. Ці "обчислювальні елементи" мозку функціонують набагато повільніше, ніж обчислювальні елементи комп'ютерних систем. Але, незважаючи на це, біологічні нейронні мережі вирішують багато задач (особливо нечислових) у тисячі разів швидше, ніж електронний процесор.

Причинами цього є:

1. Нейрони людського мозку функціонують паралельно і постійно обмінюються інформацією.
2. У людському мозку пам'ять не локалізована в одному місці (як у традиційній обчислювальній машині), а є розподіленою. В біологічних системах пам'ять реалізується підсиленням або послабленням зв'язків між нейронами, а не зберіганням двійкових символів.
3. Біологічні мережі реагують не на всі, а тільки на визначені зовнішні подразнення. Кожний нейрон виступає як елемент прийняття рішення і як елемент зберігання інформації. Перевага такої структури — "життєздатність" (вихід з ладу кількох нейронів не призводить до значної зміни даних, що зберігаються, або ж до руйнування всієї системи).

4. Можливість адресації за вмістом (асоціативної пам'яті) є ще однією важливою характеристикою систем з розподіленою пам'яттю (кожний елемент відшукується за його вмістом, а не зберігається в комірці пам'яті з визначеним номером).

Мислення людини є здебільшого ***асоціативним***, і в основу роботи систем, які моделюють мислення людини, може бути покладено ***принцип асоціації***. Під асоціацією розуміється деяка міра схожості або наявність певних спільних рис між тими чи іншими явищами і поняттями. Асоціативні зв'язки пронизують все мислення людини. При цьому процес породження асоціацій і висновків за асоціацією носить підсвідомий і здебільшого випадковий і неконтрольований характер. Існує думка, що **процеси мислення носять хвильовий характер і є не що інше, як розповсюдження певного збудження, як деяка ланцюгова реакція**. Недоліком такого типу мислення є те, що внаслідок нефіксованої організації такі системи можуть приходити до невірних висновків, плутати різні об'єкти тощо. Людина долає такі проблеми шляхом навчання, звикання до конкретних ситуацій, контролю над своєю поведінкою. На підсвідомому ж рівні процес породження асоціацій, очевидно, також скеровується деяким механізмом, який спрямовує його в бажаному напрямі. Подібних рис мають набути і системи штучного інтелекту.

Важливою особливістю нейроноподібних систем є їх ***здатність до ефективного навчання***, навіть в умовах, якщо не вдається на вербальному рівні, тобто в словесній формі, сформулювати мету системи та виробити правила її поведінки. Навчання йде в основному на підсвідомому рівні.

Усі ці особливості зумовили привабливість конекціоністського підходу до побудови систем штучного інтелекту. Такі системи повинні функціонувати за принципами, подібними до тих, за якими функціонує мозок людини, хоча самі ці принципи залишаються не до кінця зрозумілими. Безумовно, це не виключає використання в таких системах можливостей, характерних для символьного підходу, зокрема оперування поняттями, дедуктивне логічне виведення тощо.

Як і вказує її назва, **штучна нейронна мережа** (ШНМ) — це система, яка складається з деякої кількості пристроїв, так званих штучних нейронів, з'єднаних між собою. Ці пристрої називаються штучними нейронами завдяки певній подібності до нейронів людського мозку. Однак вони не призначені для точного копіювання всіх електрохімічних процесів, які відбуваються у мозку людини. Мозок лише виступає як модель для побудови ШНМ. У деяких випадках ми називатимемо штучні нейрони (ШН) просто нейронами, якщо при цьому не виникатиме загроза переплутати штучні та біологічні нейрони.

Існує багато способів з'єднання між собою ШН, отже, можна виокремити різні типи ШНМ. В основному вони різняться за такими двома параметрами, як операційні характеристики нейронів та різні конфігурації утворення мережі.

Оскільки поведінка мережі істотно залежить від поведінки окремих нейронів, почнемо з того, що розглянемо основні ідеї побудови штучного нейрона. На цьому етапі розглядатимемо ШН як деякий абстрактний пристрій, що працює за такою схемою. Він одержує вхідні сигнали з одного або кількох входів. Зокрема, можна вважати, що ШН отримує на своїх входах кілька чисел. Тому вхідний сигнал ШН можна також сприймати як деякий кортеж чисел. Традиційно ШН перетворює значення всіх вхідних сигналів у мережевий вхід. Існує багато способів перетворення кортежу чисел в єдине число, але звичайно ШН формує мережевий вхід, підраховуючи зважену суму входів за певними коефіцієнтами.

ШН відповідає на різні значення мережевого входу різними значеннями мережевого виходу. Зв'язок між мережевим входом і мережевим виходом зветься *активаційною функцією нейрона*. Існує багато можливих функцій або зв'язків, що визначають, яке значення мережевого виходу відповідатиме даному значенню мережевого входу.

У більшості випадків передатна функція порівнює значення мережевого входу з деяким порогом. При значенні мережевого входу, рівному або більшому за поріг, нейрон виробляє вихід 1, а в іншому разі не виробляє жодного виходу. Завдяки різкій зміні значення при досягненні порогу така активаційна функція зветься жорсткою.

Історично поняття ШН із жорсткою пороговою активаційною функцією було вперше запропоновано двома вченими Уорреном Мак-Каллоком та Уотером Піттсом на початку 40-х років. На їх честь нейрони подібного типу названі нейронами Мак-Каллока—Піттса, або МР-нейронами.

При розробці штучних нейронних мереж виникає ряд очевидних проблем.

По-перше, **пряме моделювання мозку вимагає величезної кількості нейронів**. Так, модель мурашки потребує використання близько 20 000 нейронів, людини — 100 млрд нейронів, і реалізація таких моделей є практично неможливою.

По-друге, **здатність мозку як високоорганізованої інтелектуальної системи до підсвідомого мислення і до навчання була вироблена протягом сотень мільйонів років еволюції**. Механізми ж самонавчання і самоорганізації штучних нейронних мереж, зв'язки яких можуть мати випадковий характер, сьогодні дуже недосконалі, і їх можна застосовувати лише для вузького кола порівняно простих задач. Проте ці методики продовжують інтенсивно розвиватися.

29.2. Основні сфери застосування

Існують і продовжують створюватися комп'ютерні системи, побудовані на основі штучних нейронних мереж. Такі системи дістали назву ***нейрокомп'ютерів***. Нейрокомп'ютер — це програмно-технічна система (здебільшого спеціалізована), яка реалізує деяку формальну модель природної мережі нейронів. Значного розвитку набули програмні пакети, які

моделюють роботу нейронних мереж на універсальних комп'ютерах. Так, у [149] описуються такі пакети, як BrainMaker, OWL, "The AI Trilogy", Neural10, NeuroPro та ін. Серед основних сфер застосування таких систем можна відмітити розпізнавання образів, обробку сигналів і зображень, прогнозування природних і соціально-економічних явищ і т. п.

Багато дослідників вважають, що майбутнє належить комп'ютерам, які базуються на аналізі зв'язків, а не на обробці символів, тобто конекціоністський підхід до створення систем штучного інтелекту стане домінуючим. М. Мінський говорив, що якщо комп'ютер повинен діяти подібно до мозку, тоді і його конструкція повинна бути також подібна до мозку.

Дуже важливою є така головна відмінність нейрокомп'ютера від звичайних комп'ютерів обчислювального типу. У традиційному обчислювальному пристрої для того, щоб при даному вході дійти до бажаного виходу, йому потрібна точна програма, яка складається з конкретних інструкцій виконання. Нейрокомп'ютери (ШНМ) мають можливості по досягненню бажаного результату, що робить їх істотно відмінними. Вони можуть навчатися на прикладах. Якщо існує навчальна вибірка, тобто певна множина пар даних, перша компонента яких є деяким вхідним кортежем, а друга — бажаним виходом на цьому кортежі, ШНМ може навчитися змінювати власні ваги таким чином, щоб з кожним входом асоціювався вірний вихід. Така можливість є дуже важливою, оскільки існує багато проблем, для яких відомий вірний результат, але важко визначити точну процедуру, або список правил, для пошуку результату. Якщо в таких випадках навчання на прикладах допомагає ШНМ побудувати власні приховані правила у термінах використання доречних ваг, вона має безперечні переваги.

Пари даних (вхід разом з бажаним виходом), які використовуються для навчання нейронної мережі, називаються навчальною вибіркою даних. Проілюструємо його на простому прикладі [39] з єдиним ШН. Спробуємо навчити його виконувати логічну функцію ТА. Тобто, ми хочемо, щоб він налаштував свої ваги таким чином, що на вхідних кортежах (0, 0), (0, 1) та (1, 0) він казав би "ні", або ХИБНІСТЬ, та видавав 0, а на вхідному кортежі (1, 1) казав би "так", або ІСТИНА, і видавав би 1. Звичайно, ми повинні надати ШН якесь навчальне правило, що вказує, як змінювати ваги, якщо з початковими вагами ШН виробляє невірні відповіді. Припустимо, що ми запускаємо ШН з початковими вагами 0.5 та 0.6 для двох входів, та з порогом 0.3.

Його мережевий вхід дорівнює:

$$0.5 \times X_1 + 0.6 \times X_2 - 0.3 \times X_0,$$

де X_0 — постійний вхід, завжди рівний 1. За допомогою цього входу ми моделюємо поріг T нейрона Мак-Каллока—Піттса. Подібний фіктивний вхід дає змогу вважати поріг незмінним і рівним нулю. ШН збуджується або не збуджується залежно від того, чи перевищує мережевий вхід поріг або дорівнює йому, чи ні.

Ми бачимо, що мережевий вхід для чотирьох можливих вхідних кортежів дорівнює:

0	0	−0.3
1	0	0.2
0	1	0.3
1	1	0.8

Отже, нейрон видає 1 для вхідних кортежів (1, 1), (0, 1) та (1, 0), у той час як ми бажаємо, щоб він видавав 1 лише на кортежі (1, 1). Навчальне правило, яке звичайно використовується, є інтуїтивно правильним і вимагає таких регулювань:

1. Подати на вхід ШН один з навчальних вхідних кортежів та дозволити йому виробити свій вихід за допомогою поточних ваг.
2. Якщо вихід ШН дорівнює бажаному, повернутися до кроку 1 і взяти наступний навчальний вхідний кортеж.
3. Якщо ШН видає 0, а бажаний вихід дорівнює 1, збільшити ті ваги, які асоційовані з активними входами, тобто з тими, які дорівнюють 1. Збільшення ваг повинно дорівнювати деякій частині від помилки ШН. Ми бачимо, що помилка дорівнює 1 (ШН видає 0 замість 1). Тож збільшимо ваги активних входів, скажімо, на 0.1. Має сенс збільшувати ваги саме активних входів, бо ми хочемо збільшити вихід ШН. Також зрозуміло, що немає сенсу змінювати ваги входів, які не є активними (вони дорівнюють 0), оскільки вони не роблять жодного внеску у вихід нейрону.
4. З тих самих міркувань, якщо ШН видає 1 при бажаному виході 0, ваги активних входів зменшуються.
5. Повторюємо налаштовуючий процес по черзі для кожної пари з навчальної вибірки.

Неочевидно, що якщо робити таке налаштовування циклічно для кожної пари навчальної вибірки, це призведе до ситуації, в якій правильний бажаний вихід вироблятиметься для кожного вхідного кортежу (тобто, ШНМ запам'ятає правильні відповіді). Френк Розенблатт, видатний піонер у сфері ШНМ, довів [237], що якщо існує рішення проблеми, то відповідні ваги будуть знайдені за скінченну кількість налаштувань. Його досягнення вважається одним з найвидатніших теоретичних результатів у цій області.

Основну відмінність між традиційним комп'ютером і нейрокомп'ютером можна продемонструвати і на такому класичному прикладі. Нехай нам постійно потрібно вирішувати задачу типу додавання двох чисел. Скільки б разів ми не запускали програму її обчислення на комп'ютері, вона щоразу потребує проведення певного обчислення процесором. У разі ж проведення аналогічних обчислень на нейрокомп'ютері через певну кількість таких обчислень виробиться асоціативний зв'язок між мережевими входом (двома числами) і виходом (сумою двох чисел). Тому при потребі аналогічних обчислень у майбутньому проміжні обчислення не проводитимуться.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Дайте характеристику конекціоністського підходу до побудови систем штучного інтелекту.
2. Що являє собою природний нейрон?
3. Чому безпосереднє моделювання мозкової діяльності людини і тварини є неможливим?
4. Охарактеризуйте основні переваги і недоліки штучних нейронних мереж.
5. Дайте характеристику асоціативного мислення.
6. У чому полягає основний недолік асоціативного мислення?
7. Перелічіть відомі вам програмні засоби для моделювання нейронних мереж.
8. Наведіть відомі вам застосування нейромережевих технологій.

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. В яких практичних задачах, на вашу думку, доцільно використовувати нейрокомп'ютерні технології?
2. Як, на вашу думку, можна поєднати переваги символьного і конекціоністського підходів?

Розділ 30
МОДЕЛЬНІ НЕЙРОНИ

Нажми на кнопку —
получишь результат...
З пісні

30.1. Модельні нейрони як порогові елементи

Розглянемо формальніший опис досліджуваних раніше на абстрактному рівні понять нейрона та нейронної мережі. У своїй піонерській роботі [316], опублікованій у 1943 р., Мак-Каллок і Піттс розглянули просту модель нейрона (модельний нейрон) як деякий пороговий елемент з n входами та одним виходом (рис. 30.1). Кожному входу приписується вага w_i, яка може бути як додатною, так і від'ємною.

Нехай на вхід модельного нейрону подаються вхідні сигнали x_i, $i = 1, \ldots n$. Обчислюється зважена сума входів:

$$S = \sum_{i=1}^{n} w_i x_i ,$$

і вихідний сигнал нейрона y обчислюється як деяка задана функція від цієї суми:

$$y = f(S).$$

Функція $f(S)$ називається ***активаційною***. У найпростішому випадку вона є пороговою функцією, тобто дорівнює 0, якщо $S \le p$ та 1, якщо $S > p$. Тут p — деякий фіксований поріг.

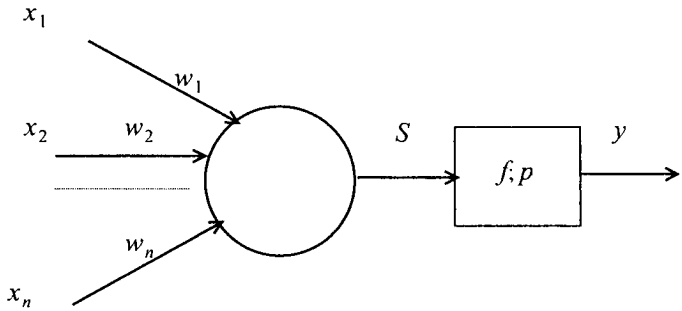


Рис. 30.1. Простий модельний нейрон

Мак-Каллок і Піттс розглядали саме таку просту активаційну функцію.

30.2. Повнота системи модельних нейронів

Мак-Каллок і Піттс довели, що обчислення будь-якої обчислюваної за Тьюрингом функції можна реалізувати за допомогою спеціально організованої мережі модельних нейронів. Цей фундаментальний результат стає зрозумілішим, якщо розглянути порогові елементи, які реалізують три базові логічні функції: кон'юнкцію, диз'юнкцію та заперечення.

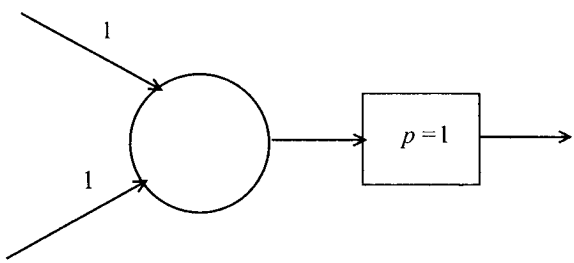


Рис. 30.2. Пороговий елемент, що реалізує кон'юнкцію

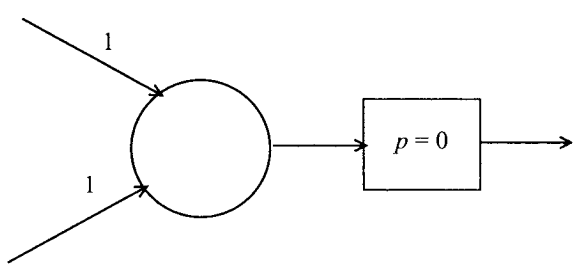


Рис. 30.3. Пороговий елемент, що реалізує диз'юнкцію

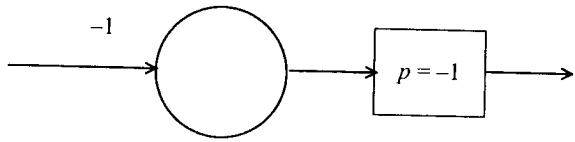


Рис. 30.4. Пороговий елемент, що реалізує заперечення

Результат Мак-Каллока і Піттса стверджує **принципову можливість** побудови обчислювальної системи на основі порогових елементів. Але для більшості реальних практичних задач характерним є те, що коефіцієнти w_i майже ніколи не бувають відомими заздалегідь. Проте їх часто вдається отримати шляхом навчання. Спочатку вони розставляються довільно, **і процес навчання нейронної мережі полягає в зміні цих коефіцієнтів за тими чи іншими правилами**. Основною проблемою при цьому є пошук ефективних алгоритмів такого навчання.

30.3. Процедура Уїдроу—Хопфа

Одна з типових процедур, призначених для навчання окремого модельного нейрона, носить назву **процедури Уїдроу—Хопфа**. Її можна розглядати як реалізацію градієнтного методу з огляду на те, що активаційна функція не є диференційованою [320]. Інша назва цієї процедури — **дельта-процедура.**

Нехай ми бажаємо налаштувати вагові коефіцієнти таким чином, щоб реакція нейрона на будь-який вхідний вектор, помічений як 0, дорівнювала 0, а реакція на будь-який вхідний вектор, помічений як 1, дорівнювала 1. Тоді процедуру зміни вагових коефіцієнтів можна записати таким чином:

$$w^{(i+1)} := w^{(i)} + c\,(d - q)\,x^{(i)},$$

де $x^{(i)}$ — черговий вхідний вектор; d — бажаний вихід; q — фактичний вихід; $w^{(i)}$ — вектор вагових коефіцієнтів, отриманий на даній ітерації; $c > 0$ — параметр, який характеризує швидкість навчання.

30.4. Сигмоїдальні активаційні функції

Порогова активаційна функція не є єдино можливою. Дуже поширеними є так звані **сигмоїдальні активаційні функції** (інша назва — **сигмоїди**). Їх застосування дозволяє уникнути проблем, пов'язаних з недиференційованістю активаційних функцій, і поліпшує властивості процедур навчання.

Сигмоїдальною прийнято називати монотонно зростаючу функцію $f(s)$, яка дорівнює 0 при $s = -\infty$ та 1 при $s = \infty$. Часто за таку функцію приймається

$$f(s) = \frac{1}{1 + e^{-s}}.$$

Для сигмоїдальних активаційних функцій дельта-процедура замінюється на алгоритм, який дістав назву **узагальненої дельта-процедури** [320]:

$$w^{(i+1)} := w^{(i)} + c\,(d - q)\,f\,(1 - q)\,x^{(i)},$$

де $x^{(i)}$ — черговий вхідний вектор; d — бажаний вихід; q — фактичний вихід; $w^{(i)}$ — вектор вагових коефіцієнтів, отриманий на даній ітерації; $c > 0$ — параметр, який характеризує швидкість навчання.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Опишіть модель нейрона як пороговий елемент.
2. Що таке активаційна функція порогового елемента?
3. Сформулюйте результат Мак-Каллока і Піттса про функціональну повноту системи модельних нейронів.
4. Наведіть порогові елементи, які реалізують кон'юнкцію, диз'юнкцію та заперечення.
5. У чому полягає основний принцип навчання модельних нейронів?
6. Опишіть процедуру Уїдроу—Хопфа для навчання модельних нейронів.
7. Які функції, крім порогових, можуть використовуватися як активаційні?
8. Що таке сигмоїдальна функція? Наведіть приклад.
9. Опишіть узагальнену дельта-процедуру для сигмоїдальної активаційної функції.

Розділ 31

ПЕРСЕПТРОНИ ТА СУЧАСНІ НЕЙРОННІ МЕРЕЖІ

Помилки слід не визнавати, а змивати...
кров'ю...

З кінофільму "Кавказька полонянка"

31.1. Персептрон Розенблатта

У 1958 р. Ф. Розенблатт запропонував одну з перших реалізацій нейронної мережі. Вона була достатньо простою, але здатною до навчання на прикладах. Ця схема дістала назву ***персептрона***. Персептрони в основному використовувалися для розпізнавання образів.

Типова схема персептрона подана на рис. 31.1.

Найпростіший персептрон складається з трьох шарів. На першому розташовані чутливі рецепторні елементи (*S*-елементи), на які подаються вхідні зображення. *S*-елементи пов'язані з елементами другого шару (*A*-елементами). *A*-елемент збуджується, тобто генерує вихідний сигнал тоді і тільки тоді, коли збуджується достатня кількість зв'язаних з ним *S*-елементів.

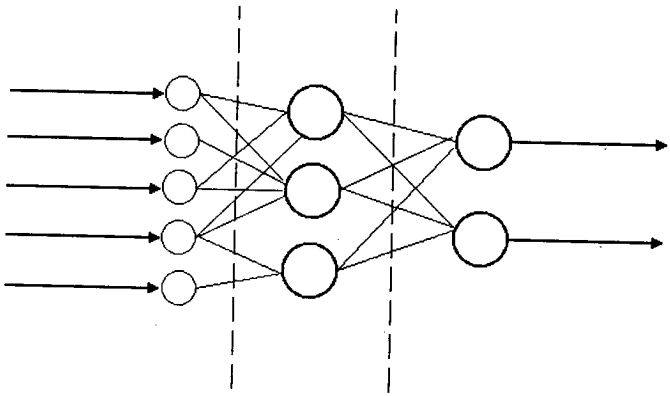


Рис. 31.1. Схема персептрона

A-елементи з'єднуються з *R*-елементами (вихідними елементами) дугами зі змінними ваговими коефіцієнтами. Показаний на рис. 31.1 персептрон відповідає випадку двох класів. Якщо вихід елемента R_1 перевищує вихід елемента R_2, то система відносить об'єкт, який розпізнається, до першого класу, а якщо навпаки — тоді до другого.

Навчання персептрона відбувається шляхом зміни вагових коефіцієнтів. Якщо класифікація правильна, зв'язки залишаються без змін. Навпаки, за неправильної класифікації коефіцієнти зв'язків, що йдуть від елементів, які спрацювали, зменшуються.

У [38] описуються різні модифікації персептрона.

Виявилося, що персептронні моделі можна застосовувати для розпізнавання простих об'єктів. Але початкова ейфорія, пов'язана з персептронами, швидко закінчилася, коли з'ясувалося, що можливості цих пристроїв дуже обмежені.

М. Мінський та С. Пейперт детально проаналізували персептрони з математичної точки зору (результати цього аналізу наведені в [182, 317, 318]). Показано, що персептрони при досить слабких обмеженнях у принципі не можуть розрізнити певні класи об'єктів і мають дуже обмежені можливості. Зокрема, вони не можуть розпізнавати частково затулені об'єкти.

Після цього розвиток персептронної теорії дещо призупинився, і М. Мінський навіть висловлював жаль з приводу опублікування цього результату.

31.2. Загальна характеристика сучасних нейронних мереж

Новий поштовх до подальшого розвитку теорії нейронних мереж був пов'язаний з дослідженнями, які встановили, яким чином можна будувати нейронні мережі так, щоб зняти або послабити обмеження, характерні для персептронів.

Схема навчання персептрона в принципі відповідає дельта-правилу. Тому класичний персептрон може бути використаний для побудови лише лінійної роздільної функції, і відтак добре працює лише в умовах лінійної роздільності класів. Можна довести (див. наведені в [38, 155] ***теореми збіжності***), що персептрон дозволяє отримати лінійну роздільну функцію, якщо така функція насправді існує. Якщо ж лінійної роздільності немає, застосування персептронів не є надійним. У такому зв'язку згадайте алгоритм персептрона, описаний у розд. 26.

Згодом Хеч-Нільсен на основі одного з результатів Колмогорова довів, що, якщо додати до персептрона ще один шар, то вимога лінійної роздільності перестає бути необхідною [121]. Стало ясно, що нейронні мережі повинні бути ***багатошаровими***. Втім розвиток теорії багатошарових нейронних мереж деякий час стримувався через відсутність ефективних алгоритмів їх навчання. Такі алгоритми були запропоновані в 70-х роках. Класичним став ***метод зворотного розповсюдження помилок***. Було запропоновано ряд типів нейронних мереж (***мережі Хопфілда***, ***мережі Кохонена***, ***мережі Гросберга*** та ін).

Усі штучні нейронні мережі складаються на базовому рівні із з'єднаних між собою штучних нейронів. З'єднуючи ШН різним чином, можна одержувати різні типи мереж. Мабуть, найпопулярнішим шляхом з'єднання нейронів у мережу є так звана мережа з послідовними шарами.

Як ми вже зазначали, сутність подібної структури полягає в такому: у мережі із жорстким поділом на шари вхідний шар передає інформацію першому шару обробки, нейрони першого шару передають свої вихідні значення нейронам другого шару обробки, ... і т. д., до тих пір, поки нейрони останнього шару обробки, або вихідного шару, не видадуть вихід мережі в цілому.

Якщо мережа має не жорсткий поділ на шари, різниця полягає в тому, що виходи нейронів деякого шару можуть надходити не лише до нейронів найближчого з наступних шарів, а й до нейронів з довільних наступних шарів. Інакше кажучи, ніякі виходи нейронів даного шару не є входами нейронів попередніх шарів. Звідси й походить назва "мережа з послідовними шарами".

Усупереч цій існує інша, також широко вживана структура з'єднань, відома як мережа Хопфілда, яка є рекурентною, а не послідовною. Рекурентність у даному разі означає те, що нейрон A, який надсилає сигнал, або вихід, якомусь іншому нейрону B, може як вхід одержувати вихід B або якогось іншого нейрона C, якому B надсилає свій вихід, і т. д.

Мережа Хопфілда названа так на честь Джона Хопфілда, фундаментальні праці якого стимулювали піднесення інтересу та збільшення активності у галузі нейронних мереж у 80-ті роки. Протягом останніх років Хопфілд та інші дослідники розробили декілька модифікацій оригінальної мережі Хопфілда [53, 302, 303].

31.3. Штучна нейронна мережа Хопфілда

Штучна нейронна мережа Хопфілда орієнтована на класифікацію k-вимірних двійкових кортежів. Вона складається з k нейроноподібних елементів, які зв'язані "кожен з кожним", і зв'язки між елементами є симетричними, тобто вага t_{ij} зв'язку між елементами i та j дорівнює вазі t_{ji} зв'язку між елементами j та i.

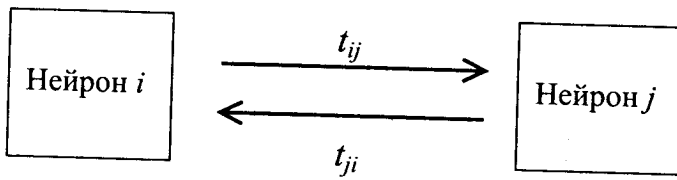


Рис. 31.2. Зв'язки між нейронами мережі Хопфілда: $t_{ij} = t_{ji}$

При обрахуванні виходу, який потім передається до усіх інших нейронів, нейрон мережі використовує порогову функцію Мак-Каллока—Піттса з порогом 0. Якщо мережевий вхід більше порога, або рівний йому, нейрон видає на вихід значення 1, інакше — значення 0.

У довільний момент часу кожен нейрон мережі може знаходитися в одному з двох станів — 0 чи 1 або, інакше кажучи, нейрон може бути "збудженим" чи "незбудженим". Початковий стан мережі задається k-вимірним двійковим кортежем, кожен біт якого визначає стан відповідного нейрона. Можна говорити, що як вхід мережа приймає k-вимірний двійковий кортеж.

Кожен нейрон мережі пов'язаний з усіма іншими, тому необхідно чітко визначити послідовність подій у мережі: в який момент часу кожен нейрон обраховує свій мережевий вхід, використовуючи стани всіх інших нейронів, і вираховує свій новий стан, змінюючи тим самим значення, яке він подає на вхід усіх інших нейронів. Хопфілд постулював, що кожен нейрон перераховує власний стан випадково у часі, але із постійною середньою кількістю перераховувань за секунду. Такий спосіб функціонування можна називати несинхронізованим, або асинхронним.

Хопфілд розглядав мережу такого типу як пристрій, що за відомими еталонними, або "запам'ятованими", кортежами міг би відновлювати частково відомі чи деяким чином пошкоджені кортежі. Таку мережу можна розглядати як асоціативну пам'ять: вона асоціює невідомий їй вхід з одним із запам'ятованих кортежів, і як вихід видає еталонний кортеж, який у певному розумінні найбільш відповідає вхідному.

Хопфілд показав, що досягти такої поведінки можна, задаючи ваги зв'язків між нейронами таким чином: якщо, наприклад, $a = (a_1, a_2, \ldots, a_k)$, $b = (b_1, b_2, \ldots, b_k)$, $c = (c_1, c_2, \ldots, c_k)$ — еталонні кортежі, які мережа має "запам'ятати", тоді ваги $t_{ij} = t_{ji}$, які пов'язують нейрони i та j, повинні дорівнювати:

$$(2 \times a_i - 1) \times (2 \times a_j - 1) + (2 \times b_i - 1) \times (2 \times b_j - 1) + (2 \times c_i - 1) \times (2 \times c_j - 1).$$

Якщо ваги зв'язків налаштовані таким чином, то при поданні на вхід мережі одного з еталонних кортежів — *a*, *b* чи *c*, — мережа "стабілізується", тобто який би з нейронів мережі не перераховував свій стан, його новий стан дорівнюватиме попередньому. Але якщо на вхід мережі надходить "невідомий" кортеж, то стани нейронів починають змінюватись. Цей процес триватиме доти, поки мережа не потрапить у такий стан, в якому вона стабілізується, тобто її стан буде рівний одному з еталонних кортежів. Тоді вважається, що кортеж, який було подано на вхід мережі, потрапляє у той клас, який визначається фінальним еталонним кортежем.

Експериментально показано, що існують істотні обмеження на кількість кортежів, які мережа може запам'ятати. Мережа з K нейронами може знаходитись у 2^K різних станах, але на практиці запам'ятовує не більше ніж $0.15 \times K$ кортежів. До того ж бажано, щоб еталонні кортежі були не дуже схожими між собою.

Наприклад, при розпізнаванні літер необхідно розрізняти близько ста класів зображень — великі та малі літери, цифри, знаки пунктуації тощо. Отже, мережа повинна складатися не менше ніж з 700 нейронів (бажано, аби набагато більше). Така мережа містить $700^2 = 490\,000$ зв'язків, її утримання потребує багато пам'яті, і стабілізація стану такої мережі може виявитися дуже довгою.

На відміну від мережі з поділом на шари, навчання якої є процесом ітеративним і не завжди збіжним, мережа Хопфілда навчається обрахуванням ваг зв'язків за відомими еталонними кортежами. Це означає, що у довільний момент часу їх можна вирахувати знову. А отже, при моделюванні роботи мережі Хопфілда взагалі немає необхідності зберігати значення зв'язків між нейронами мережі. Але час роботи програми при цьому збільшується як мінімум на порядок.

31.4. Загальна схема зворотного розповсюдження помилок

У найзагальніших рисах схему навчання зі зворотним зв'язком можна описати так, як це зроблено в [121]:

1. *Обрати черговий навчальний вектор і подати його на вхід мережі.*
2. *Обчислити вихід мережі.*
3. *Обчислити різницю між фактичним і бажаним виходами.*
4. *Скоригувати вагові коефіцієнти так, щоб мінімізувати помилку.*
5. *Повторювати процедуру, поки помилка не досягне прийнятної величини.*

Легко побачити, що описане вище дельта-правило можна розглядати як частковий випадок цієї загальної схеми. Але для методу зворотного розповсюдження помилок в його загальному вигляді, який можна ефективно застосовувати для багатошарових нейронних мереж, необхідно окремо розглядати навчання нейронів у вихідному шарі і навчання нейронів у проміжних шарах.

31.5. Використання модифікованої мережі Хопфілда при розв'язку задачі розпізнавання літер

Подивимось на розглянуту вже нами задачу розпізнавання з іншого боку. Розпізнавання образів, тобто читання, є для людини традиційно найзручнішим шляхом отримання нової інформації, навчання. Це пояснюється тим, що багато років звітування на папері було єдиним шляхом представлення результатів праці, передачі отриманих знань. У світі накопичено величезну кількість друкованої інформації, і для того щоб її обробляти, людських можливостей у багатьох випадках не вистачає.

За такої постановки задача розпізнавання образів є задачею, джерелом якої виступає сама людина, і з цієї точки зору її розв'язок потрібно будувати, відштовхуючись саме від моделювання сприйняття людиною навколишнього середовища. Тому є природним застосування штучних нейронних мереж для вирішення цієї задачі, бо поняття штучної нейронної мережі виникло саме як модель роботи людського мозку, а задачу розпізнавання образів варто розглядати як класифікаційну задачу. Але перед цим введемо кілька необхідних понять та уточнень.

Зображенням, або образом, називають матрицю вигляду $(a_{ij})_{j=\overline{1,m}}^{i=\overline{1,n}}$. Елементи матриці a_{ij} можуть відрізнятися за типом, і тому розглядають три базових множини матриць.

Множина B_m^n — множина всіх можливих матриць вигляду $(a_{ij})_{j=\overline{1,m}}^{i=\overline{1,n}}$, елементи яких a_{ij} можуть набувати лише два значення: 0 або 1. Такі матриці називають двійковими або монохромними зображеннями і відповідно множину B_m^n — множиною двійкових зображень розмірності $m \times n$.

Множина G_m^n — множина всіх можливих матриць вигляду $(a_{ij})_{j=\overline{1,m}}^{i=\overline{1,n}}$, елементи яких a_{ij} можуть приймати довільні цілі значення з сегмента [0, 255]. Такі матриці називають зображеннями у сірій шкалі та інтерпретують значення елементів матриці як яскравість відповідних пікселів зображення. Множину G_m^n за аналогією називають множиною зображень у сірій шкалі розмірності $m \times n$.

Множина R_m^n — множина всіх можливих матриць вигляду $(a_{ij})_{j=\overline{1,m}}^{i=\overline{1,n}}$, елементи яких a_{ij} можуть приймати довільні дійсні значення з сегмента [0, 1]. Такі матриці, фактично, теж можна розглядати як зображення у сірій шкалі, але між множинами R_m^n і G_m^n є істотна відмінність: існує скінченна кількість матриць розміром $m \times n$, елементи яких набувають цілих значень з фіксованого проміжку. Отже, множина матриць G_m^n є скінченною. Але зрозуміло, що множина R_m^n не є скінченною; існує нескінченна кількість матриць розміром $m \times n$, елементи яких приймають дійсні значення з фіксованого проміжку. Тому множина R_m^n дістає традиційну назву — множина дійсних матриць розмірності $m \times n$ (пам'ятаючи, що елементи цих матриць належать сегменту [0, 1]).

Подальший розгляд робитимемо для множини B_m^n, а якщо це буде необхідно — зробимо уточнення для кожної з базових множин.

Якщо серед усіх елементів множини матриць вибрати кілька фіксованих попарно, не рівних між собою еталонних зображень $\{E_t\}_{t \in T}$, де T — скінченна множина, тоді можна вважати, що вся множина матриць розбивається певним чином на класи еквівалентності, і кожне з еталонних зображень є представником окремого класу образів. Інакше кажучи, задається представник кожного класу, а потім намагаються за ними побудувати канонічне розбиття (або відношення еквівалентності). Зрозуміло, що це можна зробити багатьма способами.

Канонічне розбиття на множині матриць можна задати за допомогою канонічного відображення, тобто такого, яке за конкретною матрицею відновлює клас, до якого вона потрапляє. Наприклад, F: $B_m^n \to [B_m^n/R]$. Тут R — відношення еквівалентності, яке будується, $[B_m^n/R]$ — множина класів еквівалентності, на які розбито множину B_m^n. Відображення F повинно бути всюди визначеним, і, зрозуміло, функціональним. Очевидно, що заданням такого відображення фактично будується відношення еквівалентності R.

У задачі розпізнавання літер важливим є таке розбиття, в якому різні зображення, які з позицій людини відповідають одній літері, потрапляли б до одного класу еквівалентності. Але зрозуміло, що існуватимуть зображення, які не відповідають жодній літері. Отже, в кожній множині матриць мають існувати такі елементи, які не повинні потрапляти до жодного з класів еквівалентності. Тому, щоб зберегти всі побудовані конструкції, до множини еталонів $\{E_t\}_{t \in T}$ додають фіктивний еталон, який позначають E_0 — він буде представляти той клас еквівалентності, до якого потрапляють зображення, що не є літерами.

Отже, нашу задачу розпізнавання можна перетворити на таку задачу класифікації. За множиною еталонних зображень $\{E_t\}_{t \in T}$ потрібно побудувати канонічне відображення F, яке задає розбиття всіх можливих зображень на класи еквівалентності, такі, що зображення, які з людської точки зору відповідають одній літері, потрапляли б до одного класу.

Для розв'язку такої задачі класифікації зручно [35, 36, 38] застосувати модифіковану нейронну мережу Хопфілда, в якій кількість зв'язків залежить від кількості нейронів мережі не квадратично, а лінійно. Всі зображення, що можуть надходити на вхід мережі, подаються як бітові матриці розмірності $m \times n$. Для кожного класу одна з можливих $2^{m \times n}$ матриць є еталонною, тобто являє собою зображення тієї літери, яку мережа повинна розпізнавати — це окреме зображення в подальшому називається еталоном. На відміну від мережі Хопфілда, модифікована мережа налаштовується не на кілька, а лише на один еталон.

За еталоном будується нейроноподібна мережа, що містить $m \times n$ нейронів, кожен з яких однозначно відповідає певному біту вхідного зображення. Позначатимемо як N_{ij} той нейрон, що відповідає біту зображення з координатами (i, j), $i = \overline{1, n}$, $j = \overline{1, m}$. Нейрони зв'язані між собою за таким

правилом: між нейронами N_{ij} та N_{kl} є зв'язок, якщо $|k-i|=1$ або $|1-j|=1$. Фактично це означає, що зв'язуються між собою ті нейрони, які відповідають сусіднім бітам зображення (див. рис. 31.3).

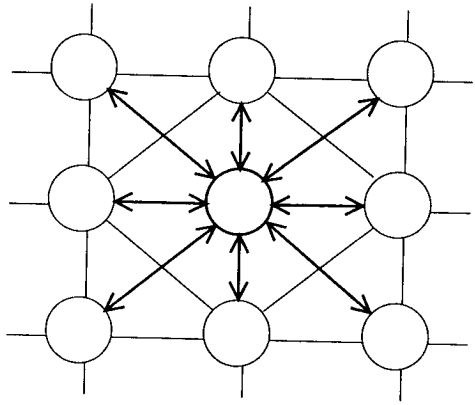


Рис. 31.3. Структура модифікованої мережі (стрілками позначено вісім зв'язків центрального нейрона)

На відміну від мережі Хопфілда, зв'язки тут не є симетричними, тобто якщо між нейронами N_{ij} та N_{kl} є зв'язок, то насправді це два різні зв'язки:

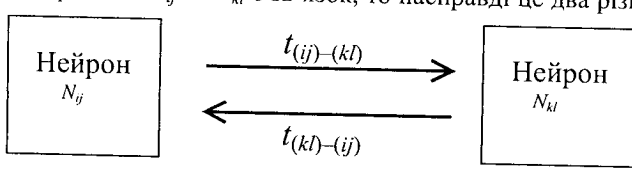


Рис. 31.4. Зв'язки модифікованої мережі: $t_{(ij)-(kl)} \neq t_{(kl)-(ij)}$

За еталонним зображенням кожний нейрон N_{ij} "запам'ятовує" відповідний йому біт (i, j). Це запам'ятоване значення в подальшому називатимемо еталонним бітом нейрона.

Як можна проінтерпретувати таке з'єднання нейронів? Нехай на вхід мережі подано деяке зображення з множини B_m^n. Кожному нейрону у цьому зображенні відповідає деякий біт. Якщо порівняти його з еталонним бітом даного нейрона, можливі чотири випадки:

Початковою впевненістю нейрона називають міру збігу між відповідним бітом зображення та еталонним бітом нейрона. Чому саме "початкова" впевненість? Тому що це значення обраховується для кожного нейрона на початку розпізнавання, тоді, коли зображення потрапляє на вхід мережі.

Значення еталонного біту	Значення біту зображення	Початкова впевненість нейрона
0	0	1
0	1	0
1	0	0
1	1	1

Можна сказати, що для двійкових зображень початкова впевненість — це просто інвертована функція додавання за модулем 2. Для зображень з множин R_m^n та G_m^n початкову впевненість необхідно обраховувати як відстань:

$$D_G(a_{ij}, e_{ij}) = 255 - |a_{ij} - e_{ij}|,$$
$$D_R(a_{ij}, e_{ij}) = 1 - |a_{ij} - e_{ij}|.$$

Тут e_{ij} — запам'ятований еталон нейрона N_{ij}, a_{ij} — значення відповідного пікселя зображення, поданого на вхід. Якщо $e_{ij} = a_{ij}$, тоді впевненість нейрона буде максимальною. Якщо різниця між e_{ij} та a_{ij} максимальна, тоді впевненість нейрона дорівнюватиме нулю.

Перейдемо від окремих нейронів до мережі в цілому і підсумуємо початкову впевненість нейронів. Ми отримаємо величину, яку називають початковою впевненістю мережі. Якщо розглядати зображення з множини B_m^n, то початкова впевненість мережі дорівнює кількості бітів збігу на однакових позиціях в еталонному та вхідному зображеннях. Ця величина є "двоїстою" до відстані Хемінга між зображеннями. Відстанню Хемінга між двома двійковими кортежами однакової розмірності називають кількість незбіжних бітів на однакових позиціях у цих кортежах. Якщо кортежі повністю збігаються, відстань Хемінга між ними дорівнює 0, а початкова впевненість мережі — кількості бітів, або розмірності кортежу. Якщо ж у кортежах не збігається жоден біт, тоді відстань Хемінга між ними дорівнює розмірності кортежу, а початкова впевненість мережі — нулю.

Отже, коли на вхід мережі подається деяке зображення, для кожного нейрона мережі обраховується початкова впевненість, і це значення береться як початкове збудження нейрона. Далі, як і у мережі Хопфілда, вхідне зображення більше не впливає на процес розпізнавання.

Зупинимося на тому, з яких міркувань встановлюються зв'язки між нейронами мережі і яку роль вони відіграють. Кожен нейрон бере до уваги не тільки власну впевненість, але й впевненість сусідніх до нього нейронів. Це дуже подібно до того, як людина сприймає зображення в цілому — якщо картинка загалом знайома, але незначна частина картинки відрізняється від запам'ятованої, людина все ж таки впізнає її. У штучній нейронній мережі це можна промоделювати таким чином: якщо впевненість нейрона є малою, але впевненість його сусідів (зв'язки встановлюються між сусідніми нейронами) достатньо велика, то нейрон "довіряє" їм і збільшує власну впевненість. І навпаки, якщо впевненість нейрона є великою, але впевненість його сусідів замала, то нейрон зменшує власну впевненість.

Опишемо цей процес формальніше. Впевненість кожного нейрона обмежена як згори, так і знизу, для зображень усіх трьох видів: для B_m^n $a_{ij} \in \{0, 1\}$, для G_m^n $a_{ij} \in [0, 255]$, для R_m^n $a_{ij} \in [0, 1]$. Вважатимемо, що якщо впевненість нейрона перевищує 50 % максимальної впевненості, то він збільшує впевненість сусідніх нейронів; якщо впевненість нейрона менше за 50 % максимальної впевненості, то він зменшує впевненість сусідів; у тому разі, коли впевненість нейрона дорівнює 50 % максимальної впевненості, він жодним чином не впливає на впевненість сусідів.

Перерахунок впевненості для деякого нейрона повинен відповідати таким умовам, які випливають з побудови мережі і наведених вище правил. Якщо середня впевненість сусідніх нейронів дорівнює 50 % максимальної, то нейрон не змінює власну впевненість. Якщо середня впевненість сусідів понад 50 % максимальної, то нейрон збільшує власну впевненість пропорційно середній впевненості сусідів; якщо при цьому його нова впевненість перевищує максимально можливу, то вона зменшується до максимальної. Аналогічно, якщо середня впевненість сусідів менша, ніж 50 % максимальної, то нейрон зменшує власну впевненість пропорційно середній впевненості сусідів; якщо при цьому його нова впевненість стає меншою за нуль, то вона збільшується до нуля.

При цьому, коли ми підраховуємо середню впевненість сусідніх нейронів, застосовуються ваги відповідних зв'язків — тих, які йдуть від сусідніх нейронів до центрального. Фактично, це є зважена сума:

$$S = \sum \alpha_i p_i,$$

де α_i — вага зв'язку, який іде від i-го сусіда до центрального нейрона, p_i — впевненість i-го сусіда. Ваги зв'язків можна розглядати як ступінь довіри між нейронами мережі: чим більше вага $t_{(ij)-(kl)}$, тим більше впевненість нейрона N_{kl} впливає на впевненість нейрона N_{ij}.

Робота мережі аналогічна мережі Хопфілда: після подання вхідного зображення в мережу та обрахунку початкової впевненості нейрони починають асинхронно змінювати власну впевненість (перераховувати її за впевненостями сусідів). Якщо цей процес продовжувати досить довго, то можливі лише два випадки.

Якщо мережа стабілізується, тобто настає момент, коли жоден нейрон не може змінити власну впевненість, остаточну впевненість мережі покладають рівній середній впевненості всіх нейронів мережі. Здається, якщо кількість нейронів з малою впевненістю більша, ніж кількість нейронів з великою впевненістю, то остаточна впевненість мережі повинна прямувати до нуля. І навпаки, якщо кількість нейронів з великою впевненістю більша, ніж кількість нейронів з малою впевненістю, то остаточна впевненість мережі повинна прямувати до максимально можливої. Найчастіше так і буває, але, на жаль, можливі ситуації, коли мережа стабілізується у деякому стані, де наявні нейрони, впевненість яких є проміжною між нулем та максимальною впевненістю.

Якщо мережа не стабілізується занадто довго, треба зупинити перерахунки і вважати, що остаточна впевненість мережі дорівнює 50 % від максимальної — це дозволить уникнути зациклень.

Коли за цим алгоритмом отримано остаточну впевненість мережі, її інтерпретують так: якщо вона більша або дорівнює певній пороговій впевненості (наприклад, 50 % від максимальної), ми вважаємо, що мережа відповідає "так". Це зображення потрапляє до того ж самого класу, що й еталонне зображення даної мережі. Інакше мережа відповідає "ні" — це зображення з іншого класу.

Налаштування зв'язків між нейронами мережі проводиться за тією ж схемою, що і для персептрона Розенблатта — ітеративно, багаторазово для кожної пари навчальної вибірки. При перерахунку впевненості деякого нейрона сусідні нейрони можна розглядати як входи. Але тут вже немає "активних" чи "пасивних" входів, і ваги змінюються пропорційно до впевненості кожного входу.

Якщо на вхід мережі подане зображення, яке потрапляє в той самий клас, що й еталонне зображення, а мережа відповідає "ні", збільшують ваги, які йдуть від тих нейронів, початкова впевненість яких перевищує 50 % максимальної. Збільшення робиться не на константу, а пропорційно до початкової впевненості нейрона, від якого йдуть ці ваги.

Якщо на вхід мережі подане зображення, яке належить до іншого класу, а мережа відповідає "так", роблять усе навпаки — зменшують ваги, що йдуть від тих нейронів, початкова впевненість яких більша, ніж 50 % максимальної. Зменшення також роблять і початковій впевненості нейрона, від якого йдуть ці ваги.

Що ж врешті отримують? Для довільного зображення мережа за цим алгоритмом обраховує ступінь відповідності його еталонному зображенню, і цей процес завжди зупиняється, повертаючи правильний з погляду здорового глузду результат. Отже, ми можемо за цим алгоритмом побудувати канонічне відображення F: $B_m^n \to [B_m^n/R]$.

Для кожного еталонного зображення з множини $\{E_t\}_{t\in T}$ будується описана вище мережа. Проводиться навчання за існуючою навчальною вибіркою, після чого можна починати розпізнавання. При розпізнаванні деякого зображення воно подається на вхід кожної мережі окремо, і кожна мережа вирішує, чи відповідає воно еталону мережі, і якщо так, то наскільки. Якщо кілька мереж (тобто більше за одну) вважає, що це зображення з їх класу, вибирають переможцем ту, остаточна впевненість якої була найвища. Якщо і така мережа не одна, виграє та, швидкість збіжності якої була найбільшою. Якщо ж усі мережі при поданні на вхід деякого зображення відповідають "ні", вважається, що воно потрапляє у клас, утворений за фіктивним еталоном E_0.

Однак у разі практичної реалізації запропонованої системи розпізнавання ми повинні пам'ятати про дві основні проблеми.

По-перше, потрібно якось реалізовувати задачу виокремлення символьної інформації на зображенні аркуша паперу. Це означає, що на вхід системи подається не зображення окремої літери (у попередньому смислі, з множин B_m^n, G_m^n чи R_m^n), а картинка, на якій друкованими або рукописними літерами "написано" кілька речень (наприклад, відскановане зображення звичайного аркуша паперу). Як результат роботи системи розпізнавання користувач бажає отримати текстовий файл, зміст якого збігався б з тим, що було написано на папері й відсканова но. Без вирішення цієї проблеми розпізнавання літер перетворюється на виключно теоретичну задачу.

По-друге, як би гарно система не розпізнавала символи у "лабораторних" умовах, при спробі її реалізації програміст стикається з цілою низкою пов'язаних між собою проблем технічного характеру. Ці проблеми поділяються на чотири основні класи, які можна охарактеризувати так: зашумленість зображення, різниця в масштабах зображень, виділення на зображенні окремих рядків і слів у рядках, виділення окремих літер у словах.

Запропонований підхід до системи розпізнавання літер був реалізований у праці [53]. Проведені практичні експерименти засвідчили, що використання штучних нейронних мереж для розв'язування задач розпізнавання зображень, хоча й має певні недоліки, але в цілому є досить перспективним. Як було показано, на якість розпізнавання сильно впливають сторонні фактори, не пов'язані з розпізнаванням як таким: необхідність виокремлення символьної інформації на зображенні, уніфікації розмірів зображень літер тощо. Але навіть при цьому досягнута якість розпізнавання є цілком задовільною.

Можна також запропонувати кілька напрямів розвитку цього підходу. Найважливіший з них — удосконалення моделі нейроноподібної мережі. Побудована модель є сукупністю не зв'язаних між собою штучних нейронних мереж, у той час як нейрони людського мозку об'єднані в загальну, неподільну систему — переваги такої структури безсумнівні.

Беручи до уваги значний розвиток нейрокомп'ютерів і супутніх технологій за останній період, цілком імовірно, що через деякий час апарат штучних нейронних мереж отримає сильну практичну підтримку, і використання ШНМ стане набагато поширенішим.

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Опишіть схему функціонування персептрона Розенблатта.
2. Для яких задач в основному використовувалися персептрони?
3. З чим пов'язана обмеженість класичного персептрона?
4. В якому випадку персептрон дозволяє знайти роздільну функцію?
5. Як потрібно змінити класичний персептрон, щоб вимога лінійної роздільності перестала бути необхідною?
6. Опишіть загальну схему методу зворотного поширення помилок.

ЧАСТИНА VIII | РОЗУМІННЯ І СПІЛКУВАННЯ

Розділ 32
РОЗУМІННЯ ПРИРОДНОЇ МОВИ

И тогда я поняла, чё те надо, чё те надо,
Но не дам я тебе чё ты хошь.
З пісні

32.1. Основні задачі, пов'язані з обробкою природної мови

Розуміння природної мови являє собою одну з найважливіших і найскладніших проблем теорії та практики штучного інтелекту. Ця задача найчастіше пов'язується з проблемою створення систем штучного інтелекту, які здатні вести осмислений ***діалог*** з людиною. Відомо, що за останнє десятиліття в організації людино-машинного інтерфейсу було досягнуто величезного прогресу, але цей прогрес було в основному здобуто іншим, обхідним шляхом — на основі систем підказок, меню тощо. Успіхи ж власне у розумінні природної мови залишаються скромними: природна мова є надто складною і неоднозначною. І справа тут не лише в розумінні. В процесі діалогу машина має **реагувати** на фрази людини, і ця відповідь повинна бути максимально адекватною і соціально бажаною; вона має враховувати мету діалогу, індивідуальні особливості співрозмовника тощо. На сучасному етапі неможливо сподіватися на повне вирішення цих проблем. Водночас можна і потрібно говорити про створення ***обмежених природних мов***, які, з одного боку, припускатимуть ефективні методи аналізу, а з іншого — вони повинні бути достатньо природними для людини, а їх вживання не повинно становити для неї особливих труднощів.

Але задача обробки природної мови виникає не тільки в процесі діалогу з людиною. Можна згадати, наприклад, ***системи автоматизованого перекладу***, які дозволяють здійснювати переклад з однієї мови на іншу (дуже відомими є, зокрема, програми Stylus, ПРОМТ та ін.). Якість такого перекладу все ще досить низька, і отриманий переклад вимагає серйозних подальших доробок. Серед професійних перекладачів немає єдності з приводу доцільності застосування таких систем. Дехто переконаний, що вони істотно полегшують роботу перекладача; інші ж вважають, що якісний переклад простіше зразу набирати вручну.

У цьому розділі ми розглянемо обробку речень, отриманих у вигляді готових текстів. Якщо ж мову доводиться сприймати на слух, то, як ви знаєте, виникає ще одна задача — задача розпізнавання мови [50].

32.2. Типова схема обробки природної мови

При всьому розмаїтті задач, пов'язаних з обробкою природної мови, і підходів до їх вирішення, роботу систем штучного інтелекту з природною мовою можна уявляти у вигляді такої загальної схеми:

1. **Сприйняття тексту**, написаного природною мовою (на практиці, як правило, обмеженою, речення якої прості за своєю структурою і належать до обмеженої предметної області).
2. **Зіставлення тексту зі своїми знаннями і переклад його на внутрішню мову системи** (саме цей етап, власне кажучи, і складає основу розуміння).
3. **Планування відповіді** на основі тих чи інших алгоритмів, специфічних для даної предметної області або загальноінтелектуальних метапроцедур.
4. **Генерація відповіді** — у вигляді зв'язного тексту або, можливо, в деякій іншій формі.

Для вирішення певних практичних задач не обов'язково прагнути до глибокого розуміння. Так, в основі роботи ряду інформаційно-пошукових систем, зокрема ***інтелектуальних агентів*** (автономних мобільних програмних засобів, призначених насамперед для роботи в Інтернеті [58, 321, 331]) лежать прості продукційні правила, які дуже нагадують ідею фатичного діалогу.

Слід розрізняти такі види аналізу речень природної мови:

- ***морфологічний***, який виокремлює слова та форми слів;
- ***синтаксичний***, метою якого є з'ясування структури речення;
- ***семантичний***, у процесі якого з'ясовується значення фрази та робиться зіставлення його зі знаннями системи;
- ***прагматичний*** — з'ясування, з якою метою було сказане речення.

Для сучасних методик характерне значне взаємопроникнення всіх видів аналізу.

Крім цього, необхідним є розуміння не тільки окремих речень, але й усього тексту.

32.3. Рівні розуміння

Можна вважати, що система розуміє текст, якщо вона дає правильні відповіді на будь-які запитання, що мають відношення до тексту. Інтуїтивно ясно, що "розуміти" тексти можна різною мірою та з різною глибиною. У [129] наводиться така класифікація рівнів розуміння.

Нехай на вході системи маємо текст $B = <T, E, P>$, де T — власне текст; E — розширений текст, що включає умови його породження; P — розширений текст, який враховує інформацію про суб'єкта, який породжує текст.

Крім цього, введемо такі позначення: *TR* — формальні правила поповнення тексту, що базуються на його внутрішній структурі та псевдофізичних логіках; *ER* — правила поповнення тексту, що базуються на знаннях системи про світ; *PR* — правила поповнення тексту, що базуються на комунікативних і психологічних знаннях, а також на знаннях про особистість того, хто говорить; *A* — відповідь, що генерується системою.

Перший рівень характеризується схемою $T \rightarrow A$, відповіді генеруються лише на основі прямого та безпосереднього сприйняття інформації. Нехай система сприймає, наприклад, такий текст:

"О восьмій ранку Федя пішов на роботу. Він повернувся о восьмій вечора, після чого повечеряв і пішов спати".

При першому рівні розуміння система в змозі відповідати на запитання *"Коли Федя пішов на роботу?"* або *"Коли він повернувся?"*.

Другий рівень характеризується схемою $<T, TR> \rightarrow A$. Система в змозі відповідати на запитання типу *"Що було раніше: повернення Феді чи його вечеря?"*.

Третій рівень характеризується схемою $<T, TR, ER> \rightarrow A$. Система може відповісти на запитання типу *"Де був Федя о четвертій дня?"*. Легко зрозуміти, що відповідь може істотно залежати від наявних знань.

Четвертий рівень характеризується схемою $<E, TR, ER> \rightarrow A$. Крім власне тексту, беруться до уваги інші джерела інформації. Наявність таких джерел дає можливість правильно сприймати висловлювання типу *"Подивись, яка погода! Чи варто їхати на пікнік?"*.

П'ятий рівень розуміння, який характеризується схемою $<P, TR, PR> \rightarrow$ $\rightarrow A$ бере до уваги інформацію про конкретну особистість, що є джерелом тексту, а також загальну інформацію про психологію спілкування, мету учасників спілкування тощо. Так, якщо система сприймає слова Васі про те, що *"Таня обіцяла незабаром повернутися"*, слід брати до уваги:

- чи слід взагалі довіряти Васі?
- який зміст вкладає Вася у слово "незабаром?";
- що ми знаємо про Таню та про те, куди вона пішла?

Крім цього, розглядаються ***метарівні розуміння***, які визначають можливі зміни у базах знань та сприйняття і породження метафор та асоціацій.

32.4. Глибинні відмінки

Розглянемо один з найбільш відомих і класичних підходів до розуміння речень, написаних природною мовою. Він полягає в аналізі структури речення на основі **глибинних відмінків**. Під глибинним відмінком слід розуміти роль, яку відіграє дане слово в реченні; ця роль визначає відношення слова до основної ситуації, яка описується в реченні.

Виокремлюють різні системи глибинних відмінків. Однією з найперших є ***система відмінків Філмора*** [279], основними з яких є такі: **агент** (суб'єкт, що є ініціатором події); **контрагент** (сила, проти якої спрямова-

но дію), **пацієнс** (суб'єкт, що відчуває на собі ефект дії), **джерело** (місце, з якого починається дія) тощо.

Звичайно, виокремлення структури глобальних відмінків має сенс лише тоді, коли вдається розробити **формальні правила**, за допомогою яких можна отримувати конкретні значення цих відмінків. Ці правила можуть ґрунтуватися на формальній структурі речення (наприклад, деякий відмінок може йти лише після визначеного прийменника), на специфічних знаннях про предметну область і т. п.

Однією з перших програм, побудованих на цій основі, була відома програма Т. Віноградa [49].

32.5. Типова схема аналізу речень на основі глибинних відмінків

П. Уїнстон у [254] описує, яким чином можна використовувати теорію глибинних відмінків при аналізі речень, які написані простою мовою і належать до обмеженої предметної області.

У більшості речень йдеться про певну дію або перехід від одного стану до іншого. Це описується дієсловом, яке в такому випадку виступає ключовим елементом речення. Отже, аналіз речення повинен починатися з виокремлення основного дієслова.

Далі з'ясовуються значення глибинних відмінків, що показують, як саме іменники, що входять до речення, пов'язані з ключовим дієсловом. П. Уїнстон описує такі відмінки:

1. **Агент** — те, що викликає дію або зміну стану. Наприклад, у реченні *"Дівчина нагодувала цуцика"* як агент виступає слово *"дівчина"*.
2. **Об'єкт** — те, на що спрямована дія. У попередньому реченні об'єкт — це *"цуцик"*.
3. **Інструмент** — засіб, який використовується агентом.
4. **Співагент** — той, хто допомагає агентові в здійсненні дії.
5. **Пункт відправки** — початковий стан.
6. **Пункт призначення** — кінцевий стан. Наприклад: *"Дівчина пішла зі школи додому"*.
7. **Засіб доставки**. Спосіб, яким здійснюється переміщення з пункту відправки до пункту призначення.
8. **Траєкторія**. Траєкторія, по якій здійснюється переміщення.
9. **Місцезнаходження**. Місце, де здійснюється дія.
10. **Споживач**. Той, для кого виконується дія.
11. **Сировина**. Відмінок "сировина" виникає, якщо деякий матеріал зникає, перетворюючись на продукт. Наприклад: *"Таня зробила компот з яблук"*.
12. **Час**. Час виконання дії.

Отже, аналіз речення можна розглядати як заповнення деякої фреймоподібної структури. Слотами цієї структури виступає ключове дієслово та глибинні відмінки, які розкривають ролі іменників стосовно дієслова.

Можна також вводити слоти, які описують ознаки іменників, спосіб виконання дії (*"сильно"*, *"слабко"* і т. д).

Після того як аналіз речення проведено, система може відповідати на запитання, пов'язані з цим реченням. **Для відповіді на запитання, очевидно, достатньо звернутися до слотів, асоційованих з даним типом запитань.**

Наприклад, до системи вводиться текст *"Вася написав вірус"*, після чого її запитують *"Хто написав вірус?"*. Якщо з запитанням *"Хто?"* асоційований відмінок *"Агент"*, система звертається до відповідного слоту і відповідає *"Вася"*.

32.6. Розширені мережі переходів

При аналізі природної мови широке застосування знаходить формалізм ***розширених мереж переходів*** [167, 254], які збільшують можливості базових мереж переходів. ***Базова мережа переходів*** являє собою граф переходів скінченного автомата і, відповідно, дозволяє аналізувати мови, що породжуються автоматними граматиками.

Рекурсивні мережі переходів дозволяють здійснювати рекурсивні виклики. Вони відповідають контекстно-вільним граматикам.

Нарешті, розширена мережа переходів являє собою рекурсивну мережу переходів, дуги якої можуть мати додаткові мітки. Ці додаткові мітки, по-перше, можуть задавати умови, за яких може бути здійснено перехід (наприклад, перехід неможливий, якщо значенням деякого слоту є певне "заборонене" слово"), а по-друге, дії, які слід здійснити при реалізації переходу (наприклад, занести до певного слоту останнє слово).

КОНТРОЛЬНІ ЗАПИТАННЯ

1. Опишіть типову схему обробки текстів, написаних природною мовою.
2. Охарактеризуйте морфологічний, синтаксичний, семантичний і прагматичний етапи аналізу речень.
3. Наведіть відомі вам рівні розуміння.
4. Охарактеризуйте поняття глибинного відмінка. Наведіть власні приклади.
5. Опишіть методику аналізу речень на основі глибинних відмінків.
6. Що таке розширена мережа переходів?

ТЕМИ ДЛЯ ОБГОВОРЕННЯ

1. Охарактеризуйте можливі напрями розвитку діалогових людино-машинних систем та систем автоматизованого перекладу.
2. Що змінюється при переході від нижчого рівня розуміння до вищого?
3. Наведіть приклади речень, які можна перекласти з однієї мови на іншу без розуміння змісту цих речень.

ЛІТЕРАТУРА

1. *Абрамов А. С.* Элементы анализа программ.— М.: Наука, 1986.— 128 с.
2. *Аверкин А. Н., Батыршин И. З., Блишун А. Ф. и др.* Нечеткие множества в моделях управления и искусственного интеллекта.— М.: Наука, 1986.— 312 с.
3. *Адельсон-Вельский Г. М.* Программирование игр.— М.: Наука, 1978.— 255 с.
4. *Адельсон-Вельский Г. М., Арлазаров В. Л., Битман А. Р. и др.* Машина играет в шахматы.— М.: Наука, 1983.— 208 с.
5. *Айвазян С. А., Бежаева З. И., Староверов О. Л.* Классификация многомерных наблюдений.— М.: Статистика, 1974.— 239 с.
6. *Айзенк Г.* Проверьте свои способности.— М.: Мир, 1992.
7. *Айзенк Г.* Коэффициент интеллекта.— К.: Гранд, 1994.
8. Алгоритмы и программы решения задач на графах и сетях / М. И. Нечипоренко, В. К. Попков, С. М. Найнагашев и др.— Новосибирск: Наука, Сиб. отд., 1990.
9. *Амосов Н. М.* Алгоритмы разума.— К.: Наук. думка, 1979.— 220 с.
10. *Андон Ф. И., Ящунин А. Е., Резниченко В. А.* Логические модели интеллектуальных информационных систем.— К.: Наук. думка, 1999.— 398 с.
11. *Анисимов А. В.* Рекурсивные преобразователи информации.— К.: Вища шк., 1987.— 392 с.
12. *Анісімов А. В., Глибовець М. М., Кравченко І. В., Олецький О. В. та ін.* Системи штучного інтелекту: Навч. посібник.— К.: ВПЦ "Київський університет", 2000.— 100 с.
13. *Аоки М.* Введение в методы оптимизации.— М.: Наука, 1978.
14. *Ахмед Н., Рао К.* Ортогональные преобразования при обработке сигналов.— М.: Связь, 1980.— 248 с.
15. *Ахо А., Хопкрофт Дж., Ульман Дж.* Построение и анализ вычислительных алгоритмов.— М.: Мир, 1979.— 536 с.
16. *Ахо А., Хопкрофт Дж., Ульман Дж.* Структуры данных и алгоритмы.— М.: Вильямс, 2000.— 384 с.
17. *Басакер Р., Саати Г.* Конечные графы и сети.— М., 1974.
18. *Батыщев Д. И.* Методы оптимального проектирования.— М., 1984.
19. *Башняков О. М., Гаращенко Ф. Г., Пічкур В. В.* Практична стійкість та структурна оптимізація динамічних систем.— К.: ВПЦ "Київський університет", 2000.— 197 с.
20. *Беллман Р.* Динамическое программирование.— М.: Изд-во иностр. лит-ры, 1960.
21. *Белнап Н., Стил Т.* Логика вопросов и ответов.— М.: Прогресс, 1981.— 287 с.
22. *Бонгард М. М.* Проблема узнавания.— М.: Наука, 1967.— 320 с.
23. *Бондарев В. М., Рублинецкий В. И., Качко Е. Г.* Основы программирования.— Харьков: Фолио, 1997.— 368 с.
24. *Ботвинник М. М.* О решении неточных переборных задач.— М.: Сов. радио, 1979.— 152 с.
25. *Браверман Э. М., Мучник И. Б.* Структурные методы обработки эмпирических данных.— М.: Наука, 1983.— 464 с.

26. *Братко И.* Программирование на языке Пролог для систем искусственного интеллекта.— М.: Мир, 1990.— 560 с.
27. *Бублик Б. Н., Гаращенко Ф. Г.* Оптимизация структур линейных резонансных ускорителей: Препринт АН УССР: Ин-т математики; 83.7.— К., 1983.— 62 с.
28. *Бублик Б. Н., Гаращенко Ф. Г., Кириченко Н. Ф.* Структурно-параметрическая оптимизация и устойчивость динамики пучков.— К.: Наук. думка, 1985.— 304 с.
29. *Бублик Б. Н., Кириченко Н. Ф.* Основы теории управления.— К.: Вища шк., 1975.— 328 с.
30. *Бусленко Н. П., Калашников В. В., Коваленко И. Н.* Лекции по теории сложных систем.— М.: Сов. радио, 1973.— 440 с.
31. *Буч Г.* Объектно-ориентированный анализ с примерами приложений на C++.— М.: Бином, СПб.: Невский диалект, 1999.— 560 с.
32. *Буч Г., Рамбо Дж., Джекобсон А.* Язык UML: Руководство пользователя.— М.: ДМК, 2000.— 432 с.
33. *Вагин В. Н.* Дедукция и обобщение в системах принятия решений.— М.: Наука, 1988.
34. *Вагнер Г.* Основы исследования операций.— Т. 1—3.— М.: Мир, 1973.
35. *Вапник В. Н.* Задача обучения распознаванию образов.— М.: Знание, 1971.
36. *Вапник В. Н., Червоненкис А. Я.* Теория распознавания образов.— М.: Наука, 1974.— 415 с.
37. *Варшавский В. И., Поспелов Д. А.* Оркестр играет без дирижера.— М.: Наука, 1984.— 208 с.
38. *Васильев В. И.* Распознающие системы.— К.: Наук. думка, 1983.
39. *Васильев В. И., Шевченко А. И.* Искусственный интеллект: Проблема обучения опознаванию образов.— Донецк, 1997.— 223 с.
40. *Васильев Ф. П.* Лекции по методам решения экстремальных задач.— М.: Изд-во МГУ, 1974.
41. *Васильев Ф. П.* Численные методы решения экстремальных задач.— М.: Наука, 1988.— 552 с.
42. *Вельбицкий И. В.* Технология программирования.— К.: Техника, 1984.— 279 с.
43. *Вентцель Е. С.* Теория вероятностей.— М.: Наука, 1969.— 576 с.
44. *Верлань А. Ф., Горошко И. О., Олецкий А. В.* Объектно-ориентированная архитектура интеллектуального решателя задач обработки и интерпретации экспериментальных зависимостей: У зб. наук. праць: Моделювання та інформаційні технології.— Вип. 1.— Львів: Світ, 1999.— С. 11—18.
45. *Верлань А. Ф., Сизиков В. С.* Интегральные уравнения: Методы, алгоритмы, программы: Справ. пособие.— К.: Наук. думка, 1986.— 544 с.
46. *Верхаген К., Дейн Р. и др.* Распознавание образов: Состояние и перспективы.— М.: Радио и связь, 1985.— 104 с.
47. *Винер Н.* Кибернетика.— М.: Сов. радио, 1968.
48. *Винер Н.* Кибернетика, Или управление и связь в животном и машине.— М.: Наука, 1983.
49. *Виноград Т.* Программа, понимающая естественный язык.— М.: Мир, 1976.
50. *Винцюк Т. К.* Анализ, распознавание и интерпретация речевых сигналов.— К.: Наук. думка, 1987.— 264 с.
51. *Вирт Н.* Алгоритмы + Структуры данных = Программы.— М.: Мир, 1985.— 406 с.
52. *Вирт Н.* Систематическое программирование: Введение.— М.: Мир, 1977.— 183 с.

53. *Вороновский Г. К., Сергеев С. А. и др.* Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности.— Харьков: Основа, 1997.
54. Вычислительные машины и мышление.— М.: Мир, 1967.
55. *Гаазе-Рапопорт М. Г., Поспелов Д. А.* От амебы до робота: Модели поведения.— М.: Наука, 1987.— 288 с.
56. *Гаазе-Рапопорт М. Г., Поспелов Д. А., Семенова Е. Т.* Порождение структур волшебных сказок: Препринт АН СССР.— М.: НСКП "Кибернетика", 1980.
57. *Гаврилова Т. А., Червинская К. Р.* Извлечение и структурирование знаний для экспертных систем.— М.: Радио и связь, 1992.
58. *Гаврилова Т. А., Хорошевский В. Ф.* Базы знаний интеллектуальных систем.— СПб.: Питер, 2000.— 384 с.
59. *Гаек П., Гавранек Т.* Автоматическое образование гипотез: Математические основы общей теории.— М.: Наука, 1984.— 280 с.
60. *Гаращенко Ф. Г.* Исследование задач практической устойчивости численными методами и оптимизация динамики пучков // Прикладная математика и механика.— 1987.— Т. 51.— № 5.— С. 717—723.
61. *Гаращенко Ф. Г.* Недифференцируемые задачи структурно-параметрической оптимизации и проектирование ускоряющих и фокусирующих систем // Автоматика.— 1986.— № 1.— С. 50—53.
62. *Гаращенко Ф. Г.* Устойчивость по направлению, построение оптимальных функций Ляпунова и множеств устойчивости // Автоматика.— 1986.— № 5.— С. 39—42.
63. *Гаращенко Ф. Г., Башняков А. Н., Пичкур В. В.* О недифференцируемых задачах оптимального управления матричными дифференциальными уравнениями // Кибернетика и системн. анализ.— 1998.— № 5.— С. 92—101.
64. *Гаращенко Ф. Г., Осипчук М. В.* О некоторых минимаксных задачах матричной параметрической оптимизации // Проблемы управления и информатики.— 1995.— № 3.— С. 50—60.
65. *Гаращенко Ф. Г., Панталієнко Л. А.* Аналіз та оцінка параметричних систем.— К.: ІСДО, 1995.— 140 с.
66. *Гаращенко Ф. Г., Панталиенко Л. А.* Исследование задач практической устойчивости систем с переменной структурой // Автоматика.— 1993.— № 4.— С. 43—55.
67. *Гаращенко Ф. Г., Пічкур В. В.* Застосування узагальненого принципу Беллмана до структурної оптимізації динамічних систем // Праці міжнар. конф. "Modelling and Investigation of systems stability", Systems Investigation (May 19—23, 1997).— К.: Київ. ун-т ім. Т. Шевченка, 1997.— С. 24.
68. *Гаращенко Ф. Г., Пичкур В. В.* Об оптимальных множествах практической устойчивости // Проблемы управления и информатики.— 1998.— № 5.— С. 5—18.
69. *Гаращенко Ф. Г., Пичкур В. В.* Структурная оптимизация динамических систем на основе обобщенного принципа Беллмана // Проблемы управления и информатики.— 1997.— № 6.— С. 6—13.
70. *Гарднер М.* Путешествие во времени.— М.: Мир, 1990.— 341 с.
71. *Гасс С.* Линейное программирование: Методы и приложения.— М.: Физматгиз, 1961.
72. *Гемпель К. Г.* Логика объяснения.— М., 1998.— 240 с.
73. *Герман О. В.* Введение в теорию экспертных систем и обработку знаний.— Минск: ДизайнПРО, 1995.— 255 с.

74. *Гик Е. Я.* Занимательные математические игры.— М.: Знание, 1987.— 160 с.
75. *Гилл Ф.* Практическая оптимизация.— М.: Мир, 1985.— 569 с.
76. *Гирсанов И. В.* Лекции по математической теории экстремальных задач.— М.: Изд-во МГУ, 1970.
77. *Гладун В. П.* Планирование решений.— К.: Наук. думка, 1987.— 168 с.
78. *Гладун В. П.* Эвристический поиск в сложных средах.— К.: Наук. думка, 1977.— 166 с.
79. *Глыбовец Н. Н.* Структурно-функциональные сети: Автореф. дисс. ... канд. физ.-мат. наук.— К.: Вища шк., 1987.— 13 с.
80. *Глибовець М. М., Глибовець А. М.* Евристичний алгоритм побудови розкладу роботи багатопроцесорного комплексу // Проблемы программирования, 1998, № 2.— С. 34—40.
81. *Глибовець М. М., Кравченко І. В., Олецький О. В., Терещенко В. М.* Програмування в Пролозі: Навч. посібник.— К.: ВПЦ "Київський університет", 1998.— 110 с.
82. *Глушков В. М.* Введение в кибернетику.— К., 1964.
83. *Глушков В. М.* Основы безбумажной информатики.— М.: Наука, 1982.— 552 с.
84. *Глушков В. М., Цейтлин Г. Е., Ющенко Е. Л.* Алгебра. Языки. Программирование.— К.: Наук. думка, 1974.— 328 с.; 2-е изд., 1978.— 318 с.; 3-е изд., 1989.— 368 с.
85. *Гнеденко Б. В.* Курс теории вероятностей.— М.: Наука, 1988.— 448 с.
86. *Головкин Б. А.* Вычислительные системы с большим числом процессоров.— М.: Радио и связь, 1995.
87. *Головкин Б. А.* Параллельные вычислительные системы.— М.: Наука, Гл. ред. физ.-мат. лит-ры, 1980.
88. *Головкин Б. А.* Расчет характеристик и планирование параллельных вычислительных процессоров.— М.: Радио и связь, 1983.— 324 с.
89. *Горелик А. Л., Скрипкин В. А.* Методы распознавания.— М.: Высш. шк., 1989.— 232 с.
90. *Гремилов А. А.* Как принять наилучшее решение в реальных условиях.— М.: Радио и связь, 1991.— 320 с.
91. *Гренандер У.* Лекции по теории распознавания образов: В 3-х т.— М.: Мир, 1979—1983.
92. *Гринченко Т. А., Стогний А. А.* Машинный интеллект и новые информационные технологии.— К.: Манускрипт, 1993.— 168 с.
93. *Гриценко В. И., Бакаев А. А., Козлов Д. Н.* Методы организации и обработки баз знаний.— К.: Наук. думка, 1993.— 150 с.
94. *Гриценко В. И., Бакаев А. А., Козлов Д. Н.* Экспертные системы и логическое программирование.— К.: Наук. думка, 1992.— 219 с.
95. *Грунский И. С., Козловский В. А., Пономаренко Г. Г.* Представления конечных автоматов фрагментами поведения.— К.: Наук. думка, 1990.— 232 с.
96. *Гупал А. М., Пашко С. В., Сергиенко И. В.* Эффективность байесовской процедуры классификации объектов // Кибернетика и системн. анализ.— 1995.— № 4.— С. 76—89.
97. *Гэри М., Джонсон Д.* Вычислительные машины и труднорешаемые задачи.— М., 1982.— 416 с.
98. *Даффин Р., Питерсон Э., Зенер К.* Геометрическое программирование.— М., 1972.— 312 с.
99. *Дейт К. Дж.* Введение в системы баз данных.— К.: Диалектика, 1998.— 784 с.
100. *Джордж Ф.* Основы кибернетики.— М.: Радио и связь, 1984.— 272 с.

101. *Диниц Е. А.* Алгоритм решения задачи о максимальном потоке в сети со степенной оценкой: Докл. АН СССР; Серия "Мат., Физ.", 1970.— С. 754—757.
102. *Довгялло А. М.* Диалог пользователя и ЭВМ: Основы проектирования и реализации.— К.: Наук. думка, 1981.— 232 с.
103. *Довгялло А. М., Брановицкий В. И., Вершинин К. П. и др.* Диалоговые системы: Современное состояние и перспективы развития.— К.: Наук. думка, 1987.— 248 с.
104. *Донец Г. А., Шор Н. З.* Алгебраический подход к раскраске плоских графов.— М., 1982.
105. *Драган Я. П.* Структура и представления моделей стохастических сигналов.— К.: Наук. думка, 1980.— 384 с.
106. *Дрейфус Х.* Чего не могут вычислительные машины: Критика искусственного разума.— М.: Прогресс, 1978.
107. *Дуда Р., Харт П.* Распознавание образов и анализ сцен.— М.: Мир, 1976.
108. *Дюбуа Д., Прад А.* Теория возможностей: Приложения к представлению знаний в информатике.— М.: Радио и связь, 1990.— 288 с.
109. *Дюк В. А.* Компьютерная психодиагностика.— СПб.: Братство, 1994.
110. *Евстигнеев В. А.* Применение теории графов в программировании.— М.: Наука, 1985.
111. *Ершов А. П.* Введение в теоретическое программирование.— М.: Наука, 1977.
112. *Ершов Ю. Л., Палютин Е. А.* Математическая логика.— М.: Наука, 1979.— 320 с.
113. *Ефимов Е. И.* Решатели интеллектуальных задач.— М.: Наука, 1982.— 320 с.
114. *Загоруйко Н. Г., Скоробогатов В. А., Хворостов П. В.* Вопросы анализа и распознавания молекулярного строения на основе общих фрагментов // Алгоритм анализа структурной информации: Вычисл. системы.— Вып. 10.— Новосибирск: ИМ СО АН СССР, 1984.— С. 26—50.
115. *Заде Л.* Понятие о лингвистической переменной и ее применении к принятию приближенных решений.— М.: Мир, 1976.— 167 с.
116. *Зайченко Ю. П.* Исследование операций.— К.: Вища шк., 1988.— 552 с.
117. *Залманзон Л. А.* Преобразования Фурье, Уолша, Хаара и их применение в управлении, связи и других областях.— М.: Наука, 1989.— 496 с.
118. *Зарипов Ш. Х.* Машинный поиск вариантов при моделировании творческого процесса.— М.: Наука, 1983.— 232 с.
119. *Зелковиц М., Шоу А., Гэннон Дж.* Принципы разработки программного обеспечения.— М.: Мир, 1982.— 368 с.
120. *Земляченко В. Н., Корнеенко Н. М., Тышкевич Р. И.* Проблема изоморфизма графов: Записки науч. семинаров ЛОМИ.— Т. 118.— Л.: Наука, Ленингр. отд-ние, 1972.
121. *Змитрович А. И.* Интеллектуальные информационные системы.— Минск: ТетраСистемс, 1997.— 368 с.
122. *Зуев Е. А.* Программирование на языке TURBO PASCAL 6.0, 7.0.— М.: Радио и связь, 1993.— 384 с.
123. *Журавлев Ю. И.* Об алгебраических методах в задачах распознавания и классификации: В кн.: Распознавание, классификация, прогноз: Математические методы и их применение.— М.: Наука, 1989.— С. 9—16.
124. *Журавлев Ю. И., Камилов М. М., Туляганов Ш. Е.* Алгоритмы вычисления оценок и их применение.— Ташкент: Фан, 1974.— 38 с.
125. *Ивахненко А. Г.* Индуктивный метод самоорганизации моделей сложных систем.— К.: Наук. думка, 1982.— 296 с.

126. *Ивахненко А. Г.* Моделирование сложных систем: Информационный подход / Под общ. ред. В. В. Павлова.— К.: Вища шк., 1987.— 63 с.
127. Интеллектуализация информационных систем / Ф. И. Андон, Э. Г. Захарова, В. А. Резниченко, А. Е. Яшунин // УСиМ.— 1987.— № 6.— С. 84—92.
128. Интеллектуальные решающие машины: Проблемы создания и основные принципы / А. В. Палагин, В. Н. Коваль, З. Л. Рабинович и др. // Там же.— 1992.— № 1—2.— С. 27—34.
129. Искусственный интеллект: Справочник: В 3-х т.— М.: Радио и связь, 1990.
130. Исследование операций: В 2-х т.— М.: Мир, 1981.
131. *Калиниченко Л. А.* Методы и средства интеграции неоднородных баз данных.— М.: Наука, 1983.— 424 с.
132. *Калихман И. Л.* Сборник задач по математическому программированию.— М., 1975.— 270 с.
133. *Кандрашина Е. Ю., Литвинцева Л. В., Поспелов Д. А.* Представление знаний о времени и пространстве в интеллектуальных системах.— М.: Наука, 1989.
134. *Капитонова Ю. В., Летичевский А. А.* Методы и средства алгебраического программирования // Кибернетика и системн. анализ.— 1993.— № 3.— С. 7—12.
135. *Капитонова Ю. В., Летичевский А. А.* Об основных парадигмах программирования // Кибернетика и системн. анализ.— 1994.— № 6.— С. 3—20.
136. *Кириченко Н. Ф.* Введение в теорию стабилизации движения.— К.: Вища шк., 1978.— 184 с.
137. *Кириченко Н. Ф.* Некоторые задачи устойчивости и управляемости движения.— К.: Изд-во Киев. ун-та, 1972.— 212 с.
138. *Кириченко Н. Ф., Матвиенко В. Т.* Множественные проблемы анализа и синтеза систем управления // Кибернетика и системн. анализ.— 1996.— № 3.— С. 68—77.
139. *Кириченко Н. Ф., Матвиенко В. Т.* Оптимальный синтез структур для линейных систем управления // Проблемы управления и информатики.— 1996.— № 1—2.— С. 162—171.
140. *Клайн П.* Справочное руководство по конструированию тестов.— К.: ПАН Лтд, 1994.— 286 с.
141. *Клин М. Х.* Об аксиоматике клеточных колец: В кн.: Исследования по алгебраической теории комбинаторных объектов.— М.: ВНИИСИ, 1985.— С. 6—32.
142. *Кнут Д. Э.* Искусство программирования: В 3-х т.— М.: Вильямс, 2000.
143. *Коваленко И. Н., Гнеденко Б. В.* Теория вероятностей.— К.: Вища шк., 1990.— 328 с.
144. *Ковальски Р.* Логика в решении проблем.— М.: Наука, 1990.— 280 с.
145. *Кокорева Л. В., Перевозчикова О. Л., Юценко Е. Л.* Диалоговые системы и представление знаний.— К.: Наук. думка, 1992.— 448 с.
146. Компьютерная технология обучения: Междунар. словарь-справочник / Под ред. В. И. Гриценко, А. М. Довгялло, А. Я. Савельева: В 2-х т.— К.: Наук. думка, 1992.— 650 с.
147. *Коннолли Т., Бегг К., Страчан А.* Базы данных: Проектирование, реализация и сопровождение: Теория и практика.— М.: Вильямс, 2000.— 1120 с.
148. *Кормен Т., Лейзерсон Ч., Ривест Р.* Алгоритмы: Построение и анализ.— М.: МЦНМО, 1999.— 960 с.
149. *Корнеев В. В., Гарев А. Ф., Васютин С. В. и др.* Базы данных: Интеллект. обработка информации.— М.: Нолидж, 2000.— 352 с.
150. *Кофман А.* Введение в теорию нечетких множеств.— М.: Радио и связь, 1982.— 432 с.

151. *Кофман А., Анри-Лабордер А.* Методы и модели исследования операций.— М.: Мир, 1977.— 432 с.
152. *Краснощеков П. С., Петров А. А.* Принципы построения моделей.— М.: МГУ, 1983.— 264 с.
153. *Криницкий Н. А.* Алгоритмы и роботы.— М.: Сов. радио, 1983.
154. *Кристофидес Н.* Теория графов: Алгоритмический подход.— М.: Мир, 1978.
155. *Круглов В. В., Борисов В. В.* Искусственные нейронные сети: Теория и практика.— М.: Горячая линия-Телеком, 2001.— 382 с.
156. *Крушевский А. В.* Теория игр.— К.: Вища шк., 1977.— 216 с.
157. *Кудрявцев В. Б., Алешин С. В., Подколзин А. С.* Введение в теорию автоматов.— М.: Наука, 1985.— 320 с.
158. *Кузин Л. Т.* Основы кибернетики: В 2-х т.— М.: Энергия, 1979.
159. *Кузнецов В. Е.* Представление в ЭВМ неформальных процедур.— М.: Наука, 1989.— 160 с.
160. *Куржанский А. Б.* Управление и наблюдение в условиях неопределенности.— М.: Наука, 1977.— 392 с.
161. *Куссуль Э. М.* Ассоциативные нейроподобные структуры.— К.: Наук. думка, 1992.— 144 с.
162. *Лебедев А. Н.* Моделирование в научно-технических исследованиях.— М.: Радио и связь, 1989.— 223 с.
163. *Лингер Р., Миллс Х., Уитт Б.* Теория и практика структурного программирования.— М.: Мир, 1982.— 404 с.
164. Линейное и нелинейное программирование / И. Н. Ляшенко, Е. А. Карагодова, Н. В. Черникова, Н. З. Шор.— К.: Вища шк., 1975.— 372 с.
165. *Липский В.* Комбинаторика для программистов.— М.: Мир, 1988.— 213 с.
166. *Литвинов В., Марьянович Т.* Методы построения имитационных систем.— К.: Наук. думка, 1991.— 115 с.
167. Логический подход к искусственному интеллекту: От классической логики к логическому программированию / А. Тейз, П. Грибомон, Ж. Луи и др.— М.: Мир, 1990.— 432 с.
168. *Лорьер Ж.-Л.* Системы искусственного интеллекта.— М.: Мир, 1991.— 568 с.
169. *Любарский Ю. Я.* Интеллектуальные информационные системы.— М.: Наука, 1990.— 232 с.
170. *Майер Б., Бодуэн К.* Методы программирования: В 2-х т.— Т. 1.— М.: Мир, 1982.— 356 с.; Т. 2.— 368 с.
171. *Максименко Н. В.* Анализ алгоритмов диспетчеризации задач мультипроцессорной ЭВМ // Управляющие системы и машины.— 1978.— № 3.
172. *Малпас Дж.* Реляционный язык Пролог и его применение.— М.: Наука, 1990.— 464 с.
173. *Мальцев А. И.* Алгоритмы и рекурсивные функции.— М.: Наука, 1986.— 368 с.
174. *Марковский А. В., Шипилина Л. Б.* О машинной реализации операций со скобочными булевыми выражениями: В кн.: Алгоритмические исследования в комбинаторике / Под ред. И. А. Фараджева.— М.: Наука, 1978.— С. 172—183.
175. *Марселлус Д.* Программирование экспертных систем на Турбо-Прологе.— М.: Финансы и статистика, 1994.— 256 с.
176. *Мартиросян А. А.* О некоторых классах надежных алгоритмов индуктивного вывода // Математ. вопросы кибернетики и вычисл. техники.— Ереван, 1986.— Т. 17.— С. 13—38.
177. Математическое моделирование и эксперимент.— К.: Наук. думка, 1987.— 159 с.

178. *Мелихов А. Н., Бернштейн Л. С., Коровин С. Я.* Ситуационные советующие системы с нечеткой логикой.— М.: Наука, 1990.— 272 с.
179. *Мендельсон Э.* Введение в математическую логику.— М.: Наука, 1984.— 320 с.
180. *Миль Д. С.* Система логики силлогистической и индуктивной.— М.: Книжное дело, 1900.— 261 с.
181. *Минский М.* Фреймы для представления знаний.— М.: Энергия, 1979.— 151 с.
182. *Минский М., Пейперт С.* Персептроны.— М.: Мир, 1971.
183. *Михалевич В. С., Волкович В. Л.* Вычислительные методы исследования и проектирования сложных систем.— М.: Наука, 1982.— 286 с.
184. *Михалевич В. С., Кукса А. И.* Методы последовательной оптимизации.— М.: Наука, 1983.— 208 с.
185. *Михалевич В. С., Шкурба В. В.* Последовательные схемы оптимизации в задачах упорядочения выполнения работ // Кибернетика.— 1966.— № 2.— С. 34—40.
186. *Михалевич В. С., Шор Н. З.* Метод последовательного анализа вариантов при решении вариационных задач управления, планирования и проектирования: Докл. на IV Всесоюз. мат. съезде.— Л., 1961.— 91 с.
187. *Мичи Д., Джонстон Р.* Компьютер-творец.— М.: Мир, 1987.— 255 с.
188. Моделирование языковой деятельности в интеллектуальных системах / Под ред. А. Е. Кибрика, А. С. Нариньяни.— М.: Мир, 1987.— 280 с.
189. *Моисеев Н. Н.* Численные методы в теории оптимальных систем.— М.: Наука, 1971.— 424 с.
190. *Моисеев Н. Н.* Элементы теории оптимальных систем.— М.: Наука, 1975.— 528 с.
191. *Моисеев Н. Н., Иванилов Ю. П., Столярова Е. М.* Методы оптимизации.— М.: Наука, 1978.— 352 с.
192. *Морган М.* Java 2. Руководство разработчика.— М.: Вильямс, 2000.— 720 с.
193. *Морозов А. А., Ященко В. А.* Интеллектуализация ЭВМ на базе нового класса нейроподобных растущих сетей.— К.: Тираж, 1997.— 125 с.
194. *Мушик Э., Мюллер П.* Методы принятия технических решений.— М.: Мир, 1990.— 208 с.
195. *Мэйндоналд Дж.* Вычислительные алгоритмы в прикладной статистике.— М.: Финансы и статистика, 1988.— 350 с.
196. *Нейлор К.* Как построить свою экспертную систему.— М.: Энергоатомиздат, 1991.— 286 с.
197. *Нейман Дж., Моргенштерн О.* Теория игр и экономическое поведение.— М.: Наука, 1970.
198. Нейрокомпьютеры и интеллектуальные роботы / Э. М. Куссуль, Н. М. Амосов, Т. Н. Байдык и др.; Под общ. ред. Н. М. Амосова.— К.: Наук. думка, 1991.— 310 с.
199. Нечеткие множества и теория возможностей: Последние достижения / Под ред. Р. Ягера.— М.: Радио и связь, 1986.— 408 с.
200. *Нильсон Н.* Искусственный интеллект: Методы поиска решений.— М.: Мир, 1973.
201. *Нильсон Н.* Принципы искусственного интеллекта.— М.: Радио и связь, 1985.— 373 с.
202. *Новиков Ф. А.* Дискретная математика для программистов.— СПб.: Питер, 2000.— 304 с.
203. *Ноден П., Китте К.* Алгебраическая алгоритмика.— М.: Мир, 1999.— 720 с.

204. *Олецкий А. В.* Параметризация непрерывных диагностических сигналов на основе интегрального преобразования Карунена—Лоэва: Автореф. дисс. ... канд. техн. наук.— К., 1991.— 17 с.
205. *Олецкий А. В.* О применении интегрального разложения Карунена—Лоэва при моделировании динамических систем // УСиМ.— 1999.— № 2.— С. 12—15.
206. *Олецький О. В.* Про застосування інтегрального розкладу Карунена—Лоева для вирішення інтегральних рівнянь Фредгольма першого роду: У зб. наук. праць Ін-ту проблем моделювання в енергетиці.— Вип. 9.— К., 2000.— С. 35—40.
207. *Омельченко В. А.* Основы спектральной теории распознавания сигналов.— Харьков: Вища шк., 1983.— 156 с.
208. *Отнес Р., Эноксон Л.* Прикладной анализ временных рядов.— М.: Мир, 1982.— 428 с.
209. *Павлидис Т.* Алгоритмы машинной графики и обработки изображений.— М.: Радио и связь, 1986.— 400 с.
210. *Петренюк А. Я.* Применение инвариантов в комбинаторных исследованиях // Вопросы кибернетики.— М.: Изд-во АН СССР, 1973.— С. 129—135.
211. *Петросян Л. А., Зенкевич Н. А., Семина Е. А.* Теория игр.— М.: Высш. шк., 1998.— 304 с.
212. *Петрушин В. А.* Экспертно-обучающие системы.— К.: Наук. думка, 1992.— 196 с.
213. *Питерсон Дж.* Теория сетей Петри и моделирование систем.— М.: Мир, 1984.
214. *Понтрягин Л. С., Болтянский В. Г., Гамкрелидзе Р. В., Мищенко Е. Ф.* Математическая теория оптимальных процессов.— М.: Наука, 1983.— 392 с.
215. *Попов Э. В.* Общение с ЭВМ на естественном языке.— М.: Наука, 1982.— 360 с.
216. *Попов Э. В.* Экспертные системы. Решение неформализованных задач в диалоге с ЭВМ.— М.: Наука, 1987.— 288 с.
217. *Попов Э. В., Юдин Д. Б., Гольштейн Е. Г.* Линейное программирование.— М.: Наука, 1969.
218. *Поспелов Д. А.* Введение в теорию вычислительных систем.— М.: Сов. радио, 1972.— 280 с.
219. *Поспелов Д. А.* Логико-лингвистические методы в системах управления.— М.: Энергоатомиздат, 1981.— 190 с.
220. *Поспелов Д. А.* Моделирование рассуждений.— М.: Радио и связь, 1989.— 184 с.
221. *Поспелов Д. А.* Ситуационное управление: Теория и практика.— М.: Наука, 1986.— 288 с.
222. *Поспелов Д. А.* Фантазия или наука: На пути к искусственному интеллекту.— М.: Наука, 1982.— 224 с.
223. Представление и использование знаний / Под ред. Х. Уэно, М. Исидзука.— М.: Мир, 1989.— 220 с.
224. Прикладная статистика: Классификация и снижение размерности: Справ. изд. / С. А. Айвазян, В. М. Бухштабер, И. С. Енюков, Л. Д. Мешалкин.— М.: Финансы и статистика, 1989.— 607 с.
225. Прикладные нечеткие системы / Под ред. Т. Тэрано, Д. Ватада, С. Иваи.— М.: Мир, 1993.— 368 с.
226. Приобретение знаний / Под ред. С. Осуги, Ю. Саэки.— М.: Мир, 1990.— 304 с.
227. *Продеус А. Н., Захрабова Е. Н.* Экспертные системы в медицине.— К.: ВЕК+, 1998.— 320 с.
228. *Пропп В. Я.* Морфология сказки.— М.: Наука, 1969.
229. *Проценко В. С., Чаленко П. И., Сорока Р. А.* Техника программирования.— К.: Наук. думка, 1990.— 183 с.

230. *Пытьев Ю. П.* Математические методы интерпретации эксперимента.— М.: Высш. шк., 1989.— 351 с.
231. *Пшеничный Б. Н., Данилин Ю. М.* Численные методы в экстремальных задачах.— М.: Наука, 1975.
232. *Рабинович З. Л.* О концепции машинного интеллекта и ее развитии // Кибернетика и системн. анализ.— 1995.— № 2.— С. 163—173.
233. *Расстригин Л. А.* Современные принципы управления сложными объектами.— М.: Сов. радио, 1980.— 232 с.
234. *Расстригин Л. А.* Адаптивное обучение с моделью обучаемого.— Рига: Зинатне, 1988.— 160 с.
235. Реальность и прогнозы искусственного интеллекта: Сб. статей.— М.: Мир, 1987.— 247 с.
236. *Рейнгольд Э., Нивергельт Ю., Део Н.* Комбинаторные алгоритмы: Теория и практика. М.: Мир, 1980.
237. *Розенблатт Ф.* Принципы нейродинамики.— М.: Мир, 1966.
238. *Сергиенко И. В.* Математические модели и методы решения задач дискретной оптимизации.— К.: Наук. думка, 1988.— 472 с.
239. *Сергієнко І. В.* Становлення і розвиток досліджень з інформатики.— К.: Наук. думка, 1998.— 206 с.
240. *Сергиенко И. В., Гупал А. М., Пашко С. В.* О сложности задач распознавания образов // Кибернетика и системн. анализ.— 1996.— № 4.— С. 70—88.
241. *Сергиенко И. В., Каспшицкая М. Ф.* Модели и методы решения на ЭВМ комбинаторных задач оптимизации.— К.: Наук. думка, 1981.— 288 с.
242. *Скурихин В. И., Шифрин В. Б., Дубровский В. В.* Математическое моделирование.— М.: Техника, 1983.— 270 с.
243. Словарь по кибернетике / Под ред. В. С. Михалевича.— К.: Гл. ред. УСЭ, 1989.— 751 с.
244. *Слэйгл Дж.* Искусственный интеллект.— М.: Мир, 1973.— 320 с.
245. *Смаллиан Р.* Принцесса или тигр? — М.: Мир, 1985.— 221 с.
246. Справочник по прикладной статистике: В 2-х т. / Под ред. Э. Ллойда, У. Ледермана, Ю. Тюрина.— М.: Финансы и статистика, 1989.— 510 с.
247. *Солсо Р. Л.* Когнитивная психология.— М.: Тривола, 1996.— 600 с.
248. *Ставровський А. Б., Коваль Ю. В.* Елементи програмування.— К.: Фонд "Молоді надії України", 1997.— 210 с.
249. Теория расписаний и вычислительные машины / Под ред. О. Г. Коффмана.— М.: Наука, Гл. ред. физ.-мат. лит-ры, 1984.
250. *Тихонов А. Н., Арсенин В. Я.* Методы решения некорректных задач.— М.: Наука, 1986.— 288 с.
251. Толковый словарь по искусственному интеллекту / Авт.- сост. А. Н. Аверкин, М. Г. Гаазе-Рапопорт, Д. А. Поспелов.— М.: Радио и связь, 1992.— 256 с.
252. *Ту Дж., Гонсалес Р.* Принципы распознавания образов.— М.: Мир, 1978.
253. *Тыугу Э. Х.* Концептуальное программирование.— М.: Наука, 1984.— 256 с.
254. *Уинстон П.* Искусственный интеллект.— М.: Мир, 1980.
255. *Уоссермен Ф.* Нейрокомпьютерная техника: Теория и практика.— М.: Мир, 1992.
256. *Урясьев С. П.* Адаптивные методы теории игр и стохастического программирования.— М.: Наука, 1990.— 184 с.
257. *Фараджев И. А.* Конструктивное перечисление комбинаторных объектов // Алгоритмические исследования в комбинаторике.— М.: Наука, 1978.— С. 3—11.

258. Философский словарь / Под ред. И. Т. Фролова.— М.: Политиздат, 1981.— 445 с.
259. *Фомин Я. А., Тарловский Г. Р.* Статистическая теория распознавания образов.— М.: Радио и связь, 1986.— 264 с.
260. *Фор А.* Восприятие и распознавание образов.— М.: Машиностроение, 1989.— 272 с.
261. *Фу К.* Последовательные методы в распознавании образов и обучении машин.— М.: Мир, 1971.— 256 с.
262. *Фу К.* Структурные методы в распознавании образов.— М.: Мир, 1977.
263. *Фукунага К.* Введение в статистическую теорию распознавания образов.— М.: Наука, 1979.— 367 с.
264. *Хант Э.* Искусственный интеллект.— М.: Мир, 1978.
265. *Харари Ф.* Теория графов.— М.: Мир, 1973.
266. *Хейес-Рот Ф., Уотермен Д., Ленат Д.* Построение экспертных систем.— М.: Мир, 1987.— 441 с.
267. *Хювёнен Э., Сеппянен Й.* Мир Лиспа: В 2-х т.— М.: Мир, 1990.
268. *Цаленко М. Ш.* Моделирование семантики в базах данных.— М.: Наука, 1989.— 288 с.
269. *Цейтлин Г. Е.* Введение в алгоритмику.— К.: Сфера, 1998.— 310 с.
270. *Цикритзис Д., Лоховски Ф.* Модели данных.— М.: Финансы и статистика, 1985.— 344 с.
271. *Цыпкин Я. З.* Основы теории обучающихся систем.— М.: Наука, 1970.— 252 с.
272. *Чень Ч., Ли Р.* Математическая логика и автоматическое доказательство теорем.— М.: Наука, 1983.— 358 с.
273. *Чери С., Готлоб Г., Танка Л.* Логическое программирование и базы данных.— М.: Мир, 1992.— 352 с.
274. *Шабхазян К. В., Тушкина Т. А., Сохранская В. С.* Статистические испытания различных методов диспетчеризации многопроцессорных систем // Программирование.— 1976.— № 4.— С. 91—100.
275. *Шевченко А. И.* Интеллектуальная система нового поколения // Искусственный интеллект.— 1997.— № 1.— С. 15—19.
276. *Шенк Р.* Обработка концептуальной информации.— М.: Энергия, 1980.— 360 с.
277. *Ширяев А. Н.* Вероятность.— М.: Наука, 1989.— 640 с.
278. *Шлеер С., Меллор С.* Объектно-ориентированный анализ: Моделирование мира в состояниях.— К.: Диалектика, 1993.— 240 с.
279. *Штерн І. Б.* Вибрані топіки та лексикон сучасної лінгвістики: Енцикл. словник.— К.: АртЕк, 1998.— 336 с.
280. Экспертные системы: Принципы работы и примеры / Под ред. Р. Форсайта.— М.: Радио и связь, 1987.— 224 с.
281. *Элти Дж., Кумбс М.* Экспертные системы: Концепции и примеры.— М.: Финансы и статистика, 1987.— 191 с.
282. *Эндрю А.* Искусственный интеллект.— М.: Мир, 1985.— 264 с.
283. *Эшби У.* Введение в кибернетику.— К., 1959.
284. *Юдин Д. Б., Гольштейн Е. Г.* Линейное программирование.— М.: Наука, 1969.
285. *Ямпольський Л. С., Лавров О. А.* Штучний інтелект у плануванні та управлінні виробництвом.— К.: Вища шк., 1995.— 255 с.
286. *Янов Ю. И.* О логических схемах алгоритмов: Проблемы кибернетики.— Вып. 1.— С. 75—127.
287. *Bell A.* Games Playing With Computers.— London: Allen & Unwin, 1972.
288. *Bellman R. E.* On a routing problem. Quart. Appl. Math. 1958, 16, p. 87—90.

289. *Bentley J. L., Haken D., Saxe J. B.* A general method for solving divide-and-conquer recurrences. SIGACT News 12 (3): 36—44, 1980.
290. *Bitner J. R., Reingold E. M.* Backtrack programming techniques, c. ACM. vol. 18, (1975), 651—656.
291. *Borodin A. V.* Fast modular transforms via division, IEEE 13th Annual Symposium on Switching and Automata Theory, 90—96, 1972.
292. *Bratko I.* Prolog programming for artificial intelligence.— Pearson Education Limited, 2001.
293. *Cook C., Rosenfeld A., Aronson A.* Grammatical inference by hill-climbing // Information Science.— 1976.— N 10.— P. 59—80.
294. *Edmonds J.* Matroids and the greedy algorithms // Math. Programming.— 1971.— N 1.— P. 127—136.
295. *Floyd R. W.* Algorithm 97: Shortest path. Comm. ACM, 1962.— 701 p.
296. *Ford L. R.* Network flow theory. The Rand Corp., P-923, August 1956.
297. *Ford L. R., Fulkerson D. R.* Flows in networks.— Prinston Univ. Press, Princeton, N. J., 1962.
298. *Garfinkel R. and Nemhauser G.* Integer Programming.— John Wiley, 1973.
299. *Gold E. M.* Language identification in the limit // Information and Control.— 1967.— N 10.— P. 447—474.
300. *Golomb S. and Baumen L.* Backtrack programming J. ACM, vol. 12, (1965), 516—524.
301. *Hall M. and Knuth D. E.* Combinatorial analysis and computers // American Mathematical Monthly, vol. 72, Part II, Feb. 1965, 21—28.
302. *Haykin S.* Neural networks: A comprehensive foundation.— N. Y.: Macmillan college publishing company, 1994.
303. *Holland J. H.* Adaptation in natural and artificial systems.— The University of Michigan Press, 1975.
304. *Horowitz E., Sahni S.* Fundamentals of computer algorithms.— Computer Science Press, Inc.: USA, 1978.— 625 p.
305. *Jackson P.* Introduction to expert systems.— Addison Wesley Longman Publishing, 1999.
306. *Johnson D.* Efficient algorithms for shortest paths in sparse networks.— J. ACM, 1977, 24, p. 1—13.
307. *Johnson S. M.* Generation of permutations by adjacent transpositions.— Math. Comp., 1963, 17, p. 282—285.
308. *Knuth D. E.* Estimating the efficiency of backtrack programs // Mathematics of Computation, vol. 29. (1975), p. 121—136.
309. *Kruskal J. B.* On the shortest spanning subgraph of a graph and the traveling salesman problem // Proc. Amer. Math. Soc., 1956, 7, p. 48—49.
310. *Lawler E. L.* Combinatorial Optimization: Networks and Matroids.— New York, Holt, Rinehart and Winston, 1976.
311. *Lenat D.* AM: an artificial intelligence approach to discovery in mathematics as heuristic search // Comp. Science Dept. Rep. STAN-CS-76-570. Stanford Univ.— Stanford, Calif., 1976.
312. *Lenat D.* Automated theorem formation in mathematics // Contemporary math., 1984.— N 29.— P. 287—314.
313. *Lipson J.* Chinese remainder and interpolation algorithms, Proc. 2nd Symposium in Symbolic and Algebraic Manipulation, 372—391, 1971.
314. *Liu C. L.* Introduction to combinatorial mathematics.— McGraw-Hill, 1968.
315. *Malhotra V. M., Kumar P. M., Mahtshwari S. N.* An O ($|V|^3$) algorithm for finding maximum flows in network. Information Processing Lett., 1978, 7, p. 277—278.

316. *McCulloch W. S., Pitts W.* A logical calculus of the imminent in nervous activity // Bull. Math. Biophysics, 1943, 5, p. 115—137.
317. *Minsky M.* Steps toward artificial intelligence, p. 406—450.
318. *Minsky M., Selfridge O. G.* Learning in Random nets. In: Information Theory.— London Butterworth, 1961.— P. 335—347.
319. *Nijenhuis A. and Wilf H. S.* Combinatoral Algorithms.— Academic Press, New York, 1975.
320. *Nillson N.* Artificial Intelligence. A New Synthesis.— San Francisco: Morgan Kaufmann Publishers, CA, 1998.— 514 p.
321. *Nwana H.* Software agents. An overview // The Knowledge Engineering Review, vol. 11 (3), 1996.— P. 205—244.
322. *Purdom P. W. Jr. and Brown C. A.* The analysis of algorithms.— Holt, Rinehart and Winston, 1985.
323. *Rado R.* Note on independence function // Proc. London Math. Soc.— 1957, 7, p. 337—343.
324. *Shapiro E.* Inductive inference of theories from facts // Tech. Rep. 192 Dept. Comput. Scien., Yale Univ.— New Haven, Conn., 1981.
325. *Sowa J.* Conceptual Structures: Information Processing in Mind and Machine.— Addison Wesley, 1984.
326. *Tarjan R. E.* Depth first search and linear graph algorithms // SIAM J. Comput., 1972, 1, p. 146—160.
327. *Walker R. J.* An enumerative technique for a class of combinatorial problems // Proceedings of Symposia in Applied Mathematics, vol. X, American Mathematical Society, Providence, R. I., 1960.
328. *Warshall S.* A theorem on Boolean matrices // J. ACM, 1962, 9, p. 11—12.
329. *Welsh D. J. A.* Matroid Theory.— London, Academic Press, 1976.
330. *Whitney H.* On the abstract properties of linear independence // Amer. J. Math., 1935, 57, p. 509—533.
331. *Wooldridge M., Jennings N.* Intelligent agents: theory and practice // The Knowledge Engineering Review, vol. 10 (2), 1995, p. 115—152.

ПРЕДМЕТНИЙ ПОКАЖЧИК

Р

С

Т

ЗМІСТ

ЧАСТИНА II

МОДЕЛІ ПОДАННЯ ЗНАНЬ І МЕТОДИ ЛОГІЧНОГО ВИВЕДЕННЯ

ЧАСТИНА VI

РОЗПІЗНАВАННЯ ОБРАЗІВ

ЧАСТИНА VII

НЕЙРОННІ МЕРЕЖІ

ЧАСТИНА VIII

РОЗУМІННЯ І СПІЛКУВАННЯ

Навчальне видання

М. М. Глибовець, О. В. Олецький

Штучний інтелект

Підручник

Редактор *Т. О. Заремба*
Художнє оформлення *Н. В. Михайличенко*
Комп'ютерна верстка *В. Ю. Романенка*
Коректори *Н. І. Слєсаренко, І. Г. Ярошенко*

Підписано до друку 24.01.2002 р. Формат 60×90 1/16.
Папір офсетний. Друк офсетний. Гарнітура «Тип Таймс».
Умов. друк. арк. 23,00. Обл.-вид. арк. 23,00.
Наклад 1000 прим. Зам. № 2—17.

Видавничий дім "КМ Академія".
Друкарня НаУКМА.

Адреса видавництва та друкарні:
04070, Київ-70, вул. Сковороди, 2.
тел./факс (044) 416-60-92, 238-28-26.

Глибовець М. М., Олецький О. В.

Г54 Штучний інтелект : Підруч. для студ. вищ. навч. закладів, що навчаються за спец. "Комп'ютер. науки" та "Приклад. математика".— К.: Вид. дім "КМ Академія", 2002.— 366 с.: іл.— Бібліогр.: с. 339—351.

ISBN 966-518-153-X

Метою підручника є ознайомлення студентів з основами теорії штучного інтелекту. Розглядаються основні методи роботи зі знаннями, зокрема з нечіткими та недостовірними; планування цілеспрямованих дій та прийняття рішень; ігрові задачі; розпізнавання образів та нейронні мережі.

ББК 32.813я73

ДОВІДКА ПРО АВТОРІВ

Микола Глибовець *(офісна адреса: університет "Києво-Могилянська академія", Київ 254070, вул. Сковороди, 2, тел. (044) 463—6985, e-mail: glib@ukma. kiev. ua) — доцент, кандидат фізико-математичних наук, декан факультету інформатики університету "Києво-Могилянська академія". Базова освіта — Київський державний університет ім. Т. Шевченка (факультет кібернетики, рік закінчення — 1979), спеціалізація — прикладна математика. Закінчив аспірантуру і захистив дисертаційну роботу під керівництвом професора А. В. Анісімова. Вивчав кращі світові стандарти науки та освіти за кордоном (1996 — стажування в Бостонському університеті США; 1989—90 стажування в Тамперському технологічному університеті, Фінляндія). Має досвід виконання спільних міжнародних проектів, наприклад, 1999—2001 — робота за грантом проекту TEMPUS TACIS Joint European Project № T_JEP-10809-1999 "Human Resource Development for Publishing in Ukraine (електронні видання)" з університетом Оксфорд Брукс, Великобританія та Вищою школою техніки, мистецтва та культури, Лейпциг, Німеччина. Наукові інтереси — алгоритми на графах, дослідження паралельних систем обробки інформації, інтелектуальні бази знань, розподілені обчислення, інструментальні засоби створення систем дистанційної освіти. Автор понад 40 праць, серед яких 6 навчальних посібників. Має публікації в міжнародних виданнях. Підготував до друку монографію (Основи комп'ютерних алгоритмів, сайт факультету інформатики НаУКМА, 2000, 498 с.). Сфера застосування останніх наукових результатів — системи дистанційної освіти.*

Найкращий відпочинок — праця в селі у батьків, рибальство та полювання за грибами. Йому також до вподоби фантастика, шахи, футбол, українська пісня.

Олексій Олецький *народився в 1962 р. Доцент факультету інформатики Національного університету "Києво-Могилянська академія". У 1984 р. закінчив факультет кібернетики Київського університету ім. Т. Шевченка. Кандидат технічних наук з 1992 року.*

Брав участь у написанні навчальних посібників "Системи штучного інтелекту", "Програмування в Пролозі", "Основні поняття та закони екології" (з орієнтацією на застосування комп'ютерних засобів навчання). Розробив навчальні курси "Основи проектування систем штучного інтелекту", "Принципи роботи комп'ютерних систем", "Програмне забезпечення моделюючих систем".

Брав участь у проектах, які стосувалися розробки алгоритмів і програм стиску даних, відновлення сигналів і зображень, комп'ютерного моделювання, створення засобів комп'ютерного навчання, електронних публікацій в Інтернет.

Автор понад 45 наукових праць, а також аналітичних оглядів для газет "Computer World/Киев" та "Бизнес".

Хобі — туризм, фотографія.