МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
МИКОЛАЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ
ІМЕНІ В. О. СУХОМЛИНСЬКОГО

О. С. Булгакова
В. В. Зосімов
В. О. Поздєєв

МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ: ТЕОРІЯ ТА ПРАКТИКА

Навчальний посібник

ОЛДІ ПЛЮС
2020

УДК 004.89(075)
Б90

Рекомендовано до друку Вченою радою
Миколаївського національного університету імені В. О. Сухомлинського
(протокол № 18 від 27.01.2020 року)

Рецензенти:

В. В. Осипенко – доктор технічних наук, професор кафедри комп'ютерної інженерії та електромеханіки Київського національного університету технологій та дизайну;

В. І. Литвиненко – доктор технічних наук, професор, завідувач кафедри інформатики і комп'ютерних наук Херсонського національного технічного університету;

І. П. Атаманюк – доктор технічних наук, професор, завідувач кафедри вищої та прикладної математики Миколаївського національного аграрного університету

Булгакова О. С., Зосімов В. В., Поздєєв В. О.

Б90 Методи та системи штучного інтелекту: теорія та практика: навчальний посібник / О. С. Булгакова, В. В. Зосімов, В. О. Поздєєв. – Херсон : «ОЛДІ-ПЛЮС», 2020. – 356 с.

ISBN 978-966-289-364-9

Навчальний посібник є складовою частиною курсу «Методи та системи штучного інтелекту» для студентів спеціальностей галузі знань 12 Інформаційні технології та спеціальності 113 Прикладна математика, який орієнтований на студентів підготовки ступеня бакалавр. В навчальному посібнику розглянуто моделі і методи, які застосовуються в системах штучного інтелекту. Значну увагу приділено питанням проектування систем штучного інтелекту, розгляду основних напрямів розвитку інтелектуальних систем. Всі головні положення підручника розглядаються з використанням навчальних прикладів і ілюструються графічними матеріалами.

В навчальному посібнику викладено основний курс лекцій, практичні заняття з курсу, лабораторні роботи та завдання для самостійної роботи.

УДК 004.89(075)

ISBN 978-966-289-364-9

ЗМІСТ

ПЕРЕЛІК УМОВНИХ ПОЗНАЧЕНЬ

ШІ	Штучний інтелект
МНК	Метод найменших квадратів
НМ	Нейронна мережа
ФС	формальна система
ЕС	Експертна система
МГУА	Метод групового урахування аргументів
БІА	Багаторядний ітераційний алгоритм
УІА	Узагальнений ітераційний алгоритм
SVD	Singular Value Decomposition
ПНМ	Поліноміальна Нейронна Мережа
ГА	Генетичний алгоритм
CHC	Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation
HUX	Half Uniform Crossover
ІІС	Інтелектуальні інформаційні системи
БЗ	База знань

ВСТУП

У навчальному посібнику «Методи та системи штучного інтелекту: теорія та практика» відображені базові знання з дисципліни «Методи та системи штучного інтелекту», обумовлені змістовими модулями Державних освітніх стандартів. Ці модулі встановлюють основні напрями і аспекти розгляду предмету, а також послідовність викладення навчального матеріалу.

Зміст навчального посібника відображає систему науково-предметних знань, необхідних для опанування професією і використання їх у професійній ІТ-діяльності.

Навчальний посібник містить курс лекцій, практичних занять та завдань для самостійної роботи. У посібнику включені необхідні матеріали для самостійного вивчення теорії інтелектуальних інформаційних систем, тести для контролю правильного засвоєння матеріалу, література для глибшого вивчення кожної теми.

В посібнику розглядаються основні завдання досліджень в області штучного інтелекту, методи та моделі представлення знань в комп'ютерних системах. Викладаються основи логіки висловлювань і методи вирішення завдань в даній області знань, наводяться приклади розв'язання задач. Розглядаються принципи побудови нейромережевих систем, систем індуктивного моделювання, як окремого типу нейромережі. Окремий розділ присвячений системам Fuzzy logic та еволюційному моделюванню.

Всі головні положення навчального посібника розглядаються з використанням навчальних прикладів і ілюструються графічними матеріалами. Матеріал носить навчально-методичний характер і сприяє формуванню знань і практичних навичок в галузі інтелектуальних інформаційних систем.

Навчальний посібник може бути корисним для студентів комп'ютерних спеціальностей вищих навчальних закладів, а також може використовуватися студентами спеціальності «Прикладна математика», аспірантами, науковими та педагогічними працівниками, практичними фахівцями.

РОЗДІЛ І. ОСНОВНІ НАПРЯМКИ ДОСЛІДЖЕНЬ В ОБЛАСТІ ШТУЧНОГО ІНТЕЛЕКТУ

1.1. Вступ до штучного інтелекту: визначення, історія, завдання

Термін «штучний інтелект» (artificial intelligence) був запропонований у 1956 році. Слово intelligence означає «вміння мислити мудро», а зовсім не «інтелект», для якого є термін intellect.

Штучний інтелект (ШІ) займається вивченням розумної поведінки (у людей, тварин і машин) і намагається знайти способи моделювання подібної поведінки в будь якому типі штучно створеного механізму. Незважаючи на те що терміну більше півстоліття, єдиного визначення його не існує. Різні вчені по-різному визначають цю науку, залежно від свого погляду на неї, і працюють над створенням систем, котрі:

– думають подібно людям;
– думають раціонально;
– діють подібно людям;
– діють раціонально.

Аналізуючи десятки визначень штучного інтелекту з різних джерел, в якості робочого визначення можна запропонувати наступне.

Штучний інтелект – це один з напрямків інформатики, метою якого є розробка апаратно-програмних засобів, що дозволяють користувачеві-непрограмістові ставити і вирішувати свої, що традиційно вважаються інтелектуальними, завдання, спілкуючись з ЕОМ на обмеженій підмножині природної мови.

При відтворенні міркувань і дій виникають певні труднощі. По-перше, в більшості випадків, виконуючи якісь дії, людина не усвідомлює, як це вона робить, не відомий точний спосіб, метод або алгоритм розуміння тексту, розпізнавання осіб, доказу теорем, рішення завдань, твору віршів і т. п. По-друге, на сучасному рівні розвитку комп'ютер занадто далекий від людського рівня компетентності і працює по інших принципах.

Штучний інтелект завжди був міждисциплінарною наукою, являючись одночасно і наукою і мистецтвом, і технікою і психологією. Методи

штучного інтелекту різноманітні. Вони активно переплітаються з іншими науками, адаптуються і змінюються під задачу, яка вирішується.

Історія розвитку штучного інтелекту

Ідея створення штучної подібності людини для вирішення складних завдань і моделювання людського розуму була ще в прадавні часи. Так, в Стародавньому Єгипті була створена «оживаюча» механічна статуя Бога Амона. У Гомера в «Іліаді» Бог Гефест кував людиноподібних істот.

Штучний інтелект є в деякому розумінні наукою майбутнього, в якому немає жорсткого поділу по областях і ясно видно зв'язок між окремими дисциплінами, які лише відбивають деяку грань пізнання.

Точне зведення законів, керуючих раціональною частиною мислення, було сформульоване Аристотелем (384–322 рр. до н. е.). Проте родоначальником штучного інтелекту вважається середньовічний іспанський філософ, математик та поет Раймунд Луллій, який ще в XIII столітті спробував створити механічну машину для вирішення різних завдань на основі розробленої їм загальної класифікації понять. У XVIII столітті Лейбніц і Декарт незалежно один від одного продовжили цю ідею, запропонувавши універсальні мови класифікації усіх наук. Праці цих вчених можна вважати першими теоретичними працями в області штучного інтелекту. Теорія ігор та теорія прийняття рішень, дані про будову мозку, когнітивна психологія - усе це стало будівельним матеріалом для штучного інтелекту. Але остаточне народження штучного інтелекту як наукового напряму сталося тільки після створення ЕОМ в 40-х роках XX століття та випуску Норбертом Вінером робіт по новій науці – кібернетиці.

Формування штучного інтелекту як науки відбулось у 1956 році. Д. Маккарті, М. Мінський, К. Шенон та Н. Рочестер організували двомісячний семінар в Дартмуті для американських дослідників, що займаються теорією автоматів, нейронними мережами, інтелектом. Хоча дослідження в цій області вже активно велися, але саме на цьому семінарі з'явилися термін і окрема наука – штучний інтелект.

Одним із засновників теорії штучного інтелекту вважається відомий англійський вчений Алан Тюрінг, який в 1950 році опублікував статтю «Обчислювальні машини і розум» (перекладену російською мовою під назвою «Чи може машина мислити?»). Саме у ній описувався

«тест Тюрінга», що став класичним, дозволяє оцінити «інтелектуальність» комп'ютера по його здатності до осмисленого діалогу з людиною.

У 1954 році А. Ньюелл, Дж. Шоу та Г. Саймон створили програму для гри в шахмати на основі методу, запропонованого в 1950 році К. Шеноном, формалізованого А. Тюрінгом і промодельованого їм же вручну. До роботи була залучена група голландських психологів під керівництвом Адріана Де Гроота, що вивчали стилі гри видатних шахістів. У 1956 році цим колективом була створена мова програмування ІПЛ1 – практично перша символьна мова обробки списків та написана перша програма «Логік-теоретик», призначена для автоматичного доказу теорем. Цю програму можна віднести до перших досягнень в області штучного інтелекту.

В 1960 році цією ж групою була написана програма GPS (General Problem Solver) – універсальний вирішувач задач. Вона могла вирішувати ряд головоломок, обчислювати невизначені інтеграли, вирішувати деякі інші завдання. Результати привернули увагу фахівців в області обчислень, і з'явилися програми автоматичного доказу теорем з планіметрії та розв'язку задач алгебри.

У 1958 році Д. Маккарті визначив нову мову високого рівня Lisp, яка стала домінуючою для штучного інтелекту.

Перші нейромережі з'явилися у кінці 50-х років XX століття. У 1957 році Ф. Розенблатом була зроблена спроба створити систему, яка моделювала людське око та його взаємодію з мозком – персептрон.

Перша міжнародна конференція по штучному інтелекту (IJCAI) відбулась в 1969 році у Вашингтоні.

У 1963 році Д. Робінсон реалізував метод автоматичного доказу теорем, який отримав назву «Принцип резолюції», та на основі цього методу в 1973 році була створена мова логічного програмування Prolog.

У США з'явилися перші комерційні системи, що ґрунтувалися на знаннях, – експертні системи. Відбувається комерціалізація штучного інтелекту. Ростуть щорічні капіталовкладення та інтерес до самонавчальних систем, створюються промислові експертні системи. Розробляються методи представлення знань.

Перша експертна система була створена Э. Фейгенбаумом в 1965 році. Але до комерційного прибутку було ще далеко. Лише у 1986 році перша комерційна система R1 компанії DEC дозволила заощадити приблизно

40 мільйонів доларів за рік. До 1988 року компанією DEC були розроблені 40 експертних систем. У компанії Du Pont застосовувалося 100 систем, та економія складала приблизно 10 мільйонів в рік.

У 1981 році Японія, у рамках 10-річного плану по розробці інтелектуальних комп'ютерів на базі Prolog, приступила до розробки комп'ютера 5-го покоління, заснованого на знаннях. 1986 рік став роком відродження інтересу до нейронних мереж.

У 1991 році Японія припиняє фінансування проекту комп'ютера 5-го покоління та починає проект створення комп'ютера 6-го покоління – нейрокомп'ютера.

У 1997 році комп'ютер «Deep Blue» переміг в шахи чемпіона світу Г. Каспарова, що підтвердило можливість того, що штучний інтелект може порівнятися з людиною або перевершити її у ряді інтелектуальних задач (нехай і в обмежених умовах).

Величезну роль у боротьбі за визнання штучного інтелекту зіграли вітчизняні академіки А. И. Берг та Г. С. Поспелов.

У 1954–1964 рр. створюються окремі програми та проводяться винаходи в зоні пошуку рішення логічних задач. Створюється програма «АЛПЕВ ЛОМИ» (російською), що автоматично доводить теореми. Вона основана на оригінальному зворотному методу Маслова, аналогічному методу резолюцій Робінсона. Серед найбільш значимих результатів, отриманих вітчизняними вченими в 60-і роки, слід зазначити алгоритм «Кора» М. М. Бонгарда, що моделює діяльність людського мозку при розпізнаванні образів.

З 1965–1980 рр. сталося народження нового напряму – ситуаційного управління (у західній термінології відповідає представленню знань). Засновником цієї наукової школи став професор Д. А. Поспелов.

У Московському державному університеті в 1968 році В. Ф. Турчиним була створена мова символьної обробки даних РЕФАЛ.

Завдання штучного інтелекту

Штучний інтелект переслідує безліч цілей. Одним з основних завдань штучного інтелекту є створення повного наукового опису інтелекту людини, тварини та машини і обчислення принципів, загальних для усіх трьох. Моделювання розуму потрібне для вирішення задач. До інтелектуальних задач можна віднести усі задачі, алгоритм знаходження

яких невідомий. Але, наприклад, перебір усіх можливих комбінацій також є алгоритмом. Застосувати його на практиці, на жаль, на сучасному рівні розвитку техніки до більшості задач неможливо (сучасна ЕОМ не може згенерувати усі прості перестановки більш ніж 12-ти різних предметів, оскільки цих перестановок більше 479 млн).

Комбінаторний вибух, з яким зіткнулися дослідники вже в ранніх дослідженнях, – приклад цього. У таких випадках, коли незначне збільшення вхідних даних задачі веде до зростання кількості дій, що повторюються, в статичній залежності, говорять про неполіноміальні алгоритми, які характеризуються тим, що кількість операцій в них зростає залежно від числа входів згідно із законом, близькому до експоненти.

Ефективний алгоритм має не настільки різко зростаючу залежність кількості обчислень від вхідних даних, наприклад обмежено поліноміально, тобто x знаходиться в основі, а не у показнику ступеня. Такі алгоритми називаються поліноміальними, і, як правило, якщо задача має поліноміальний алгоритм, то вона може бути вирішена на ЕОМ з великою ефективністю. До таких можна віднести задачі сортування даних, багато задач математичного програмування і т. п.

Отже, сучасний комп'ютер не може виконати рішення повністю аналітично. Можлива заміна аналітичного рішення чисельним алгоритмом, який ітеративно (тобто циклічно) або рекурсивно (викликаючи процедуру розрахунку самої себе) виконує операції, крок за кроком наближаючись до розв'язку. Якщо число цих операцій зростає, час виконання, а можливо, і витрати інших ресурсів (наприклад, обмежених машинною пам'яттю), також зростає. Задачі, в яких алгоритми рішення створюють передумови для різкого зростання використання ресурсів, у загальному вигляді не можуть бути вирішені на цифрових обчислювальних машинах, так як ресурси завжди обмежені.

Рішенням подібних задач і займається штучний інтелект. Дослідники вивчають процеси мислення, розумну поведінку для того, щоб знайти методи розв'язку подібних задач, так як людина в своїй діяльності стикається з ними досить часто і успішно вирішує.

Хоча досі багато задач не вирішені, певні досягнення в цій області є. Дослідники використовували різні підходи та методи, щоб отримати результат. В кінці 50-х років XX століття була представлена модель лабіринтового пошуку та з'явилася теорія розпізнавання образів як

наслідок початку використання ЕОМ для розв'язку необчислювальних завдань. Початок 60-х років називають епохою евристичного програмування, коли використовувалися стратегії дій на основі відомих, заздалегідь заданих евристик. Евристики дозволяють скорочувати кількість розглянутих варіантів. У середині 60-х років до вирішення задач стали активно підключати методи математичної логіки. З середини 70-х років дослідники стали приділяти увагу системам, основаним на експертних знаннях.

Такі системи застосовуються до слабоформалізованих завдань. Неформалізовані задачі зазвичай володіють наступними особливостями:

– помилковістю, неоднозначністю, неповнотою та протиріччям вихідних даних;

– помилковістю, неоднозначністю, неповнотою і суперечливістю знань про проблемну область та задачі, що вирішується;

– великою розмірністю простору рішення (тобто перебір при пошуку рішення дуже великий);

– динамічно мінливими даними та знаннями.

1.2. Дослідження в області штучного інтелекту

Основні підходи до дослідження штучного інтелекту

Незабаром після визнання штучного інтелекту окремою галуззю науки відбувся поділ його на два напрямки: нейрокібернетика і кібернетика чорного ящика. Ці напрямки розвиваються практично незалежно один від одного, істотно розрізняючись як в методології, так і в технології.

Нейрокібернетики взяли за основу структуру і принципи функціонування єдиного створеного природою пристрою, здатного міркувати, – мозку. Клітини мозку називаються нейронами, звідси і назва напряму. Вчені вважають, що, змоделювавши мозок, зможуть відтворити і його роботу.

Дослідники напряму «кібернетика чорного ящика» дотримувалися думки, що не важливо, за якими принципами працює пристрій, які засоби і методи лежать в його основі, головне імітувати функції мозку, навіть якщо окрім результату це не матиме нічого спільного з природним розумом.

В даний час стали помітні тенденції до об'єднання цих напрямків знову в єдине ціле. Стало з'являтися безліч гібридних методів та систем, наприклад експертна система на базі нейронної мережі або нейронна мережа, навчається генетичним алгоритмом.

Дослідники, що моделюють лише окремі функції інтелекту, наприклад розпізнавання образів, синтез мови, прийняття рішень, працюють у рамках напряму «слабкий штучний інтелект». Спроби відтворити роботу інтелекту в повному обсязі відносяться до напряму «сильний штучний інтелект». Всі основні досягнення в області штучного інтелекту відносяться до слабкого штучного інтелекту.

Крім цього виділяють спадний (семіотичний) та висхідний (біологічний) підходи.

Спадний підхід передбачає моделювання високорівневих психічних процесів, таких як мислення, мова, емоції і т. п.

Висхідний підхід досліджує інтелектуальну поведінку систем на базі більш дрібних «неінтелектуальних» елементів. Нейронні мережі та еволюційне моделювання відносяться до цього підходу.

Інтелектуальні системи розробляються з використанням різних засобів і методів. Існує чотири основні підходи до їх побудови: логічний, структурний, еволюційний та імітаційний.

Основою для *логічного підходу* служить булева алгебра. Така інтелектуальна система являє собою машину доведення теорем. При цьому вихідні дані зберігаються в базі даних у вигляді аксіом, правила логічного висновку – як відносини між ними. Крім того, кожна така машина має блок генерації цілі, та система виводу намагається довести дану мету як теорему. Якщо мета доведена, то трасування застосованих правил дозволяє отримати ланцюжок дій, необхідних для реалізації поставленої мети. Потужність такої системи визначається можливостями генератора цілей і машиною доведення теорем. Для більшості логічних методів характерна велика трудомісткість, оскільки під час пошуку доказу можливий повний перебір варіантів. Тому даний підхід вимагає ефективної реалізації обчислювального процесу, та хороший результат зазвичай гарантується при порівняно невеликому розмірі бази даних.

Під *структурним підходом* мають на увазі спроби побудови інтелектуальної системи шляхом моделювання структури людського мозку, тобто розглядаються системи, побудовані в рамках напряму «нейрокібернетика».

При побудові інтелектуальної системи за допомогою *еволюційного підходу* основна увага приділяється побудові початкової моделі і правилам, за якими вона може змінюватися (еволюціонувати). Причому модель може бути складена з найрізноманітніших методів: це може бути і нейронна мережа, і набір логічних правил, і будь-яка інша модель. На підставі перевірки моделей відбираються найкращі з них, і на їх базі по самим різним правилам генеруються нові моделі, з яких знову вибираються найкращі, тощо.

Імітаційний підхід використовується в рамках напряму «кібернетика чорного ящика». Інтелектуальні системи при такому підході повинні моделювати якусь інтелектуальну функцію, тобто встановлювати необхідну відповідність між входами і виходами системи.

Основні напрямки досліджень в області штучного інтелекту

Тематика штучного інтелекту охоплює величезний перелік наукових напрямків, починаючи з таких задач загального характеру, як навчання і сприйняття, та закінчуючи такими спеціальними задачами, як гра в шахи, доказ математичних теорем, створення поетичних творів та діагностика захворювань. У штучному інтелекті систематизуються і автоматизуються інтелектуальні завдання, і тому ця область стосується будь-якої сфери інтелектуальної діяльності людини.

Серед безлічі напрямків штучного інтелекту є кілька ведучих, які в даний час викликають найбільший інтерес у дослідників і практиків:

– представлення знань і розробка систем, заснованих на знаннях;

Це основний напрямок в області розробки систем штучного інтелекту. Він пов'язаний з розробкою моделей представлення знань, створенням баз знань, які становлять ядро експертних систем.

– програмне забезпечення систем штучного інтелекту;

В рамках цього напрямку розробляються спеціальні мови для вирішення інтелектуальних задач, в яких наголос робиться на логічну і символьну обробки ніж на обчислювальні процедури. Мови орієнтовані на символьну обробку інформації: LISP, PROLOG, РЕФАЛ та ін. Крім цього створюються пакети прикладних програм, орієнтовані на промислову розробку інтелектуальних систем, або програмні інструментарії штучного інтелекту.

– розробка природно-мовних інтерфейсів та машинний переклад;

Починаючи з 50-х років однією з популярних тем досліджень в галузі штучного інтелекту є комп'ютерна лінгвістика, і зокрема машинний переклад.

– інтелектуальні роботи;

Роботи – це електротехнічні пристрої, призначені для автоматизації людської праці. Виділяють кілька поколінь роботів.

І покоління. Роботи з жорсткою схемою управління. Практично всі сучасні промислові роботи належать до першого покоління. Фактично це програмовані маніпулятори.

ІІ покоління. Адаптивні роботи з сенсорними пристроями. Є зразки таких роботів, але в промисловості вони поки використовуються мало.

ІІІ покоління. Самоорганізовані або інтелектуальні роботи. Це кінцева мета розвитку робототехніки. Основні невирішені проблеми при створенні інтелектуальних роботів – проблема машинного зору і проблема адекватного зберігання і обробки тривимірної візуальної інформації.

– навчання та самонавчання;

Активно розвивається область штучного інтелекту, що включає моделі, методи і алгоритми, орієнтовані на автоматичне накопичення і формування знань на основі аналізу та узагальнення даних, навчання за прикладами (або індуктивний), а також традиційні підходи з теорії розпізнавання образів.

В останні роки до цього напрямку тісно примикають системи аналізу даних і пошуку закономірностей в базах даних.

– розпізнавання образів;

Напрямок штучного інтелекту, що бере початок у самих його витоків, але в даний час виділився в самостійну науку. Її основний підхід – опис класів об'єктів через певні значення ознак. Кожному об'єкту ставиться у відповідність матриця ознак, по якій відбувається його розпізнавання. Процедура розпізнавання використовує найчастіше спеціальні математичні процедури і функції, що розділяють об'єкти на класи.

– нові архітектури комп'ютерів;

Найсучасніші процесори сьогодні засновані на традиційній послідовній архітектурі фон Неймана, яка використовувалася ще в комп'ютерах перших поколінь. Ця архітектура вкрай неефективна для символьної обробки. Тому зусилля багатьох наукових колективів і фірм вже десятки років націлені на розробку апаратних архітектур, призначених для обробки символьних і логічних даних. Створюються Пролог та Лісп-машини, комп'ютери V і VI поколінь. І хоча вдалі промислові рішення існують, висока вартість, недостатнє програмне оснащення та апаратна несумісність з традиційними комп'ютерами істотно гальмують широке використання нових архітектур.

– ігри;

Це стало скоріше історичним напрямком пов'язаним з тим традиційно ігровими інтелектуальними задачами – шахи, шашки, тощо. В основі перших програм лежав один з ранніх підходів – лабіринтова модель мислення плюс евристика.

Зараз це швидше комерційний напрямок, так як в науковому плані ці ідеї вважаються тупиковими. В даний час в комп'ютерних іграх (наприклад, Unreal Tournament, Return to Castle Wolfe Stein, Black & White, Doom, Sim) стали застосовуватися інші ідеї штучного інтелекту – нейронні мережі, інтелектуальні агенти, генетичні алгоритми і т. п., які дозволяють створювати персонажів (ботів) з різним ступенем «інтелекту». Використання методів штучного інтелекту в іграх дозволяє отримувати нові ефективні рішення, створювати шаблони проектування, підвищувати розважальність і достовірність ігор.

– машинна творчість;

Напрямок охоплює створення комп'ютером, віршів, живопису і навіть казок та афоризмів. Основним методом подібної «творчості» є метод пермутації (перестановок) та використання деяких баз знань і даних, що містять результати досліджень по структурам текстів, рим, сценаріями і т. п.

– нечіткі моделі та м'які обчислення;

Цей напрямок представлено нечіткими схемами «виводу за аналогією», поглядом на теорію нечітких стратегій з імовірнісних позицій, нечітким поданням аналітичними моделями для опису геометричних об'єктів, алгоритмами еволюційного моделювання з динамікою, такими як час життя і розмір популяції, методами розв'язку оптимізаційних задач з використанням технологій генетичного пошуку, елементів самоорганізації.

– евристичне програмування;

В рамках напряму досліджують послідовності розумових операцій, виконання яких призводить до успішного вирішення того чи іншого завдання, моделюють розумову діяльність людини для вирішення завдань, що не мають суворого формалізованого алгоритму або пов'язаних з неповнотою вихідних даних.

– штучне життя;

Напрямок досліджень, метою якого є створення штучних істот, здатних діяти не менш ефективно, ніж живі істоти. Створюються обчислювальні системи і моделі, що діють на базі біологічних та еволюційних

принципів. В рамках цього напрямку використовують генетичні алгоритми, клітинні автомати, автономні агенти і т. п.

– когнітивне моделювання;

Науковий напрямок, що є синтезом когнітивної графіки та обчислювального моделювання, що дозволяє істотно підвищити пізнавальну ефективність сучасних ЕОМ. Методологія когнітивного моделювання призначена для аналізу і прийняття рішень в погановизначених ситуаціях, ґрунтується на моделюванні суб'єктивних уявлень експерта.

– еволюційне моделювання;

При еволюційному моделюванні процес моделювання складної соціально – економічної системи зводиться до створення моделі її еволюції або до пошуку допустимих станів системи, до процедури (алгоритму) відстеження допустимих станів (траєкторій).

– багатоагентні системи;

Напрямок штучного інтелекту, який розглядає рішення однієї задачі кількома інтелектуальними підсистемами – агентами. Агент – апаратна або програмна сутність, здатна діяти в інтересах досягнення мети, поставленої перед всією системою.

Соціальні системи дають ще одне модельне уявлення інтелекту за допомогою глобальної поведінки, що дозволяє їм вирішувати проблеми, які не вдалося вирішити окремим їх частинами. Агенти в таких системах автономні або напівавтономні, у кожного агента є певне коло підзадач. Кожен агент виконує свою незалежну частину рішення проблеми та або видає власний результат, або повідомляє результат іншим агентам.

– *Онтології;*

В рамках цього напрямку досліджується можливість всеосяжної і детальної формалізації деякої області знань за допомогою концептуальної схеми – ієрархічної структури даних, що містить всі релевантні класи об'єктів, їх зв'язок і правила предметної області. Онтології використовуються і людьми, і програмними агентами, дозволяють повторно використовувати знання предметної області, відокремлювати їх від оперативних знань і аналізувати їх. Розробляються мови опису онтологій (RDF, DAML, OWL, KIF).

– комп'ютерні віруси;

Останнє покоління комп'ютерних вірусів має всі атрибути систем штучного інтелекту. Ці віруси здатні до розмноження, мутації, еволюції, навчання. Сучасні проблеми щодо захисту від них виявляться

незначними, коли вони повністю проникають в сферу штучного інтелекту. Методи штучного інтелекту необхідні як для їх створення, так і для розробки ефективних засобів захисту.

– інтелектуальне математичне моделювання;

В даному напрямку системи імітують творчу діяльність математика – професіонала, що займається рішенням, наприклад, крайових задач математичної фізики. Для цього використовуються бази знань, що містять теореми, математичні залежності, евристичні правила. Такі системи здатні до навчання та самонавчання.

Це далеко не всі напрямки штучного інтелекту, існує безліч напрямків для вирішення безлічі завдань.

1.3. Класифікація інтелектуальних інформаційних систем

Існує велика безліч інтелектуальних інформаційних систем (ІІС). Однак загальноприйнятого єдиного визначення інтелектуальної інформаційної системи немає.

Інтелектуальною інформаційною системою називають автоматизовану інформаційну систему, засновану на знаннях, або комплекс програмних, лінгвістичних та логіко-математичних засобів для реалізації основного завдання – здійснення підтримки діяльності людини і пошуку інформації в режимі діалогу на природній мові.

Крім того, інформаційно-обчислювальними системами з інтелектуальною підтримкою для вирішення складних завдань називають ті системи, в яких логічна обробка інформації переважає над обчислювальною.

Таким чином, будь-яка інформаційна система, яка вирішує інтелектуальну задачу або у вирішенні яких беруть участь методи штучного інтелекту, відноситься до інтелектуальних.

Для інтелектуальних інформаційних систем характерні наступні ознаки:

– розвинені комунікативні здібності;
– вміння вирішувати складні поганоформалізовані завдання;
– здатність до самонавчання;
– адаптивність.

Комунікативні здібності ІІС характеризують спосіб взаємодії (інтерфейсу) кінцевого користувача з системою, зокрема можливість формулювання довільного запиту в діалозі з ІІС на мові, максимально наближеної до природньої.

Складні поганоформалізовані задачі – це задачі, які вимагають побудови оригінального алгоритму рішення в залежності від конкретної ситуації, для якої можуть бути характерні невизначеність та динамічність вихідних даних і знань.

Здатність до самонавчання – це можливість автоматичного видобування знань для вирішення задач з накопиченого досвіду конкретних ситуацій.

Адаптивність – здатність до розвитку системи відповідно до об'єктивним змінам моделі проблемної області.

Класифікація інтелектуальних систем

У відповідності з перерахованими ознаками ІІС діляться на (дана класифікація – одна з можливих) (рис. 1):

– системи з комутативними здібностями (з інтелектуальним інтерфейсом);
– експертні системи (системи для вирішення складних задач);
– самонавчальні системи (системи, здатні до самонавчання);
– адаптивні системи (адаптивні інформаційні системи).

Інтелектуальні бази даних відрізняються від звичайних баз даних можливістю вибірки формуватися по запиту необхідної інформації, яка може явно не зберігатися, а виводитися з наявної в базі даних.

Природномовний інтерфейс передбачає трансляцію природномовних конструкцій на машинний рівень представлення знань. Для цього необхідно вирішувати завдання морфологічного, синтаксичного та семантичного аналізу та синтезу висловлювань на природній мові. Так, морфологічний аналіз передбачає розпізнавання і перевірку правильності написання слів за словниками, синтаксичний контроль – розкладання вхідних повідомлень на окремі компоненти (визначення структури) з перевіркою відповідності граматичним правилам внутрішнього представлення знань та виявлення відсутніх частин і, нарешті, семантичний аналіз – встановлення смислової правильності синтаксичних конструкцій. Синтез висловлювань вирішує зворотну задачу перетворення внутрішнього представлення інформації в природномовну.

Природномовний інтерфейс використовується для:

– доступу до інтелектуальних баз даних;
– контекстного пошуку документальної текстової інформації;

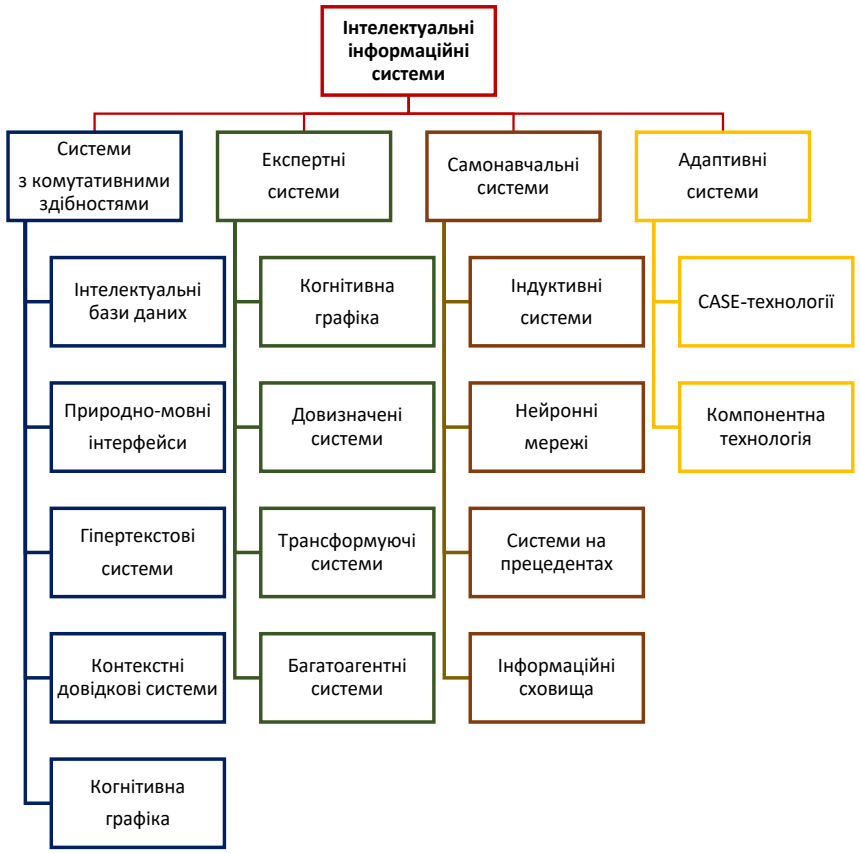


Рис. 1. Класифікація інтелектуальних інформаційних систем за типами систем

– голосового введення команд в системах управління;
– машинного перекладу з іноземних мов.

*Гіпертекстові систе*ми призначені для реалізації пошуку за ключовими словами в базах текстової інформації. Інтелектуальні гіпертекстові системи відрізняються можливістю більш складної семантичної організації ключових слів, яка відображає різні смислові відносини термінів. Таким чином, механізм пошуку працює перш за все з базою знань ключових слів, а вже потім з текстом. У більш широкому плані сказане

поширюється і на пошук мультимедійної інформації, що включає, крім текстової, й цифрову інформацію.

Системи контекстної допомоги можна розглядати як окремий випадок інтелектуальних гіпертекстових та природномовних систем. На відміну від звичайних систем допомоги, які нав'язують користувачеві схему пошуку необхідної інформації, в системах контекстної допомоги користувач описує проблему (ситуацію), а система за допомогою додаткового діалогу її конкретизує та виконує пошук рекомендацій, що відносяться до даної ситуації. Такі системи відносяться до класу систем поширення знань (Knowledge Publishing) і створюються як додаток до систем документації (наприклад, технічної документації по експлуатації товарів).

Системи когнітивної графіки дозволяють поєднувати інтерфейс користувача з ІІС за допомогою графічних образів, які генеруються відповідно до подій. Такі системи використовуються в моніторингу та управлінні оперативними процесами. Графічні образи в наочному і інтегрованому вигляді описують безліч параметрів досліджуваної ситуації. Наприклад, стан складного керованого об'єкта відображається у вигляді людського обличчя, на якому кожна риса відповідає за будь-якої параметр, а загальний вираз обличчя дає інтегровану характеристику ситуації. Системи когнітивної графіки широко використовуються також в навчальних і тренажерних системах на основі використання принципів віртуальної реальності, коли графічні образи моделюють ситуації, в яких тому хто навчається необхідно приймати рішення й виконувати певні дії.

Експертні системи призначені для вирішення задач на основі накопичуваної бази знань, що відображає досвід роботи експертів в даній проблемній області.

Багатоагентні системи. Для таких динамічних систем характерна інтеграція в базі знань декількох різнорідних джерел знань, які обмінюються між собою одержуваними результатами на динамічної основі.

Для багатоагентних систем характерні наступні особливості:

a) проведення альтернативних міркувань на основі використання різних джерел знань з механізмом усунення протиріччя;

b) розподілений розв'язок проблем, які розбиваються на підпроблеми, що вирішуються паралельно, які відповідають самостійним джерелам знань;

c) застосування стратегій роботи механізму виведення висновків залежно від типу розв'язуваної проблеми;

d) обробка великих масивів даних, що містяться в базі даних;

e) використання різних математичних моделей і зовнішніх процедур, що зберігаються в базі моделей;

f) здатність переривання розв'язку задач в зв'язку з необхідністю отримання додаткових даних і знань від користувачів, моделей, підпроблем.

В основі *самонавчальних систем* лежать методи автоматичної класифікації прикладів ситуацій реальної практики.

Характерними ознаками самонавчальних систем є:

– самонавчальні системи «з вчителем», коли для кожного прикладу задається в явному вигляді значення ознаки його приналежності певному класу ситуацій;

– самонавчальні системи «без вчителя», коли за ступенем близькості значень ознак класифікації система сама виділяє класи ситуацій.

– індуктивні системи використовують узагальнення прикладів за ступенем близькості значень ознак класифікації, система сама виділяє класи ситуацій.

Індуктивні системи використовують узагальнення прикладів за принципом від часткового до загального. Процес класифікації прикладів здійснюється наступним чином:

1) обирається ознака класифікації з множини заданих (або послідовно, або за будь-яким правилом);

2) за значенням обраної ознаки множина прикладів розбивається на підмножини;

3) виконується перевірка, чи належить кожна утворена підмножина прикладів одному підкласу;

4) якщо якась підмножина прикладів належить одному підкласу, тобто у всіх прикладів підмножини збігається значення ознаки, яка утворює клас, то процес класифікації закінчується (при цьому інші ознаки класифікації не розглядаються);

5) для підмножин прикладів з незбіжним значенням ознаки класу процес класифікації триває починаючи з пункту 1.

Нейронні мережі являють собою пристрої паралельних обчислень, що складаються з множини взаємодіючих простих процесорів. Кожен процесор такої мережі має справу тільки з сигналами, які він періодично отримує, і сигналами, які він періодично посилає іншим процесорам.

В експертних *системах, заснованих на прецедентах* (аналогіях), база знань містить описи не узагальнених ситуацій, а власне самі ситуації або прецеденти.

Пошук рішення проблеми в експертних системах, заснованих на прецедентах, зводиться до пошуку по аналогії (тобто абдуктивний висновок від часткового до часткового).

На відміну від інтелектуальної бази даних, *інформаційне сховище* являє собою сховище витягнутої значимої інформації з оперативної бази даних, яке призначене для оперативного ситуаційного аналізу даних (реалізації OLAP-технології).

Типовими завданнями оперативного ситуаційного аналізу є:

– визначення профілю споживачів конкретних об'єктів зберігання;

– прогнозування змін об'єктів зберігання в часі;

– аналіз залежностей ознак ситуацій (кореляційний аналіз).

Адаптивна інформаційна система – це інформаційна система, яка змінює свою структуру відповідно до зміни моделі проблемної області.

При цьому:

1) адаптивна інформаційна система повинна в кожен момент часу адекватно підтримувати організацію бізнес-процесів;

2) адаптивна інформаційна система повинна проводити адаптацію щоразу, як виникає потреба в реорганізації бізнес-процесів;

3) реконструкція інформаційної системи повинна проводитися швидко і з мінімальними витратами.

Ядром адаптивної інформаційної системи є модель проблемної області (підприємства), що постійно розвивається, підтримувана в спеціальній базі знань – репозиторії. На основі ядра здійснюється генерація або конфігурація програмного забезпечення. Таким чином, проектування та адаптація ІС зводиться, перш за все, до побудови моделі проблемної області та її своєчасному корегуванні.

Так як немає загальноприйнятого визначення, чітку єдину класифікацію інтелектуальних інформаційних систем дати важко.

Якщо розглядати інтелектуальні інформаційні системи з точки зору задачі, що розв'язується, то можна виділити системи управління і довідкові системи, системи комп'ютерної лінгвістики, системи розпізнавання, ігрові системи і системи створення інтелектуальних інформаційних систем (рис. 2).

При цьому системи можуть вирішувати не одну, а кілька задач або в процесі вирішення одної задачі вирішувати і ряд інших.

Наприклад, під час навчання іноземної мови система може вирішувати задачі розпізнавання мови учня, тестувати, відповідати на питання, перекладати тексти з однієї мови на іншу і підтримувати природномовний інтерфейс роботи.

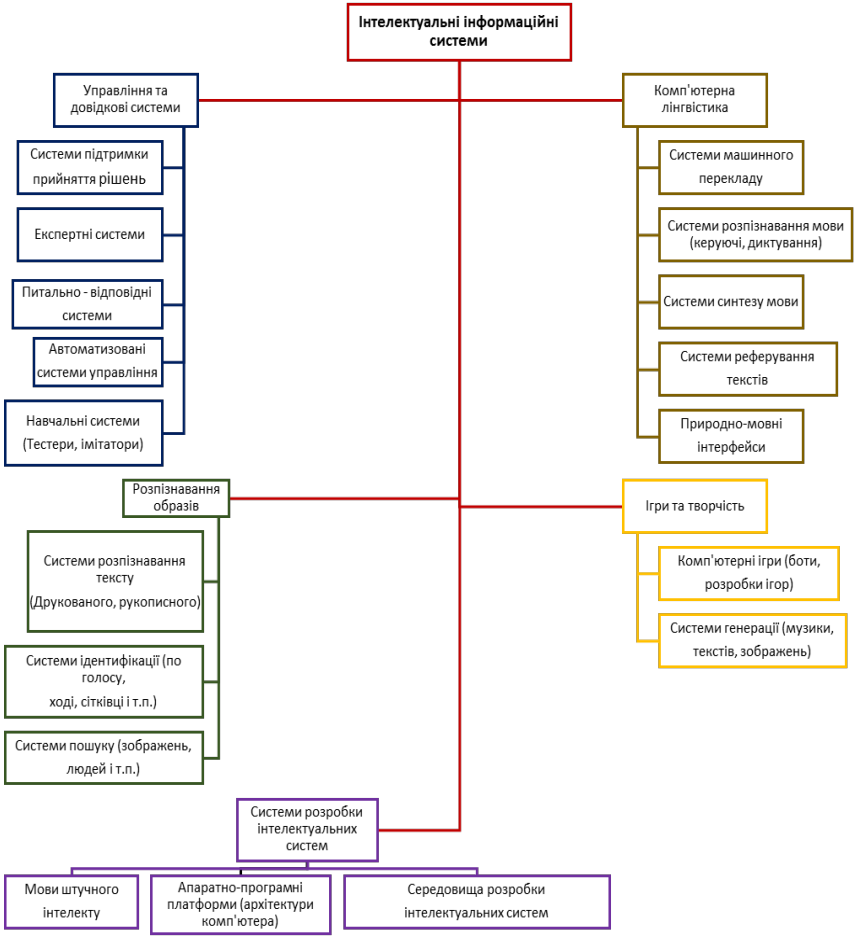


Рис. 2. Класифікація інтелектуальних інформаційних систем в залежності від задачі, що розв'язується

Якщо класифікувати інтелектуальні інформаційні системи за критерієм «методи, що використовуються», то вони діляться на жорсткі, м'які та гібридні (рис. 3).

М'які обчислення – це складна комп'ютерна методологія, заснована на нечіткій логіці, генетичних обчисленнях, нейрокомпьютерінгі та імовірнісних обчисленнях. *Жорсткі обчислення* – традиційні комп'ютерні обчислення (не м'які). *Гібридні системи* – системи, що використовують більш ніж одну комп'ютерну технологію (в разі інтелектуальних систем – технології штучного інтелекту.

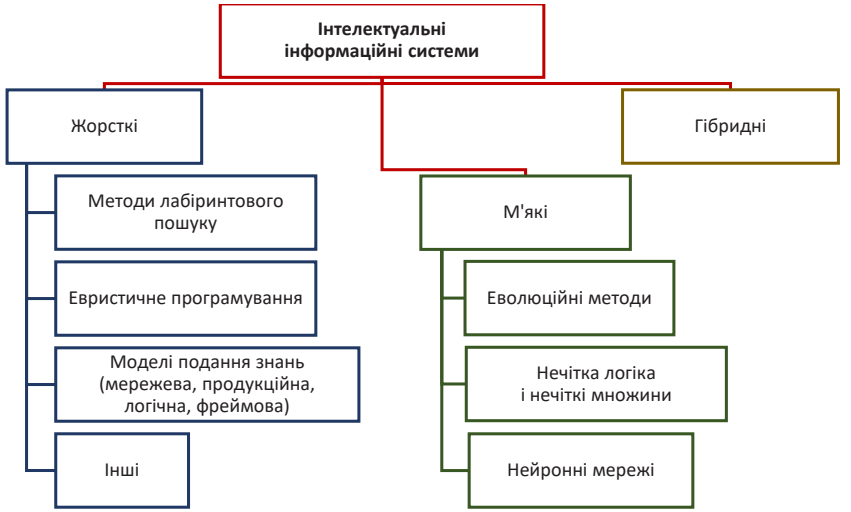


Рис. 3. Класифікація інтелектуальних інформаційних систем за методами

Можливі й інші класифікації, наприклад виділяють системи загального призначення і спеціалізовані системи (рис. 4).

Крім того, ця схема відображає ще один варіант класифікації по методам: системи, що використовують методи представлення знань, самоорганізовані та системи, створені за допомогою евристичного програмування. Також в цій класифікації системи генерації музики віднесені до систем спілкування.

До інтелектуальних систем загального призначення відносяться системи, які не тільки виконують задані процедури, але на основі метапроцедур пошуку генерують і виконують процедури рішення нових конкретних задач.

Спеціалізовані інтелектуальні системи виконують розв'язок фіксованого набору завдань, зумовленого при проектуванні системи.

Рис. 4. Класифікація інтелектуальних систем за призначенням

Відсутність чіткої класифікації також пояснюється різноманіттям інтелектуальних завдань і інтелектуальних методів, крім того, штучний інтелект – наука, яка активно розвивається, в якій щодня з'являються нові прикладні області.

1.4. Завдання для самоконтролю

Оберіть правильні відповіді.

1. Які передумови виникнення штучного інтелекту як науки?
А) поява ЕОМ
Б) розвиток кібернетики, математики, філософії, психології
В) наукова фантастика
Г) нема правильної відповіді

2. В якому році з'явився термін «штучний інтелект» (artificial intelligence)?
А) 1856
Б) 1956
В) 1954
Г) 1950
Д) нема правильної відповіді

3. Хто вважається родоначальником штучного інтелекту?
А) А. Тюрінг
Б) Аристотель
В) Р. Луллій
Г) Декарт
Д) нема правильної відповіді

4. Хто створив мову Lisp?
А) В. Ф. Турчин
Б) Д. Маккарті
В) М. Мінський
Г) Д. Робінсон
Д) нема правильної відповіді

5. Хто розробив мову РЕФАЛ?
А) Д. А. Поспєлов
Б) Г. С. Поспєлов
В) В. Ф. Турчин
Г) А. И. Берг
Д) нема правильної відповіді

6. Хто розробив теорію ситуаційного управління?
А) В. Ф. Турчин
Б) Г. С. Поспєлов

В) Д. А. Поспєлов
Г) Л. И. Микуліч
Д) нема правильної відповіді

7. Чим значимий 1964 рік для штучного інтелекту в вітчизняній історії?
А) створена мова РЕФАЛ
Б) створена Асоціації штучного інтелекту.
В) розроблений метод зворотного висновку Маслова.
Г) нема правильної відповіді

8. Який із напрямів не надає значення тому, як саме моделюються функції мозку?
А) нейрокібернетика
Б) кібернетика чорного ящика
В) немає правильної відповіді

9. Який підхід використовує булеву алгебру?
А) структурний
Б) імітаційний
В) логічний
Г) еволюційний
Д) немає правильної відповіді

10. Яку мову програмування розроблену в рамках штучного інтелекту?
А) Pascal
Б) C++
В) Lisp
Г) OWL
Д) PHP

11. Які задачі вирішуються в рамках штучного інтелекту?
А) розпізнавання мови
Б) прийняття рішень
В) кодування
Г) створення середовищ розробки інформаційних систем
Д) створення комп'ютерних ігор
Е) немає правильної відповіді

12. Експертні знання активно використовуються в наступних напрямках:
А) експертні системи
Б) когнітивне моделювання
В) розпізнавання образів

Г) комп'ютерна лінгвістика
Д) немає правильної відповіді

13. Принцип організації соціальних систем використовується в напрямку:
А) еволюційне моделювання
Б) когнітивне моделювання
В) нейронні мережі
Г) немає правильної відповіді

14. Інтелектуальна інформаційна система – це система ...
А) заснована на знаннях
Б) в якій логічна обробка інформації переважає над обчислювальною
В) відповідає на питання
Г) немає правильної відповіді

15. До яких інтелектуальних систем відноситься система, яка використовує генетичні обчислення та бази даних?
А) жорстким
Б) м'яким
В) гібридним

16. Системи генерації музики можна віднести до:
А) системам спілкування
Б) творчим системам
В) системам управління
Г) системам розпізнавання
Д) немає правильної відповіді

17. Які системи є системами загального призначення?
А) системи ідентифікації
Б) експертні системи
В) нейронні мережі
Г) робототехнічні системи
Д) немає правильної відповіді

18. До самоорганізуючих систем відносяться:
А) системи розпізнавання
Б) ігрові системи
В) системи реферування текстів
Г) нейронні мережі
Д) немає правильної відповіді

19. На знаннях ґрунтуються системи:
А) нейронні мережі
Б) системи розпізнавання тексту
В) експертні системи
Г) інтелектуальні пакети прикладних програм
Д) немає правильної відповіді

20. Евристичний пошук використовується в:
А) нейронних мережах
Б) експертних системах
В) ігрових системах
Г) немає правильної відповіді

21. До систем комп'ютерної лінгвістики належать:
А) система реферування текстів
Б) система розпізнавання мови
В) система генерації музики
Г) машинний переклад
Д) немає правильної відповіді

Література до розділу

1. Гаврилова, Т. А. Проблеми штучного інтелекту [Електронний ресурс] / Т. А. Гаврилова. – Режим доступу: http://www.big.spb.ru/publications/bigspb/km/problems_ai.shtml

2. Гаврилова Т. А. Бази знань інтелектуальних систем / Т. А. Гаврилова, В. Ф. Хорошевський. – СПб. : Пітер, 2001. – 384 с. (рос.)

3. Рассел, С. Штучний інтелект: сучасний підхід / С. Рассел, П. Норвіг. – 2-е вид. – М.: Вільямс, 2006. – 1408 с. (рос.)

4. Уітбі, Б. Штучний інтелект: реальний чи Матриця / Б. Уітбі. – М.: ФАИР-ПРЕСС, 2004. – 224 с. (рос.)

5. Ясницький, Л. Н. Введення в штучний інтелект / Л. Н. Ясницький. – М.: Изд. центр «Академія», 2005. – 176 с. (рос.)

6. Підходи до побудови систем штучного інтелекту [Електронний ресурс]. – Режим доступу: http://ai.obrazec.ru/podhody.html

7. Гаскаров, Д. В. Інтелектуальні інформаційні системи: підручник / Д. В. Гаскаров. – М. : Вища школа, 2003. – 431 с. (рос.)

8. Інтелектуальні інформаційні системи: навчальний посібник / А. А. Смагін, С. В. Ліпатова, О. С. Мельниченко. – Ульяновськ: УлГУ, 2010. – 136 с. (рос.)

РОЗДІЛ ІІ. МЕТОДИ ТА МОДЕЛІ ПРЕДСТАВЛЕННЯ ЗНАНЬ

2.1. Дані та знання

Дані – це інформація, отримана в результаті спостережень або вимірювань окремих властивостей (атрибутів), що характеризують об'єкти, процеси і явища предметної області.

Знання – форма існування і систематизації результатів пізнавальної діяльності людини. Знання допомагають людям раціонально організовувати свою діяльність і вирішувати різні проблеми, що виникають в процесі; суб'єктивний образ об'єктивної реальності, тобто адекватне віддзеркалення зовнішнього і внутрішнього світу в свідомості людини у формі уявлень, понять, суджень, теорій.

Знання (*в широкому сенсі*) – сукупність понять, теоретичних побудов та уявлень.

Знання (*у вузькому сенсі*) – ознака певного обсягу інформації, що визначає її статус і відокремлює від всієї іншої інформації за критерієм здатності до вирішення поставленого завдання.

Знання (*з точки зору представлення знань в системах штучного інтелекту*) – це зв'язки і закономірності предметної області (принципи, моделі, закони), отримані в результаті практичної діяльності та професійного досвіду, що дозволяє фахівцям ставити і вирішувати завдання в цій галузі.

Знання від даних відрізняються рядом властивостей:
– внутрішня інтерпретація;
– структурованість;
– зв'язність;
– семантична метрика;
– активність.

Внутрішня інтерпретація. Дані, що зберігаються в пам'яті або на зовнішніх носіях, позбавлені імен, таким чином, відсутня можливість їх однозначної ідентифікації системою. Дані може ідентифікувати лише програма, витягати їх за певним алгоритмом. При переході до знань

в пам'ять вводиться додаткова інформація (атрибути: прізвище, рік народження, спеціальність, стаж). Атрибути можуть грати роль імен. За ним можна здійснювати пошук потрібної інформації.

Структурованість. Інформаційні одиниці повинні володіти гнучкою структурою. Інакше кажучи, повинна існувати можливість довільного встановлення між окремими інформаційними одиницями відносин типу «частина – ціле», «рід – вид» або «елемент – клас».

Можливості підключення. Між інформаційними одиницями повинна бути передбачена можливість встановлення зв'язків різного типу. Семантика відносин може носити декларативний або процедурний характер. Наприклад, дві і більше інформаційні одиниці можуть бути пов'язані відношенням «одночасно», дві інформаційні одиниці – відношенням «причина – наслідок» або «бути поруч».

Семантична метрика. На множині інформаційних одиниць в деяких випадках корисно задавати відношення, що характеризує їх ситуаційну близькість, тобто силу асоціативного зв'язку. Його можна було б назвати відношенням релевантності для інформаційних одиниць. Воно дає можливість виділяти в інформаційній базі деякі типові ситуації (наприклад, «покупка», «регулювання руху на перехресті»). Ставлення релевантності при роботі з інформаційними одиницями дозволяє знаходити знання, близькі до вже знайденим.

Активність. Всі обчислювальні процеси ініціюють командами, а дані використовуються цими командами лише в разі потреби. Інакше кажучи, дані пасивні, а команди активні.

Знання дозволяють адаптуватися і діяти в реальній дійсності. Існує величезна безліч різних знань, починаючи від рецепта приготування омлету до квантової фізики.

Знання можна класифікувати за кількома критеріями (рис. 5).

Семантичне знання розглядається як структура, що утворює поточний контекст. Воно містить інформацію, безпосередньо пов'язану з поточними значеннями і змістом описуваних понять, та зумовлює стан зв'язків даних в інформаційній базі.

Прагматичне знання зумовлює найбільш ймовірні зв'язки, що описують дані з точки зору розв'язуваної задачі (узагальнений або «об'єктивний» контекст), наприклад з урахуванням діючих в даній задачі специфічних критеріїв та угод.

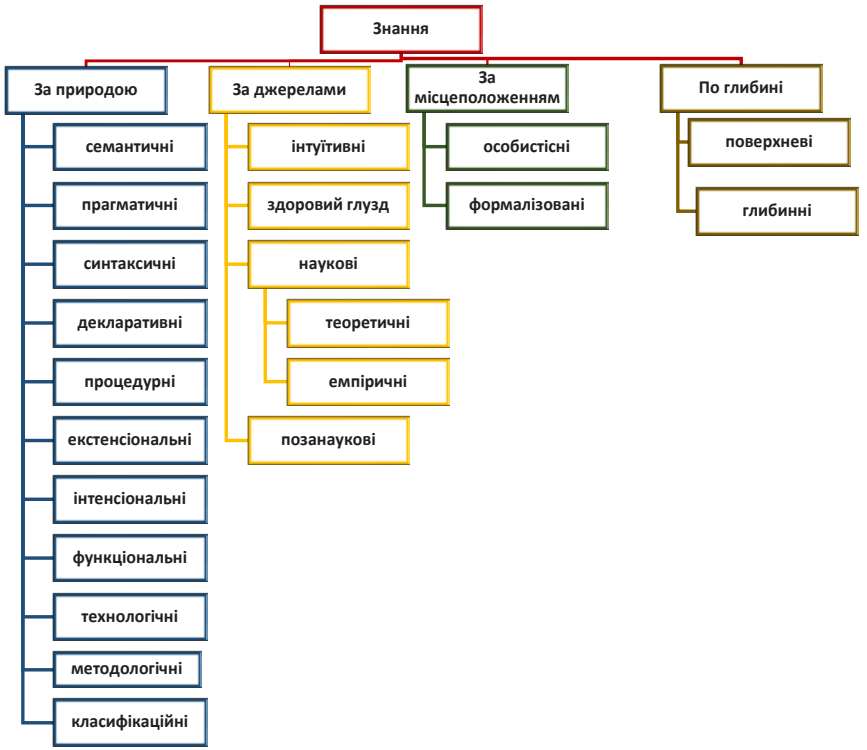


Рис. 5. Класифікація знань

Знання *синтаксичного* типу характеризує синтаксичну структуру потоку інформації, яка не залежить від змісту і змісту використовуваних при цьому понять, тобто інтелектуальну систему не утворює.

Декларативні знання містять у собі уявлення про структуру понять. Ці знання наближені до даних, фактів. Наприклад, вищий навчальний заклад є сукупність факультетів, а кожен факультет в свою чергу є сукупність кафедр.

Процедурні знання мають активну природу. Вони визначають уявлення про засоби і шляхи отримання нових знань, перевірки знань. Це алгоритми різного роду. З розвитком інформатики все більша частина знань зосереджувалася в структурах даних (таблиці, списки, абстрактні типи даних), тобто збільшувалася роль декларативних.

Суттєвими для розуміння природи знань є способи визначення понять. Один з широко застосовуваних способів заснований на ідеї інтенсіонала та екстенсіонала.

Інтенсіонал поняття – це визначення його через співвіднесення з поняттям більш високого рівня абстракції із зазначенням специфічних властивостей.

Екстенсіонал поняття – це визначення поняття через перерахування його конкретних прикладів, тобто понять більш низького рівня абстракції. Інтенсіонали формують знання про об'єкти, в той час як екстенсіонали об'єднують дані.

Звідси *інтенсіональні* знання – це знання про предметну область, яка відображає факти, закономірності, властивості і характеристики, справедливі для будь-яких ситуацій, які можуть виникнути в цій предметній області.

Екстенсіональні знання – це знання про предметну область, що відображають факти, закономірності, властивості і характеристики, типові для конкретних ситуацій або класів однотипних ситуацій, які можуть виникнути в цій області.

Функціональні знання – це знання про функції окремих предметів та про застосування їх в реальній дійсності.

Технологічні знання – спеціалізовані знання, для забезпечення підтримки технологічних параметрів виробництва; виробничий досвід і навички, які використовуються при вирішенні повсякденних виробничих питань. Це може бути знання послідовності операцій або знання технологічного ланцюга, що дозволяють досягати поставлених цілей відповідно до прийнятої технології.

Методологічні знання – знання про методи перетворення дійсності, наукові знання про побудову ефективної діяльності. До методологічних знань відносять знання цілей, форм і напрямків розвитку теорії, методів і способів ефективного перетворення практики.

Класифікаційні знання застосовуються головним чином в науці, є узагальненими, системними знаннями. Приклад – система елементів Д. І. Менделєєва.

Інтуїція – це вид знання, специфіка якого обумовлена способом його придбання. Це знання, що не потребує доведення та сприймається як достовірне. За способом отримання інтуїція – це прямий розсуд

об'єктивного зв'язку речей, що не спирається на доказ (інтуїція є розсуд внутрішнім зором; від лат. Intueri – споглядати).

Під *здоровим глуздом* розуміють знання, що дозволяють приймати правильні рішення і робити правильні припущення, ґрунтуючись на логічному мисленні та накопичений досвід. У цьому значенні термін часто акцентує увагу на здатності людського розуму протистояти забобонам, помилкам, містифікацій.

Наукові знання в будь-якому випадку повинні бути заснованими на емпіричній або теоретичній доказовій основі.

Теоретичні знання – абстракції, аналогії, схеми, що відображають структуру і природу процесів, що протікають в предметної області. Ці знання пояснюють явища і можуть використовуватися для прогнозування поведінки об'єктів. Теоретичний рівень наукового знання припускає встановлення законів, що дають можливість ідеалізованого сприйняття, описи і пояснення емпіричних ситуацій, тобто пізнання сутності явищ. Теоретичні закони мають більш строгий, формальний характер у порівнянні з емпіричними. Терміни опису теоретичного знання відносяться до ідеалізованим, абстрактним об'єктам. Подібні об'єкти неможливо піддати безпосередній експериментальній перевірці.

Емпіричні знання отримують в результаті застосування емпіричних методів пізнання: спостереження, вимірювання, експерименту. Це знання про видимі взаємозв'язки між окремими подіями і фактами в предметної області. Емпіричні знання, як правило, констатують якісні та кількісні характеристики об'єктів і явищ. Емпіричні закони часто носять імовірнісний характер і не є строгими.

Позанаукові знання можуть бути різними. Паранормальні знання – знання, несумісні з наявним гносеологічним стандартом. Широкий клас паранаукового (пара від грец. *близько*, *при*) знання включає в себе вчення або роздуми про феномени, пояснення яких не є переконливим з точки зору критеріїв науковості. *Псевдонаукові* знання – свідомо експлуатуючі домисли та забобони. Як симптоми псевдонауки виділяють малограмотний пафос, принципову нетерпимість до доводам, що спростовуються, а також претензійність. Псевдонаучні знання співіснують з науковими знаннями.

Особистісні (неявні, приховані) знання – це знання людей, отримані з практики та досвіду.

Формалізовані (явні) знання – знання, що містяться в документах, на компакт-дисках, в персональних комп'ютерах, в Інтернеті, в базах знань, в експертних системах. Формалізовані знання об'єктивізуються знаковими засобами мови, охоплюють ті знання, про які ми знаємо, їх можна записати, повідомити іншим.

2.2. Моделі представлення знань

2.2.1. Класифікація моделей представлення знань

Для зберігання даних використовуються бази даних (для них характерні великий обсяг та відносно невелика питома вартість інформації), для зберігання знань – бази знань (невеликого обсягу, але виключно дорогі інформаційні масиви).

База знань – основа будь-якої інтелектуальної системи, де знання описані на деякій мові представлення знань, наближеної до природньої. Сьогодні знання придбали чисто декларативну форму, тобто знаннями вважаються речення, записані на мовах представлення знань, наближених до природної мови і зрозумілі неспеціалістам.

Сукупність знань, потрібних для прийняття рішень, прийнято називати *предметною областю* або знаннями про предметну область. У будь-якій предметній області є свої поняття та зв'язки між ними, своя термінологія, свої закони, що зв'язують між собою об'єкти даних предметної області, свої процеси і події. Крім того, кожна предметна область має свої методи вирішення задач. Вирішуючи задачі такого виду на ЕОМ, використовують інформаційні системи, ядром яких є база знань, яка містить основні характеристики предметних областей.

Бази знань базуються на **моделях представлення знань**, подібно до баз даних, які засновані на моделях представлення даних (ієрархічної, мережевої, реляційної, постреляційної і т. п.).

При представленні знань в пам'яті інтелектуальної системи традиційні мови, засновані на чисельному представленні даних, є неефективними. Для цього використовуються спеціальні мови представлення знань, засновані на **символьному представленні даних**. Вони діляться на типи за формальними моделям представлення знань. Найбільш часто використовується на практиці класифікація моделей представлення знань, наведена на рис. 6, де моделі представлення знань поділяються на детерміновані (жорсткі) та м'які.

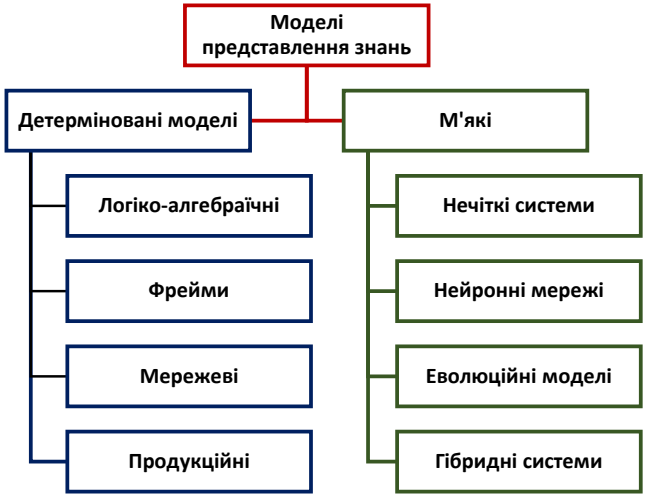


Рис. 6. Моделі представлення знань

Детерміновані моделі включають в себе фрейми, логіко-алгебраїчні моделі, семантичні мережі та продукційні моделі. М'які моделі включають в себе нечіткі системи, нейронні мережі, еволюційні моделі, гібридні системи.

З моделюванням знань безпосередньо пов'язана проблема вибору мови представлення. З метою класифікації моделей представлення знань виділяється дев'ять ключових вимог до моделей знань:

1) спільність (універсальність);
2) наочність представлення знань;
3) однорідність;
4) реалізація в моделі властивості активності знань;
5) відкритість;
6) можливість відображення структурних відносин об'єктів предметної області;
7) наявність механізму «проектування» знань на систему семантичних шкал;
8) можливість оперування нечіткими знаннями;
9) використання багаторівневих представлень (дані, моделі, метамоделі, метаметамоделі та т. п.).

Моделі представлення знань не задовольняють повністю цим вимогам, чим і пояснюється їх різноманіття та активний розвиток даного напрямку.

2.2.2. Логіко-алгебраїчні моделі представлення знань

У логічних моделях знання представляються у вигляді сукупності правильно побудованих формул будь-якої формальної системи (ФС), яка задається четвіркою

$$S = < T, P, A, R >, \quad (1)$$

де T – множина базових (терміни) елементів, з яких формуються всі вирази; P – множина синтаксичних правил, що визначають синтаксично правильні вирази з базових елементів ФС; A – множина аксіом ФС, відповідних синтаксично правильним виразами, які в рамках даної ФС апріорно вважаються істинними; R – кінцева множина відносин $\{r_1, r_2, ..., r_n\}$ між формулами-правилами виведення, що дозволяють отримувати з одних синтаксично правильних виразів інші.

Для будь-якого r_i існує ціле додатне число j, таке що для кожної множини, що складається з j формул, та для кожної формули F ефективно вирішується питання про те, чи знаходяться ці j-формули у відношенні r_i з формулою F. Якщо r_i виконується, то F називають безпосереднім висновком F-формул за правилом r_i.

Висновком формули в теорії S називається така послідовність правил, що для будь-якого з них представлена формула є або аксіомою теорії S, або безпосереднім наслідком (виводом, висновком).

Найпростішою логічною моделлю є числення висловів, яке представляє собою один з початкових розділів математичної логіки, що є основою для побудови більш складних формалізмів. У практичному плані числення висловлювань застосовується в ряді предметних областей (зокрема, при проектуванні цифрових електронних схем). Розвиток логіки висловлювань знайшло відображення в численні предикатів першого порядку.

Під численням предикатів розуміється формальна мова для представлення відносин в деякій предметній області. Основна перевага числення предикатів – добре зрозумілий механізм математичного виводу, який може бути безпосередньо запрограмований. Предикатом називають речення, яке приймає тільки два значення: «істина» або

«хибне». Для позначення предикатів застосовуються логічні зв'язки між висловлюваннями: ¬ – ні, ∨ – або, ∧ – і, ⊃ – якщо, а також квантор існування ∃ і квантор загальності ∀.

Таким чином, логіка предикатів оперує логічними зв'язками між висловлюваннями, наприклад вона вирішує питання: чи можна на основі висловлювання А отримати висловлювання В та т.п.

Допустимі висловлювання у численні предикатів називаються правильно побудованими формулами, що складаються з атомних формул. *Атомні* формули складаються з предикатів і термів, що розділяються круглими, квадратними і фігурними дужками.

Предикатні символи – це в основному дієслівна форма (наприклад: ПИСАТИ, ВЧИТИ, ПЕРЕДАТИ), але не тільки дієслівна форма, а форма прикметників, прислівників (наприклад: ЧЕРВОНИЙ, ЗНАЧЕННЯ, ЖОВТИЙ).

Предикатні символи і константи, як правило, позначаються великими символами, функціональні символи та змінні – малими.

В абстрактних прикладах вони позначаються латинськими літерами f, g, h. У реченнях предикатної форми важливі відносини і елементи. Визначаючи відносини, ми визначаємо значимість елементів вираження. Елементи можуть бути предикатами та термами.

Якщо існує деяка предметна область, то предикати визначають відносини в цій предметній області, константи – елементи цієї предметної області, функціональний символ – функцію.

Розглянемо деякі приклади. Вислів «у кожної людини є ім'я» можна записати:

∀x ∃y (ЛЮДИНА(x) ⊃ ІМ'Я (y,x))

Вираз «У студента є залікова синя книжка» записується, наприклад, так:

∃x (ВОЛОДІЄ(СТУДЕНТ, x) ⊃ ЗАЛІКОВА КНИЖКА(x) ∧СИНІЙ(x))

Представлення знань в межах логіки предикатів служить основою напрямку ШІ, який називається логічне програмування. Методи логічного програмування в даний час широко використовуються на практиці при створенні систем штучного інтелекту в ряді предметних областей. Позитивними рисами логічних моделей знань в цілому є:

– високий рівень формалізації, що забезпечує можливість реалізації системи формально точних визначень та висновків;

– узгодженість знань як єдиного цілого, що полегшує рішення проблем верифікації бази знань, оцінки незалежності та повноти системи аксіом і т. п.;

– єдині засоби опису як знань про предметну область, так і способів вирішення завдань в цій предметній області, що дозволяє будь-яку задачу звести до пошуку логічного виводу деякої формули в тій чи іншій ФС.

Однак така однаковість тягне за собою основний недолік моделі – складність використання в процесі логічного висновку евристик, що відображають специфіку предметної області. До інших недоліків логічної моделі відносять:

– «монотонність»;
– «комбінаторний вибух»;
– слабкість структурованості описів.

2.2.3. Продукційні моделі представлення знань

Продукційна модель або модель, заснована на правилах, дозволяє представити знання у вигляді пропозицій типу «якщо (умова), то (дія)».

Під «умовою» (антецедентом) розуміється деяке речення-зразок, за яким здійснюється пошук в базі знань, а під «дією» (консеквентом) – дії, що виконуються при успішному результаті пошуку (вони можуть бути проміжними, які виступають далі як умови, і термінальними або цільовими, що завершують роботу системи). Крім цього, будь-яка продукція має ім'я і пріоритет, який визначає послідовність перевірки продукцій машиною виводу. Продукції відображають причинно-наслідкові зв'язки, які і дозволяють людині приймати рішення, базуючись на знаннях та припущеннях про те, що є і що буде, якщо щось зробити.

Продукційна модель в чистому вигляді не має механізму виводу з тупикових станів у процесі пошуку. Вона продовжує працювати, поки не будуть вичерпані всі допустимі продукції. Практичні реалізації продукційних систем містять механізми повернення в попередній стан для управління алгоритмом пошуку.

Основні переваги продукційних систем:

– простота і гнучкість представлення знань;
– відокремлення знань від програми пошуку;
– модульність продукційних правил (правила не можуть «викликати» інші правила);
– можливість евристичного управління пошуком;
– можливість трасування «ланцюга міркувань»;
– незалежність від вибору мови програмування;
– продукційні правила є правдоподібною моделлю рішення задачі людиною.

Є велика кількість програмних засобів, що реалізують продукційний підхід (наприклад, мови високого рівня CLIPS і OPS 5; «оболонки» або «порожні» ЕС – EXSYS Professional та Карра, інструментальні – KEE, ARTS, PIES та інш.).

Приклад розв'язку задачі

Задача. Побудувати продукційну модель представлення знань в предметній області «Ресторан» (відвідування ресторану).

Опис процесу розв'язку.

Для побудови продукційної моделі представлення знань необхідно виконати наступні кроки:

1. Визначити цільові дії задачі (що є рішеннями).
2. Визначити проміжні дії або ланцюг дій, між початковим станом і кінцевим (між тим, що є, і цільовою дією).
3. Визначити умови для кожної дії, при яких їх доцільно і можливо виконати. Визначити порядок виконання дій.
4. Додати конкретики при необхідності, виходячи з поставленого завдання.
5. Перетворити отриманий порядок дій і відповідні їм умови в продукції.
6. Для перевірки правильності побудови продукцій записати ланцюг продукцій, явно простеживши зв'язок між ними. Цей набір кроків передбачає рух при побудові продукційної моделі від результату до початкового стану, але можливо і рух від початкового стану до результату (кроки 1 і 2).

Розв'язок

1. Обов'язкова дія, яка виконується в ресторанах – споживання їжі та її оплата. Значить, є вже дві цільової дії «з'їсти їжу» та «оплатити», які взаємопов'язані і слідують одна за одною.

2. Перш ніж що-небудь з'їсти в ресторані, туди потрібно прийти, дочекатися офіціанта та зробити замовлення. Крім того, потрібно вибрати, в який саме ресторан піти. Значить, ланцюг проміжних дій: «вибір ресторану і шлях туди», «зробити замовлення офіціантові».

3. Перш ніж йти в ресторан, необхідно переконатися, що є необхідна сума грошей. Вибір ресторану може обумовлюватися багатьма причинами, виберемо територіальну ознаку – до якого ближче в той й йдемо. У різних ресторанах працюють різні люди, тому в залежності від вибору ресторану, офіціанти будуть різні. Крім того, різні ресторани спеціалізуються на різних кухнях, тому замовлені страви будуть

в різних ресторанах відрізнятися. Значить спочатку йдуть дії, що дозволяють вибрати ресторан, потім характеризують ресторани, а вже після замовлення, їжа, і оплата замовлення.

4. Нехай в задачі будуть розглядатися два ресторани: «Смачна їжа» та «Смакота». Перший – паб і замовлення приносять швидше, ніж у другому, другий – піцерія. У першому працює офіціант Сергій, а в другому офіціантка Марина. Петро – це клієнт.

5. Представлений вище опис можна перетворити в наступні речення типу «Якщо, то»:

– *Якщо суб'єкт хоче їсти та у суб'єкта є достатня сума грошей, то суб'єкт може піти в ресторан.*

– *Якщо суб'єкт знаходиться ближче до ресторану «Смачна їжа», ніж до ресторану «Смакота» і суб'єкт може піти в ресторан, то суб'єкт йде в ресторан «Смачна їжа».*

– *Якщо суб'єкт ближче до ресторану «Смакота», ніж до ресторану «Смачна їжа» і суб'єкт може піти в ресторан, то суб'єкт йде в ресторан «Смакота».*

– *Якщо суб'єкт йде в ресторан «Смакота» і в ресторані «Смакота» працює офіціант Марина, то у суб'єкта приймає замовлення Марина.*

– *Якщо суб'єкт йде в ресторан «Смачна їжа» і в ресторані «Смачна їжа» працює офіціант Сергій, то у суб'єкта приймає замовлення Сергій.*

– *Якщо суб'єкт вибрав страви і у суб'єкта приймає замовлення Марина, то замовлення принесуть через 20 хв.*

– *Якщо суб'єкт вибрав страви і у суб'єкта приймає замовлення Сергій, то замовлення принесуть через 10 хв.*

– *Якщо замовлення принесуть через 20 хв. або замовлення принесуть через 10 хв., то суб'єкт може їсти.*

– *Якщо суб'єкт може їсти, то після їжі суб'єкт повинен оплатити замовлення*.

Введемо позначення для фактів (Ф), дій (Д) та продукцій (П), тоді:

Суб'єкт = Петро;

Ф1 = суб'єкт хоче їсти;

Ф2 = у суб'єкта є достатня сума грошей;

Ф3 = суб'єкт ближче до ресторану «Смачна їжа», ніж до «Смакота»;

Ф4 = в ресторані «Смакота» працює офіціант Марина;

Ф5 = в ресторані «Смачна їжа» працює офіціант Сергій;

Ф6 = суб'єкт вибрав страви;

Д1 = суб'єкт може піти в ресторан;

Д2 = суб'єкт йде в ресторан «Смачна їжа»;

Д3 = суб'єкт йде в ресторан «Смакота»;

Д4 = у суб'єкта приймає замовлення Марина;

Д5 = у суб'єкта приймає замовлення Сергій;

Д6 = замовлення принесуть через 20 хв.

Д7 = замовлення принесуть через 10 хв.

Д8 = після їжі суб'єкт повинен оплатити замовлення.

Для продукцій встановимо пріоритет (в дужках перед комою, чим вище пріоритет, тим раніше перевіряється правило).

П1 (4, Ф1 і Ф2) = Д1;
П2 (5, Ф3 і Д1) = Д2;
П3 (4, що не Ф3 і Д1) = Д3;
П4 (3, Д3 і Ф4) = Д4;
П5 (3, Д2 і Ф5) = Д5;
П6 (2, Д4) = Д6;
П7 (2, Д5) = Д7;
П8 (1, Д6 або Д7) = Д8;

6. Для відображення взаємозв'язку продукцій побудуємо граф (рис. 7).

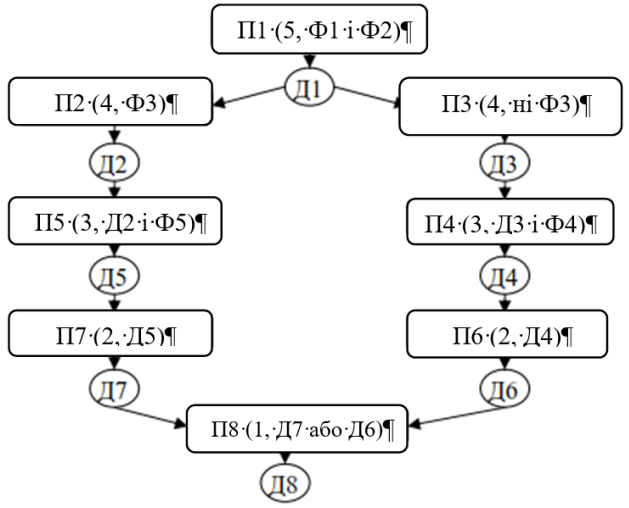


Рис. 7. Схема продукцій предметної області «Ресторан»

2.2.4. Семантичні мережі

Семантична мережа – це орієнтований граф, вершини якого – поняття, а дуги – відносини між ними. Термін *«семантична»* означає «смислова», а сама *семантика* – це наука, що встановлює *відносини* між символами і об'єктами, які вони позначають, тобто наука, яка визначає зміст знаків. Модель на основі семантичних мереж була запропонована американським психологом Куілліаном.

Вузли семантичної мережі зазвичай відповідають об'єктам, концепціям, подіям або поняттям. Будь-який фрагмент мережі, наприклад одна вершина, дві вершини та дуги, які їх з'єднують, називають підмережею. Логічний висновок (пошук рішення) на семантичній мережі полягає в тому, щоб знайти або сконструювати підмережу, що задовольняє деяким умовам. Відносини, що подаються дугами, в семантичній мережі можуть бути різними (таб. 1). Типи відносин вибираються в залежності від виду семантичної мережі (таб. 2) задачі, що розв'язується.

Таблиця 1

Основні види відносин в семантичних мережах

Тип	Опис
бути спадкоємцем (a-kind-of)	задає ієрархічні зв'язки між класами
задає ієрархічні зв'язки між класами (Is-a, наприклад)	бути екземпляром, визначає значення, описує конкретний об'єкт, поняття
це (are, є)	може використовуватися замість зв'язку a-kind-of у відносинах рівності або еквівалентності
бути частиною (has-part)	визначає структурні зв'язки, описує частини або цілі об'єкти.
Функціональні	визначаються зазвичай дієсловами, відображають різні відносини (вчити, володіти і т. п.).
Кількісні	відображають кількісні співвідношення між вершинами (Більше, менше і т. п.)
Просторові	відображають просторові відносини між вершинами (Близько, далеко і т. п.)

Закінчення таблиці 1

Тип	Опис
Тимчасові	описують тимчасові зв'язки між вершинами (скоро, довго, зараз і т. п.)
Атрибутивні	описують властивості об'єктів, понять
Логічні	описують логічні зв'язки між вершинами (і, або, не)

Таблиця 2

Типи семантичних мереж

Тип	Опис
За типом знання	
екстенсіональні	описує конкретні відносини даної ситуації.
інтенсіональні	описують імена класів об'єктів, а не індивідуальні імена об'єктів, зв'язки відображають ті відносини, які завжди притаманні об'єктам даного класу.
За типом обмежень на дуги і вершини	
Прості	вершини мережі не володіють внутрішньою структурою
ієрархічні	вершини мають внутрішню структуру, в ієрархічній мережі є можливість розділяти мережу на підмережі і встановлювати відносини не тільки між вершинами, а й між підмережами (різні підмережі, існуючі в мережі, можуть бути впорядковані у вигляді дерева підмереж, вершини якого – підмережі, а дуги – відносини)
Динамічні (сценарії)	мережі з подіями
За кількістю типів відносин	
Однорідні	володіють тільки одним типом відносин
Неоднорідні	кількість типів відносин більше двох
За арністю відносин	
Бінарні	Всі відносини в графі пов'язують рівно два поняття
N-арні	в мережі є відносини, що зв'язують більше двох об'єктів

Приклад розв'язку задачі

Задача. Побудувати мережеву модель представлення знань в предметній області «Ресторан» (відвідування ресторану).

Опис процесу розв'язку.

Для побудови мережевої моделі представлення знань необхідно виконати наступні кроки:

1. Визначити абстрактні об'єкти і поняття предметної області, необхідні для вирішення поставленого завдання. Оформити їх у вигляді вершин.

2. Задати властивості для виділених вершин, оформивши їх у вигляді вершин, пов'язаних з вихідними вершинами атрибутивними відносинами.

3. Задати зв'язки між цими вершинами, використовуючи функціональні, просторові, кількісні, логічні, часові, атрибутивні відносини, а також відносини типу «бути спадкоємцем» та «бути частиною».

4. Додати конкретні об'єкти та поняття, що описують задачу, яка розв'язується. Оформити їх у вигляді вершин, пов'язаних з уже існуючими відносинами типу «бути екземпляром», «є».

5. Перевірити правильність встановлених відносин (вершини і саме відношення при правильній побудові утворюють речення, наприклад «Двигун є частиною автомобіля»).

Розв'язок

1. Ключові поняття даної предметної області – ресторан, той, хто відвідує ресторан (клієнт) і ті, хто його обслуговують (кухарі, метрдотелі, офіціанти, для простоти обмежимося тільки офіціантами). У обслуговуючого персоналу і клієнтів є загальні характеристики, тому доцільно виділити загальне абстрактне поняття – людина. Продукцією ресторану є страви, які замовляють клієнти. Виходячи з цього, вершини графа будуть наступними: «Ресторан», «Людина», «Офіціант», «Клієнт», «Замовлення» та «Страва».

2. У цих об'єктів є певні властивості і атрибути. Наприклад, ресторани розташовуються за певними адресами, кожне блюдо з меню має свою ціну. Тому додамо вершини «Адреса» та «Ціна».

3. Визначимо для наявних вершин відносини і їх типи, використовуючи таблицю 1.

4. Додамо знання про конкретні факти задачі. Нехай є два ресторани: «Смакота» і «Смачна їжа», офіціантка Марина, а в другому офіціант Сергій. Петро вирішив піти в ресторан «Смачна їжа» і зробив замовлення офіціантові на 2 страви: картопля фрі за 30 грн., Біфштекс за 130 грн. Також відомі адреси цих ресторанів і їх специфіка. Виходячи з цього, додамо відповідні вершини в граф і з'єднаємо їх функціональними відносинами і відносинами типу «наприклад» або «бути екземпляром». Отриманий в результаті граф зображений на рис. 8.

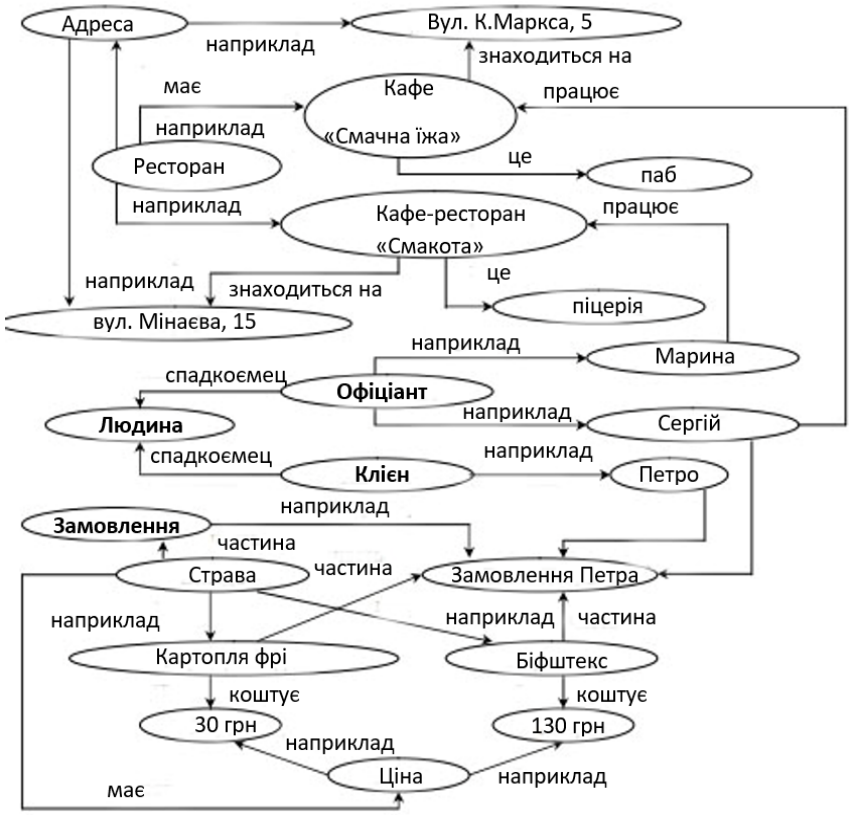


Рис. 8. Семантична мережа предметної області «Ресторан»

5. Здійснимо перевірку встановлених зав'язків. Наприклад, візьмемо вершину «Страва» і пройдемо по встановленим зв'язкам. Отримуємо наступну інформацію: блюдо є частиною замовлення, прикладами страв можуть служити картопля фрі та біфштекс.

Для отримання відповіді на будь-яке питання по цій задачі, необхідно знайти відповідну ділянку мережі і, використовуючи зв'язки, отримати результат. Наприклад, питання «Яка ціна замовлення Петра (скільки Петро заплатив за замовлення)?» З запиту зрозуміло, що необхідно знайти такі вершини: «Ціна», «Петро» та «Замовлення» або «Замовлення Петра». Частина семантичної мережі, що знаходиться між цими вершинами, містить відповідь, а саме, частиною замовлення Петра є картопля фрі і біфштекс, які коштують 30 і 130 грн. відповідно. Більше інформації про замовлення Петра в моделі немає, тому робимо висновок – Петро заплатив 160 грн.

2.2.5. Фрейми

Фрейм – це мінімально можливий опис сутності будь-якої події, ситуації, процесу або об'єкта. В історичному плані розвиток фреймовой моделі пов'язано з теорією фреймів М. Мінського, 1979 р., яка визначала спосіб формалізації знань, які використовуються при вирішенні задач розпізнавання образів (сцен) та розуміння мови. «Відправним моментом для даної теорії служить той факт, що людина, намагаючись пізнати нову для себе ситуацію або по-новому поглянути на вже звичні речі, обирає зі своєї пам'яті деяку структуру даних (образ), звану нами фреймом, з таким розрахунком, щоб шляхом зміни в ній окремих деталей зробити її придатною для розуміння більш широкого класу явищ або процесів». Іншими словами, фрейм – це форма опису знань, що окреслює рамки розглянутого (в поточній ситуації при вирішенні даної задачі) фрагмента предметної області.

Фрейм (англ. Frame) – абстрактний образ для представлення деякого стереотипу сприйняття. Кожен фрейм має власну назву і список слотів та їх значень. Значеннями можуть бути дані будь-якого типу, а також назва іншого фрейму. Таким чином, фрейми утворюють мережу.

Крім того, існує зв'язок між фреймами типу АКО (a kind of), який вказує на фрейм більш високого рівня ієрархії, звідки неявно успадковуються список і значення слотів. При цьому можливо множинне спадкування – перенесення властивостей від декількох прототипів. Будь-який фрейм може бути представлений таким чином:

(ім'я фрейму:
(ім'я 1-го слоту: значення 1-го слоту),
(ім'я 2-го слоту: значення 2-го слоту),
(Ім'я N- го слоту: значення N-го слоту)).

Табличне представлення слоту виглядає наступним чином (таб. 3)

Таблиця 3

Структура фрейму

Ім'я фрейму			
Ім'я слоту	Значення слоту	Спосіб отримання значення	Демон

При табличному представленні фрейму крім уже описаних складових фрейму вказуються і додаткові параметри. Спосіб отримання значення визначає, як саме встановлюється значення конкретного слоту. Існує кілька способів (таб. 4), вибір способу залежить від властивостей самих даних.

Таблиця 4

Способи отримання значень слотів

Спосіб	**Опис**
За замовчуванням від прототипу (одного з батьків)	Слоту присвоюється значення, визначене за умовчанням в фреймі-прототипі, деякі стандартні значення.
Через спадкування	Відрізняється від першого способу тим, що значення задано в спеціальному слоті батьківського фрейму, з'єднаного з поточним зв'язком АКО.

Закінчення таблиці 4

Спосіб	Опис
За формулою	Слоту призначається формула, результат обчислення якої є значенням слоту.
Через приєднану процедуру	Слоту призначається процедура, що дозволяє отримати значення слота алгоритмічно.
Із зовнішніх джерел даних	При використанні моделі в інтелектуальних системах дані, які є значеннями слотів, можуть надходити з баз даних, від системи датчиків, від користувача.

В теорії фреймів допускається, щоб до слотів приєднувалися різні спеціальні процедури. Для цього використовуються так звані демони. *Демоном* (таб. 5) називається процедура, яка автоматично запускається при виконанні деякої умови (події) при зверненні до відповідного слоту. Демонів може бути декілька.

Таблиця 5

Найбільш поширені демони

Демон	Подія	Опис
IF-REMOVED	якщо видалено	Виконується, коли інформація видаляється з слоту.
IF-ADDED	якщо додано	Виконується, коли нова інформація записується в слот.
IF-NEEDED	за вимогою	Виконується, коли запитується інформація з порожнього слоту.
IF-DEFAULT	за замовчуванням	Виконується, коли встановлюється значення за замовчуванням.

Існує декілька видів фреймів, які дають можливість окреслити предметну область і задачу, яка розв'язується. У таб. 6 представлені найбільш поширені типи фреймів, вказані типи знань, які вони відображають, а також приклади фреймів даного типу з різних предметних галузей.

Таблиця 6

Типи фреймів

Тип фрейма	Тип знання	Опис	Приклад
За пізнавальним призначенням			
Фрейми-прототипи (шаблони, зразки)	Інтенсіональні	відображають знання про абстрактні стереотипні поняття, які є класами якихось конкретних об'єктів	людина, автомобіль
Фрейми-екземпляри (приклади)	Екстенсіональні	відображають знання про конкретні факти предметної області	Іванов І. І., ВАЗ-2110
За функціональним призначенням			
Фрейми-структури (об'єкти)	Декларативні	відображають абстрактні і конкретні предмети і поняття предметної області (Містять набір характеристик, описує об'єкт або поняття)	позика, застава, вексель, людина, лекція
Фрейми-операції	Процедурні	відображають різні процеси перетворення або використання об'єктів предметної області (містять набір характеристик процесу)	Процеси отримання позики, синтез пристроїв
Фрейми-ситуації	Прагматичні	відображають типові ситуації, в яких можуть перебувати фрейми об'єкти і фрейми ролі (містять набір характеристик, що ідентифікують ситуацію)	аварія, тривога, робочий режим пристрою
Фрейми-сценарії	Технологічні	відображають розвиток ситуації, типову структуру для деякої дії, поняття, події, відображає динаміку	банкрутство, святкування іменин, здача іспиту
Фрейми-ролі	Функціональні	відображають типову роль, виконувану фреймом-об'єктом у певній ситуації (містять набір характеристик ролі)	менеджер, касир, клієнт, студент, викладач

Приклад розв'язку задачі.

Задача. Побудувати фреймову модель представлення знань в предметній області «Ресторан» (відвідування ресторану).

Опис процесу розв'язку.

Для побудови фреймової моделі представлення знань необхідно виконати наступні кроки:

1. Визначити абстрактні об'єкти і поняття предметної області, необхідні для вирішення поставленої задачі. Оформити їх у вигляді фреймів-прототипів (фреймів-об'єктів, фреймів-ролей).

2. Задати конкретні об'єкти предметної області. Оформити їх у вигляді фреймів-екземплярів (фреймів-об'єктів, фреймів-ролей).

3. Визначити набір можливих ситуацій. Оформити їх у вигляді фреймової ситуації (прототипів). Якщо існують прецеденти по ситуаціям в предметній області, додати фрейми-екземпляри (фрейми-ситуації).

4. Описати динаміку розвитку ситуацій (перехід від одних до інших) через набір сцен. Оформити їх у вигляді фреймів-сценаріїв.

5. Додати фрейми-об'єкти сценаріїв і сцен, які відображають дані конкретної задачі.

Розв'язок

1. Ключові поняття даної предметної області – ресторан, той, хто відвідує ресторан (клієнт) і ті, хто його обслуговують (кухарі, метрдотелі, офіціанти, для простоти обмежимося тільки офіціантами). У обслуговуючого персоналу і клієнтів є загальні характеристики, тому доцільно виділити загальне абстрактне поняття – людина. Тоді фрейми «Ресторан» і «Людина» є прототипами-зразками, а фрейми «Офіціант» і «Клієнт» – прототипами-ролями. Також потрібно визначити основні слоти фреймів – характеристики, що мають значення для даної задачі.

ЛЮДИНА			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Стать	Чоловічий чи жіночий	із зовнішніх джерел	
Вік	Від 0 до 120 років	із зовнішніх джерел	

РЕСТОРАН			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Назва		із зовнішніх джерел	
Адреса		із зовнішніх джерел	
Години роботи		із зовнішніх джерел	
Спеціалізація		із зовнішніх джерел	
Клас	Середній або вищий	із зовнішніх джерел	

Фрейми-спадкоємці містять всі слоти своїх батьків, вони явно прописуються тільки в разі зміни будь-якого параметра.

ОФІЦІАНТ (АКО **ЛЮДИНА)**			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Вік	Від 18 до 55 років	із зовнішніх джерел	
Стаж роботи		із зовнішніх джерел	
Зарплата		із зовнішніх джерел	
Графік роботи		із зовнішніх джерел	
Місце роботи	Фрейм-об'єкт	із зовнішніх джерел	

КЛІЄНТ (АКО **ЛЮДИНА)**			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Вид оплати	Готівка чи Картка	За замовчуванням (готівкові)	
Статус	Звичайний чи Vip	За замовчуванням (звичайний)	
Форма замовлення	Замовлення є чи ні	За замовчуванням (замовлення немає)	
Чайові		із зовнішніх джерел	

2. Фрейми-зразки описують конкретну ситуацію: які ресторани є в місті, як саме організовується відвідування, хто є відвідувачем, хто працює в обраному ресторані і т. п. Тому визначимо наступні фрейми-зразки, які є спадкоємцями фреймів-прототипів:

КАФЕ-РЕСТОРАН «СМАКОТА» (AKO РЕСТОРАН)			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Назва	Смакота	із зовнішніх джерел	
Адрес	м. Миколаїв, вулиця Мінаєва, 15	із зовнішніх джерел	
Години роботи	9:00-00:00	із зовнішніх джерел	
Спеціалізація	Піцерія	із зовнішніх джерел	
Клас	Середній або вищий	із зовнішніх джерел	

КАФЕ «СМАЧНА ЇЖА» (AKO РЕСТОРАН)			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Назва	Смачна їжа	із зовнішніх джерел	
Адрес	м. Миколаїв, вулиця Електронна, 5	із зовнішніх джерел	
Години роботи	9:00–00:00	із зовнішніх джерел	
Спеціалізація	Паб	із зовнішніх джерел	
Клас	Середній	із зовнішніх джерел	

СЕРГІЙ (AKO ОФІЦІАНТ)			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Вік	27	із зовнішніх джерел	
Стать	Чоловіча	із зовнішніх джерел	
Стаж роботи	5	із зовнішніх джерел	
Зарплата	7 000	із зовнішніх джерел	
Графік роботи	Через день з 18:00 до 00:00	із зовнішніх джерел	
Місце роботи	**КАФЕ «СМАЧНА ЇЖА»**	із зовнішніх джерел	

МАРИНА (AKO ОФІЦІАНТ)			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Вік	24	із зовнішніх джерел	
Стать	Жіноча	із зовнішніх джерел	
Стаж роботи	2	із зовнішніх джерел	
Зарплата	8 200	із зовнішніх джерел	
Графік роботи	Через день з 9:00 до 14:00	із зовнішніх джерел	
Місце роботи	**КАФЕ «СМАКОТА»**	із зовнішніх джерел	

ПЕТРО (AKO КЛІЄНТ)			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Стать	Чоловіча	із зовнішніх джерел	
Вік	19	із зовнішніх джерел	
Вид оплати	Готівка	За замовчуванням (готівкові)	
Статус	Звичайний	За замовчуванням (звичайний)	
Форма замовлення	Замовлення немає	За замовчуванням (замовлення немає)	
Чайові	7% від суми замовлення	із зовнішніх джерел	

3. Фрейми-ситуації описують можливі ситуації. У ресторані клієнт потрапляє в декілька типових ситуацій: замовлення і оплата. Можливі й інші не типові ситуації: клієнт подавився, у клієнта немає готівки для оплати рахунку і т. п. Розглянемо типові ситуації (їх може бути більше):

ЗАМОВЛЕННЯ			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Перелік страв		із зовнішніх джерел	IF-ADDED (змінює слот «Перелік цін»)
Перелік цін		Приєднана процедура	IF-ADDED (змінює слот «Сума замовник»)
Сума замовлення		Приєднана	
Прийняв замовлення	Фрейм-зразок	із зовнішніх джерел	
Зробив замовлення	Фрейм-зразок	із зовнішніх джерел	

ОПЛАТА			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Вид платежу		із зовнішніх джерел	IF-ADDED (змінює слот «Чайові»)
Чайові		приєднана	IF-ADDED (змінює слот «Сума замовник»)
Сплатив		приєднана процедура	
Замовлення	Фрейм-зразок	із зовнішніх джерел	IF-ADDED (змінює слот «Сплатив»)

4. Ситуації виникають після настання якихось подій, виконання умов та можуть слідувати одна за одною. Динаміку предметної області можна відобразити в фреймах-сценаріях. Їх може бути безліч, опишемо найбільш загальний і типовий сценарій відвідування ресторану:

ВІДВІДУВАННЯ РЕСТОРАНУ			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Відвідувач	Фрейм-об'єкт	із зовнішніх джерел	
Ресторан	Фрейм-об'єкт	із зовнішніх джерел	IF-ADDED, IF-REMOVED (Змінюють слот «Офіціант»)
Офіціант	Фрейм-об'єкт	Приєднана процедура (визначається за обраним рестораном)	
Сцена 1	Вхід, вибір	із зовнішніх джерел	
Сцена 2	Замовлення	із зовнішніх джерел	
Сцена 3	Їжа	із зовнішніх джерел	
Сцена 4	Оплата	із зовнішніх джерел	
Сцена 5	Вихід	із зовнішніх джерел	

5. Нехай в рамках нашого завдання Петро відвідав ресторан «Смачна їжа». Тоді фрейми буде заповнено так:

ВІДВІДУВАННЯ «Смачної їжі» (АКО **ВІДВІДУВАННЯ РЕСТОРАНУ)**			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Відвідувач	**Петро**	із зовнішніх джерел	
Ресторан	**КАФЕ «СМАЧНА ЇЖА»**	із зовнішніх джерел	IF-ADDED, IF-REMOVED (Змінюють слот «Офіціант»)
Офіціант	**Сергій**	Приєднана процедура (визначається за обраним рестораном)	
Сцена 1	Вхід, вибір	із зовнішніх джерел	
Сцена 2	**ЗАКАЗ ПЕТРА**	із зовнішніх джерел	
Сцена 3	Їжа	із зовнішніх джерел	
Сцена 4	**ОПЛАТА ПЕТРА**	із зовнішніх джерел	
Сцена 5	Вихід	із зовнішніх джерел	

ЗАМОВЛЕННЯ ПЕТРА (АКО **ЗАМОВЛЕННЯ)**			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Перелік Страв	Відбивна, темне пиво	із зовнішніх джерел	IF-ADDED (змінює слот «Перелік цін»)
Перелік цін	250, 75	Приєднана процедура	IF-ADDED (змінює слот «Сума замовник»)
Сума замовлення	325	приєднана	
Прийняв замовлення	**СЕРГІЙ**	із зовнішніх джерел	
Зробив замовлення	**ПЕТРО**	із зовнішніх джерел	

ОПЛАТА ПЕТРА (АКО **ОПЛАТА)**			
Ім'я слоту	**Значення слоту**	**Спосіб отримання значення**	**Демон**
Вид платежу	Готівка	із зовнішніх джерел	IF-ADDED (змінює слот «Чайові»)
Чайові	30	Приєднана процедура	
Сплатив	**ПЕТРО**	із зовнішніх джерел	
Замовлення	**ЗАМОВЛЕННЯ ПЕТРА**	із зовнішніх джерел	IF-ADDED (змінює слот «Сплатив»)

Взаємозв'язок різних видів фреймів відображається графічно у вигляді графа (рис. 9).

Використання фреймової моделі аналогічно семантичної, тільки в процесі отримання відповіді крім вершин враховуються і слоти. Наприклад, отримати відповідь на питання «Хто працює офіціантом в ресторані Смачна їжа?» Можна наступним чином: із запиту зрозуміло, що необхідно знайти фрейм «Ресторан «Смачна їжа» і простежити зв'язок з фреймом «Сергій», який є спадкоємцем фрейму «Офіціант».

Також можна знайти слот «Місце роботи» і перевіривши його значення у фреймах спадкоємців фрейму «Офіціант», визначити, що офіціантом в ресторані «Смачна їжа» працює Сергій.

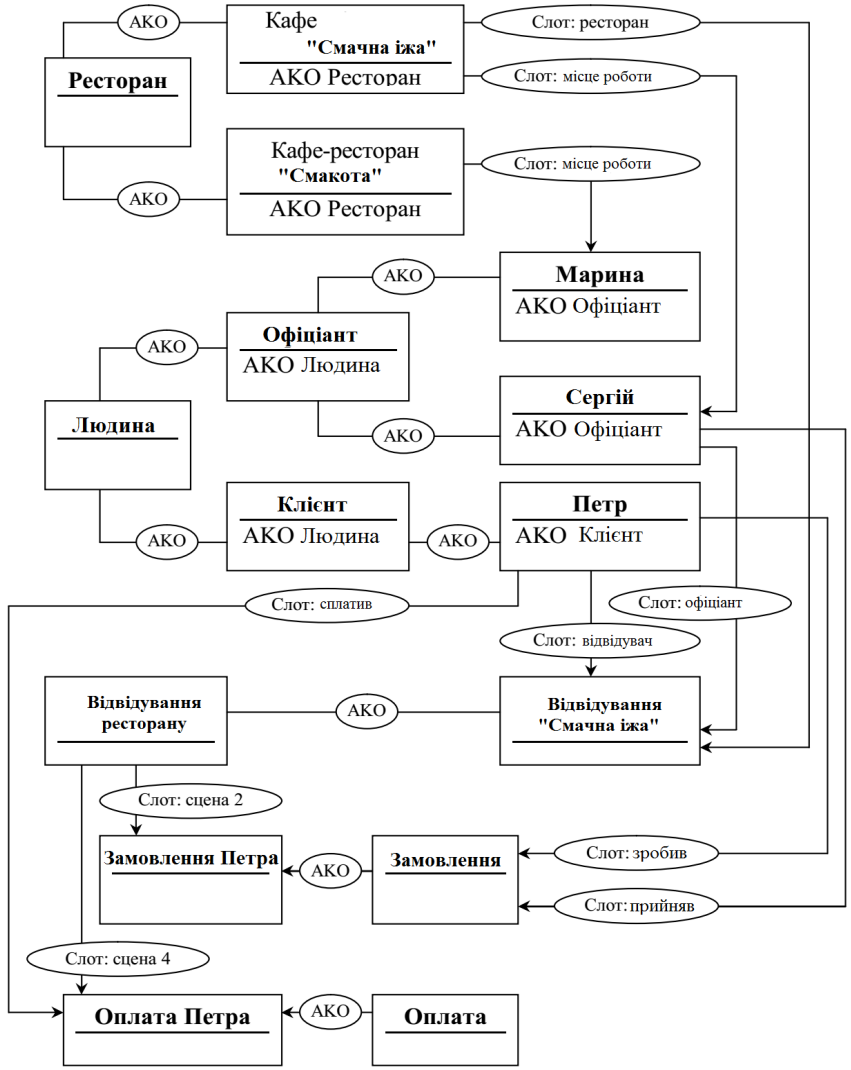


Рис. 9. Взаємозв'язок фреймів у вигляді графа

2.3. Завдання для самоконтролю

1. Що розуміється під представленням знань?
А) кодування інформації на будь-якій формальній мові
Б) знання, представлені в програмі на мові С ++
В) знання, представлені в підручниках з математики
Г) моделювання знань фахівців-експертів

2. Які поняття, представлені нижче, не є моделями представлення знань?
А) продукційні моделі
Б) фрейми
В) імітаційні моделі
Г) семантичні мережі
Д) формально-логічні моделі

3. Що являє собою семантична мережа?
А) мережевий графік, вершини якого – терміни виконання робіт
Б) нейронна мережа, що складається з нейронів
В) орієнтований граф, вершини якого – поняття, а дуги – відносини між ними

4. Який з основних типів відносин семантичної мережі, представлених нижче, може бути названий як АКО (A – Kind – Of)?
А) це
Б) елемент класу
В) бути спадкоємцем
Г) належить
Д) функціональний зв'язок

5. Чим відрізняються фрейми від семантичних мереж?
А) елемент моделі складається з множини незаповнених знань деяких атрибутів, іменованих «слотами»
Б) спадкування за АКО-зв'язками
В) елемент моделі – структура, що використовується для позначення об'єктів і понять

6. Що об'єднує семантичні мережі і фрейми?
А) організація процедури виведення
Б) успадкування властивостей
В) множині незаповнених значень деяких атрибутів, іменованих слотами

Г) структури, що використовуються для позначення об'єктів та понять

7. Які з виразів, представлених нижче, є структурною частиною фрейму?

А) значення *N*-го слота
Б) шаблон
В) примітивні типи даних

8. На якому формалізмі не засновані логічні моделі?

А) числення висловів
Б) силогізми Аристотеля
В) правильно побудовані формули
Г) нечіткі системи (fuzzy set)

2.4. Завдання для самостійної роботи

1. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Аеропорт» (диспетчерська).
2. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Залізниця» (продаж квитків).
3. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Торговий центр» (організація).
4. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Автозаправка» (обслуговування клієнтів).
5. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Автопарк» (пасажирські перевезення).
6. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Комп'ютерні мережі» (організація).
7. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Університет» (навчальний процес).

8. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Комп'ютерна безпека» (засоби і способи її забезпечення).
9. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Комп'ютерна безпека» (загрози).
10. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Інтернет-кафе» (організація і обслуговування).
11. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Розробка інформаційних систем» (ведення інформаційного проекту).
12. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Туристична агенція» (робота з клієнтами).
13. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Зоопарк» (організація).
14. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Кухня» (приготування їжі).
15. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Лікарня» (прийом хворих).
16. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Кінопрокат» (асортимент і робота з клієнтами).
17. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Прокат автомобілів» (асортимент і робота з клієнтами).
18. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Операційні системи» (функціонування).
19. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Інформаційні системи» (види та функціонування).

20. Побудувати продукційну, фреймову та семантичну модель (мережа) представлення знань в предметній області «Підприємство» (структура та функціонування).

2.5. Лабораторний практикум

2.5.1. Лабораторна робота 1. Вступ до мови Пролог. Бази даних на Пролозі

Мета: вивчити основи мови Пролог, розвинути навички роботи з середовищем програмування PIE, навчитися описувати найпростіші бази даних на Пролозі та ставити запити до них.

Теоретичні відомості

Мова Пролог є результатом багаторічних досліджень. Перша офіційна версія мови Пролог була розроблена у Франції, у Марсельському університеті Аланом Колмерое (Alain Colmerauer) на початку 70-х років 20 сторіччя як інструмент для ПРОграмування на принципах ЛОГіки. На сьогодні існує досить багато реалізацій Прологу. Найбільш відомі з них наступні: BinProlog, AMZI-Prolog, Arity Prolog, CProlog, Micro Prolog, МПролог, Prolog-2, Quintus Prolog, SICTUS Prolog, Silogic Knowledge Workbench, Strawberry Prolog, SWI Prolog, UNSW Prolog і т. п.

Традиційно під програмою розуміють послідовність операторів (команд, виконуваних комп'ютером). Цей стиль програмування прийнято називати *імперативним*. Програмуючи в імперативному стилі, програміст повинен пояснити комп'ютеру, *як* потрібно вирішувати завдання.

Протилежний йому стиль програмування – так званий *декларативний стиль*, у якому програма являє собою сукупність тверджень, що описують фрагмент предметної області. Програмуючи в декларативному стилі, програміст повинен описати, *що* потрібно вирішувати.

Відповідно й мови програмування поділяють на *імперативні* та *декларативні*. У свою чергу, найбільш важливими різновидами декларативного програмування, є функціональне і логічне (або реляційне) програмування. Пролог є мовою логічного програмування. До імперативних мов належать такі мови програмування, як Паскаль, Бейсік, Сі та ін.

Імперативні мови засновані на фон нейманівській моделі обчислень комп'ютера. Вирішуючи завдання, імперативний програміст спочатку створює модель у деякій формальній системі, а потім переписує розв'язання на імперативну мову програмування в термінах комп'ютера. Але, по-перше, для людини міркувати в термінах комп'ютера досить неприродно. По-друге, останній етап цієї діяльності (переписування розв'язання на мову програмування) по суті справи не має відношення до вирішення вихідного завдання.

В основі *декларативних мов* лежить формалізована людська логіка. Людина лише описує розв'язуване завдання, а пошуком рішення займається імперативна система програмування. У підсумку дістаємо значно більшу швидкість розробки програм, значно менший розмір вихідного коду, легкість запису знань на декларативних мовах, більш зрозумілі, у порівнянні з імперативними мовами, програми.

Декларативний стиль програмування зручний не для всіх задач. Пролог застосовується в таких сферах:

1. Автоматичний переклад.
2. Робота із природною мовою.
3. Проектування динамічних реляційних баз даних.
4. Експертні системи.
5. Автоматичне керування виробничими процесами.
6. Системи автоматичного проектування.
7. Символьні обчислення для розв'язання рівнянь, диференціювання, інтегрування.
8. Автоматичне доведення теорем.
9. Логічні задачі та ігри.
10. Пошук у графах і деревах.

Далі ми будемо вивчати Пролог і його застосування в штучному інтелекті за допомогою інтерпретатора Prolog Inference Engine (PIE). Сам цей інтерпретатор був написаний на Visual Prolog й успадкував деякі особливості даної реалізації Прологу. Проте, реалізована інтерпретатором мова ближче до класичного Прологу, і освоювати основи цієї мови на ньому простіше.

Простими об'єктами даних у Пролозі є *атоми*, *змінні* й *числа*.

Атом визначає конкретний об'єкт. Атом являє собою ланцюжок рядкових і прописних літер, цифр і спеціальних символів, що починається з малої літери. Якщо в програмі потрібно мати атом, що починається із

прописної літери, його треба взяти в лапки. Імена атомів не обов'язково повинні складатися тільки з латинських літер – Visual Prolog підтримує кирилицю. Ось кілька прикладів атомів:

```
віктор_петрович        x25        "Ferrari"
```

Змінні починаються з великої латинської літери (в інших реалізаціях Пролога і у компіляторі Visual Prolog змінні можуть починатися із символу «_»):

```
X Result P1
```

У тих випадках, коли потрібно вказати змінну, але її значення для нас несуттєво, можна використати так називану *анонімну змінну*, що складається з єдиного символу підкреслення «_».

Числа в Пролозі записуються стандартним способом:

```
6      -15
```

Інтерпретатор PIE не підтримує дійсних чисел. Це не являється істотним недоліком, оскільки Пролог у першу чергу призначений для обробки символьної інформації, а при символьній обробці необхідність у використанні дійсних чисел виникає нечасто.

Структури – складені об'єкти, що складаються з декількох компонентів, які, у свою чергу, теж можуть бути *структурами*. **Функтор** поєднує компоненти в одну *структуру*.

```
дата(4, травень, 1985)
точка(X, Y)
трикутник(точка(2, 3), точка(3, 2), точка(4, 1))
```

Тут `дата, точка, трикутник` – *функтори*.

Всі розглянуті об'єкти дані програми називаються **термами**.

Програми на Пролозі часто називають «базами даних» або навіть «базами знань» у тому розумінні, що вони являють собою сукупності речень, які визначають відношення між об'єктами предметної області або властивості цих об'єктів. Властивості й відношення в Пролозі, як і у логіці, називають предикатами. Предикати часто називають *цілями*.

Предикат визначається одним або декількома *реченнями*. Вся програма на Пролозі складається із речень.

Речення має вигляд

$A:-B_1,\ldots,B_n.$

Предикат `A` називається **заголовком** або **головою**, а предикати $B_1,\ldots,B_n$ складають **тіло** речення. Кожне речення закінчується крапкою.

Речення у Пролозі бувають трьох типів: факти, правила й питання.

Факт констатує, що між об'єктами виконане деяке відношення. Він складається тільки із заголовка. *Факт* являє собою безумовно істинне твердження.

```
столиця(україна, київ).
дата_народження(віктор_петрович, дата(4, травень, 1985)).
червоний("Ferrari").
```

Перше речення наведеної програми встановлює, що Київ є столицею України, тобто визначає, що об'єкти `україна` й `київ` знаходяться у відношенні «`столиця`».

Факт є найпростішим випадком *предиката*. У нашому прикладі столиця, `дата_народження` й `червоний` – імена предикатів. Предикат може мати кілька аргументів, що визначають об'єкти, між якими встановлюється відношення. Аргументами можуть бути конкретні об'єкти, або константи – *атоми*, *числа*, *структури*, а також *змінні*.

Число аргументів предиката називається його **арністю** або **містністю** (іноді її вказують після імені предиката: `столиця/2` – двомісний).

Правило – це речення, що встановлює відношення, виконуваність (істинність) якого залежить від істинності одного або декількох інших відношень.

```
розумний(X):-доктор_наук(X).
багатий(X):-має(X, "Ferrari").
щасливчик(X):-розумний(X), багатий(X).
```

Цю програму можна перекласти на українську мову так: дехто X є розумним, якщо він – доктор наук; X багатий, якщо в нього є Ferrari; X – щасливчик, якщо він і розумний, і багатий.

У нашій програмі змінна X зустрічається кілька разів у різних реченнях. Однак змінні з однаковими іменами в різних реченнях ніяк не пов'язані між собою, тобто областю дії змінної є одне *речення*. У різних *реченнях* може використовуватися одне ім'я змінної для позначення різних об'єктів.

Предикати в тілі речення можуть зв'язуватися за допомогою коми «`,`» або крапки з комою «`;`». Кома означає *кон'юнкцію* (логічне І) цілей (для істинності правила повинні бути істинними всі предикати з його

тіла), а крапка з комою – *диз'юнкцію* (логічне АБО) цілей (повинен бути істинним хоча б один предикат). Речення, у якому зустрічається крапка з комою, завжди можна розбити на кілька правил з однаковим заголовком.

Питання складається тільки з тіла. Питання використовують для з'ясування виконуваності деякого відношення між описаними в програмі об'єктами.

```
щасливчик(віктор_петрович),    дата_народження(віктор_
петрович, Date).
```

Отримавши питання, Пролог-система намагається встановити істинність відношень, зазначених у питанні, і видає відповідь – істинне твердження у питанні або хибне. Якщо питання містить змінні, система покаже всі можливі їхні значення, при яких відношення з питання виконуються.

Зазначимо, що крім предикатів, визначених через речення в програмі, системі відомі також **стандартні** (вбудовані) предикати. Деякі з них виконують такі дії, як робота з файловою системою, ввід з клавіатури та вивід на екран і т. ін. Ми вивчимо їх у міру необхідності.

Для чисел визначені такі арифметичні оператори:

```
+  -  /  *
```

div (цілочисленне ділення) mod (залишок від ділення)

is (присвоювання)

Предикати порівняння чисел:

```
>  <  >=  <=  <>  ><
```

(не дорівнює) =:= (дорівнює)

Арифметичні оператори діють подібно іншим предикатам, приймаючи істинне значення у випадку успішності порівняння або присвоювання. Всі перераховані оператори спочатку обчислюють значення своїх аргументів, тому змінні, які входять до виразів-аргументів, повинні дістати свої значення до виклику арифметичного предиката.

Нечислові терми можна порівнювати за допомогою предикатів «==» (еквівалентно) та «\==» (не еквівалентно).

Коментарі можуть займати один або кілька рядків. Багаторядкові коментарі повинні починатися із символів «/*», а закінчуватися символами «*/». Однорядкові коментарі починаються символом «%» і закінчуються з кінцем рядка.

Отже, у найпростішому випадку написання програми на Пролозі зводиться до опису властивостей об'єктів предметної області й відношень між ними, правил, за якими виводяться інші властивості й відношення; і постановці запитів до програми. Ми не описуємо, **як** шукати розв'язання завдання, а просто вказуємо, що потрібно знайти, тобто описуємо машині задачу. Саме ж розв'язання Пролог-система шукає сама.

З іншого боку, Пролог можна розглядати як комп'ютерну реалізацію формальної логіки. *Предикати* є ні що інше, як твердження, істинність або хибність яких встановлюється (виводиться) з інших тверджень або безперечно вірних *фактів* за допомогою *правил*. У свою чергу, *правила* являють собою звичайні логічні формули.

Розглянемо інтерфейс інтерпретатора PIE. Основне вікно програми містить меню й панель інструментів, що дозволяють виконувати стандартні операції над файлами і редагування текстів програм. Можлива одночасна робота з текстами декількох програм. У нижній частині основного вікна знаходиться вікно Dialog. Саме в ньому задаються *питання* до програми й у нього Пролог-система виводить свої відповіді та службові повідомлення.

Для того, щоб почати працювати з програмою (ставити до неї питання і отримувати відповіді), потрібно спочатку її завантажити в пам'ять. Завантаження проводиться автоматично при відкритті тексту програми або створенні нової. При цьому у вікні Dialog з'являється повідомлення

```
Reconsulted from: <ім'я файлу>
```

Це означає, що дана програма завантажена і можна ставити запити до неї. Можна завантажувати в пам'ять одночасно декілька програм.

Для того, щоб зміни, зроблені в програмі, набули чинності, необхідно перезавантажити програму командою Engine►Reconsult (F9). Цю команду слід виконувати після будь-яких змін у програмі перед її запуском. Не забувайте також, що при виконанні цієї операції повинне бути активно вікно з тією програмою, з якою збираєтеся працювати.

Для приведення системи в початковий стан (при якому жодна програма не міститься в пам'яті) слід виконати команду Engine►Reset.

Практична частина

Написати просту базу даних, що містить відомості про родинні відносини у такій сім'ї:

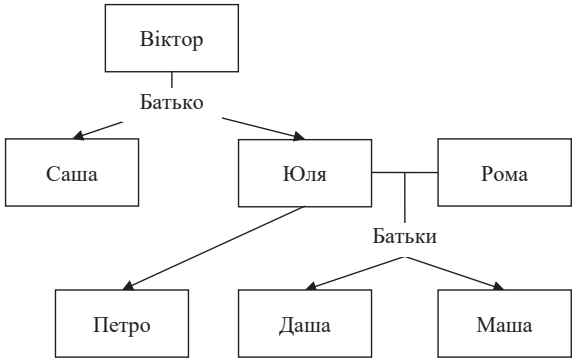


Рис. 10. Схема сім'ї

1. Відомий нам факт, що Рома є батьком (рос. Родитель) Даші, може бути записаний у вигляді:

```
батько(рома, даша).
```

Вся наша сім'я описується наступною програмою:

```
батько(рома, даша).
батько(юля, даша).
батько(рома, маша).
батько(юля, маша).
батько(юля, петро).
батько(віктор, юля).
батько(віктор, саша).
```

Запустіть інтерпретатор Пролога PIE, створіть новий файл (командою File►New (F7)) та напишіть текст програми. Збережіть створену програму на диску (File►Save as…), активізуйте вікно програми і завантажте її в пам'ять (Engine►Reconsult). Ви повинні побачити повідомлення у вікні Dialog про те, що програма завантажена. Якщо замість цього ви побачили повідомлення про помилку, то перевірте текст програми і спробуйте завантажити її знову.

2. Напишіть у вікні Dialog питання

```
батько(рома, даша).
```

і натисканням Enter запустіть програму на виконання. Система буде намагатися встановити істинність заданого твердження, зіставляючи його з відомими їй фактами. Знайшовши відповідний факт у програмі, інтерпретатор видасть:

```
True
1 Solution ,
```

що означає, що факт, зазначений нами в запиті, істинний, і Рома є батьком Даші.

Поставте також питання

```
батько(саша, даша).
```

Ви повинні отримати

```
No solutions ,
```

тобто факт, заданий нами для перевірки, невірний, і система не знайшла його в програмі. Таку ж відповідь отримаємо на запит

```
батько(паша, даша). ,
```

оскільки Паша в нас взагалі не згадується.

Для знаходження всіх батьків Маші запитайте

```
батько(X, маша). %X – в англійській розкладці!
```

Пролог-система видасть всі можливі значення X, при яких задовольняється поставлений запит (досягається поставлена ціль).

Аналогічно шукаємо дітей Віктора:

```
батько(віктор, X).
```

і взагалі всі відомі програмі відносини:

```
батько(X, Y).
```

Для знаходження спільних дітей Юлі і Роми запитайте

```
батько(юля, X), батько(рома, X).
```

Це питання означає: знайти всіх X, для яких батьками є і Юля, і Рома. Петро у відповідь інтерпретатора не потрапить.

Щоб отримати всіх братів і сестер Петра, випробуйте таке питання:

```
батько(X, петро), батько(X, Y). %відповідь одержимо на місці Y
```

Тут перша підціль знаходить батьків Петра, а друга – їхніх дітей, тобто братів і сестер Петра. Це речення можна розуміти й по-іншому: знайти всіх Y, у яких батько такий же, як і у Петра.

Щоб довідатися, чи є у Юлі діти, але не показувати їхніх імен, спитайте:

```
батько(юля, _).
```

Відповідаючи на запитання, що містить анонімну змінну, Пролог підбирає для неї значення, але не виводить його.

3. Доповніть нашу програму відомостями про стать кожної людини. Для цього додайте в її текст такі факти:

```
чоловік(віктор).
чоловік(рома).
чоловік(петро).
чоловік(саша).
жінка(юля).
жінка(даша).
жінка(маша).
```

Для того, щоб інтерпретатор врахував зроблені зміни, не забудьте виконати команду Engine►Reconsult (F9), активізувавши спочатку вікно з програмою. Ви повинні побачити повідомлення у вікні Dialog про те, що програма завантажена.

4. Припустимо, нам треба вміти знаходити маму будь-якого даної людини (наприклад, Маші). Бути матір'ю значить одночасно бути батьком даної людини і бути жінкою. Ось і запишемо ці умови як питання:

```
батько(X, маша), жінка(X).
```

Але можна зробити краще й визначити новий предикат мати, що буде задаватися *правилом*:

```
мати(X, Y):-батько(X, Y), жінка(X).
```

На українську мову це речення Пролога перекладається так: для всіх X та Y X є матір'ю Y, якщо X – батько Y та X – жінка. Додайте це правило в програму. Тепер ми можемо знаходити матерів будь-яких членів родини простим питанням на зразок

```
матір(X, даша).
```

Перевірте, що обидва способи повертають однакові результати.

Помітьте, що прологівські правила працюють і у зворотному напрямку. Завданням щойно написаного правила був пошук матерів заданих членів родини. Але ми можемо запитати й

```
матір(юля, Y). ,
```

і ми отримаємо дітей Юлі! Взагалі, практично завжди будь-який аргумент предиката може слугувати як вхідним параметром (при звертанні

до предиката аргументу задане конкретне значення), так і вихідним (передане змінна, котрій не надано ніякого значення); тобто в різних запитах ті самі аргументи можуть грати різні ролі. Ця властивість предикатів називається *оборотністю*.

Аналогічно визначається відношення тато:

```
тато(X, Y):-батько(X, Y), чоловік(X).
```

Додайте і його в програму. Помітьте, тепер у нашій програмі є два речення з однаковими іменами змінних X й Y. Проте, змінні з однаковими іменами в різних реченнях позначають різні об'єкти – областю дії змінної є одне речення.

Додайте в програму правило для знаходження спільних дітей:

```
спільна_дитина(Father, Mother, Child):-тато(Father, Child),
мати(Mother, Child).
```

Зверніть увагу, що в тілі правил можна використовувати не тільки факти, але й предикати, визначені за допомогою інших правил.

Далі додайте двомісний предикат `близький(X, Y)`, що означає, що дві людини є найближчими родичами – батьком і дитиною, братами, сестрами або братом і сестрою. Отже, цей предикат буде істинним, якщо X – батько Y, *або* Y – батько X, *або* у Y й X є спільний батько. Предикат не обов'язково визначається одним правилом:

```
близький(X, Y):-батько(X, Y).
близький(X, Y):-батько(Y, X).
близький(X, Y):-батько(Z, X), батько(Z, Y). %Z – спільний
батько
```

Спробуйте шукати близьких родичів і перевіряти членів родини на «близькість» різноманітними питаннями, наприклад, такими:

```
близький(петро, X).
близький(X, петро).
близький(віктор, юля).
близький(даша, X).
```

(У випадку з Дашею ми одержали Машу двічі. Тут спрацювало третє правило – один раз Маша була знайдена через Рому як батька Маші й Даші, другий раз – через Юлю).

Те ж відношення можна записати і по-іншому:

```
близький2(X, Y):-батько(X, Y); батько(Y, X); батько(Z, X),
батько(Z, Y).
```

Перевірте цей предикат на працездатність.

Збережіть програму, оскільки вона знадобиться у наступних лабораторних роботах.

Перевірити дію арифметичних предикатів.

1. Напишіть безпосередньо у вікні Dialog такі запити і перевірте відповіді програми:

```
   A is 1+3*4.
A = 13
 1 Solution
   A is 15 div 7.
A = 2
 1 Solution
   A is 15 mod 7.
A = 1
 1 Solution
   2+3 <= 6.
True
 1 Solution
   1+2 =:= 2+1.
True
 1 Solution
```

2. Перераховані предикати теж можна використовувати як підцілі в правилах. Створіть нову програму так само, як попередню, і запишіть такий предикат:

```
обчислити :- A is 2*2, B is A+3, A+B < 20. %обчислити – предикат без аргументів
```

Поставте запит до нього (попередньо завантаживши програму в пам'ять):

```
обчислити.
```

Одержимо, звичайно ж, істину.

Ще один приклад:

```
A+5 =:= 0.
No solutions
```

Пролог у цьому випадку не зможе підібрати значення `A`, оператор `=:=` вимагає, щоб всі змінні в операндах були *конкретизовані* (мали значення). Не забудьте, що областю дії змінної є одне речення, і тому

значення змінної треба присвоювати в тім же реченні, у якому робиться порівняння. Додайте до щойно створеної програми такі предикати:

```
   A is -5.
перевірка1:- A+5 =:= 0.
перевірка2:- A is -5, A+5 =:= 0.
```

Питання:

```
   перевірка1.
   No solutions
   перевірка2.
   True
   1 Solution
```

3. Додайте до цієї програми предикат `max/3`, що знаходить максимальне із двох чисел.

```
   max(X, Y, X):- X>=Y.
   max(X, Y, Y):- X<Y.
```

Тут максимум одержуємо на місці третього аргументу. У цих двох реченнях ми записали, що у випадку, якщо перше число більше другого, максимальним буде перше число, у зворотному випадку максимумом буде друге число. Перевірте такі питання до предиката max:

```
max(2, 3, X). %знаходження максимуму
max(2, 3, 2). %перевірка, чи є третє число найбільшим із двох перших
```

Завдання до лабораторної роботи 1

Завдання 1. Розширте родину довільно ще на кілька поколінь. Придумайте, які ще питання можна задати й перевірте їх.

Завдання 2.

Варіант 1

Напишіть правила, що визначають наступні відносини:

– дочка, син;
– сноха – дружина сина для отця.

Задайте питання для кожного нового відношення.

Варіант 2

Напишіть правила, що визначають наступні відносини:

– дочка, син;
– невістка – дружина сина для матері.

Задайте питання для кожного нового відношення.

Варіант 3

Напишіть правила, що визначають наступні відносини:

– дочка, син;
– сват – отець однієї людини з подружжя для батьків іншої.

Задайте питання для кожного нового відношення.

Варіант 4

Напишіть правила, що визначають наступні відносини:

– дочка, син;
– сваха – мати однієї людини з подружжя для батьків іншої.

Задайте питання для кожного нового відношення.

Варіант 5

Напишіть правила, що визначають наступні відносини:

– муж, дружина (виходячи з наявності спільних дітей);
– свекор – отець мужа.

Задайте питання для кожного нового відношення.

Варіант 6

Напишіть правила, що визначають наступні відносини:

– муж, дружина (виходячи з наявності спільних дітей);
– свекруха – мати мужа.

Задайте питання для кожного нового відношення.

Варіант 7

Напишіть правила, що визначають наступні відносини:

– муж, дружина (виходячи з наявності спільних дітей);
– тесть – отець дружини.

Задайте питання для кожного нового відношення.

Варіант 8

Напишіть правила, що визначають наступні відносини:

– муж, дружина (виходячи з наявності спільних дітей);
– теща – мати дружини.

Задайте питання для кожного нового відношення.

Варіант 9

Напишіть правила, що визначають наступні відносини:

– дочка, син;
– муж, дружина (виходячи з наявності спільних дітей);
– онук, бабуся.

Задайте питання для кожного нового відношення.

Варіант 10

Напишіть правила, що визначають наступні відносини:

– дочка, син;
– муж, дружина (виходячи з наявності спільних дітей);
– внучка, дід.

Задайте питання для кожного нового відношення.

Завдання 3.

Варіант 1

Створіть предикат, що знаходить максимум з трьох чисел.

Варіант 2

Створіть предикат, що має п’ять аргументів, і перевіряє, чи потрапляє точка, координати якої задані першими двома параметрами, у коло, центр якого визначають третій і четвертий параметр, а радіус – п’ятий.

Варіант 3

Створіть предикат, що знаходить абсолютне значення (модуль) числа.

Варіант 4

Створіть предикат, що одержує довжини сторін трикутника і перевіряє, чи є він прямокутним.

Варіант 5

Створіть предикат, що знаходить мінімум з трьох чисел.

Варіант 6

Створіть предикат, що одержує координати вершин трикутника і перевіряє, чи є він рівностороннім.

Варіант 7

Створіть предикат, що одержує довжини сторін трикутника і перевіряє, чи є він рівнобічним.

Варіант 8

Створіть предикат, що одержує координати кінців відрізка і перевіряє, чи є він горизонтальним.

Варіант 9

Створіть предикат, що одержує координати кінців відрізка і перевіряє, чи є він вертикальним.

Варіант 10

Створіть предикат, який перевіряє, чиє задане число парним.

Контрольні запитання

1. У чому полягає різниця між імперативним та декларативним способами програмування?
2. Для яких задач використовується мова програмування Пролог?
3. Якими об'єктами даних оперує Пролог?
4. Що таке структури на Пролозі? Як вони описуються? Наведіть власні приклади опису об'єктів за допомогою структур.
5. Які види речень ви знаєте? Опишіть, як записується кожен тип речення.
6. Що означають з точки зору формальної логіки і для чого використовуються різні види речень?
7. Що означає кома та крапка з комою у тілі речення?
8. Які існують предикати для роботи з числами?
9. Опишіть процес роботи з інтерпретатором PIE (завантаження, написання програм, постановка запитань до них).

2.5.2. Лабораторна робота 2. Семантичні моделі Прологу. Уніфікація. Відсікання. Рекурсія

Мета: вивчити семантичні моделі пролог-програм та навчитися аналізувати програми, використовуючи обидві моделі, вивчити роботу механізмів уніфікації, рекурсії, предиката відсікання, виробити навички їх застосування.

Теоретичні відомості

Семантику програми на Пролозі можна визначати з різних точок зору.

Розглядаючи програму як сукупність логічних тверджень, одержуємо *логічну* або *декларативну* семантику програми. Будь-який основний запит являє собою деяке твердження. Це твердження або виводиться із програми (тоді воно вважається істинним), або не виводиться (і значить хибне). Запит, що містить змінні, представляє собою твердження виду «існують такі змінні $x_1,...x_n$, що виконуються відношення, описані в програмі», що, знову ж, може бути істинним або помилковим. У першому випадку знайдуться деякі терми, при підстановці яких замість змінних одержимо істинне твердження. Таким чином, декларативний зміст програми визначає, чи є дана мета істинною, і якщо так, то при яких значеннях змінних вона перетворюється в істинне висловлення.

Операційна або *процедурна* семантика визначає процес пошуку відповіді. Запит розглядається як послідовність цілей, які треба виконати, а кожне правило описує, як виконати деяку мету.

У першій лабораторній роботі ми складали програми, ґрунтуючись тільки на декларативній семантиці. Однак програми, написані без врахування процедурної семантики, можуть виявитися дуже неефективними. Іноді досить поміняти місцями два правила або дві мети в правилі, щоб на порядок прискорити програму або зменшити вимоги до пам'яті. Тому далі розглянемо, як Пролог-система відповідає на питання.

Для того, щоб шукати значення змінних, Пролог використовує особливий механізм – *уніфікацію* або *співставлення* термів. Явно здійснити *уніфікацію* можна за допомогою оператора «=».

Уніфікація – це процес, на вхід якого подаються два терма, а він перевіряє, чи відповідають ці терми один одному. Загальні правила з'ясування, чи співставляються два терма S й T, такі:

– якщо S і T – константи, то вони співставимі, тільки якщо вони є тим самим об'єктом;

– якщо S – змінна, а T – довільний об'єкт, то вони співставимі, і S приписується значення T;

– якщо S і T – структури, то вони співставимі, тільки якщо мають однаковий функтор і всі їхні відповідні компоненти співставимі.

Якщо аргументи уніфікуються, то предикат «=» поверне істину. Обернений йому предикат «\=» повертає істину, лише коли уніфікація його аргументів неможлива.

Уніфікація є найбільш важливою концепцією Прологу. Її важливість полягає в тому, що за допомогою співставлення відбувається передача аргументів і повернення значень при виклику предикатів.

За допомогою уніфікації змінні можуть отримувати значення. Тут простежується цікава особливість уніфікації й прологівських змінних: змінна може одержати значення за допомогою уніфікації тільки один раз (у межах одного речення). Змінна, що ще не має значення, називається *вільною*. Змінна, що дістала певне значення, називається *конкретизованою*. Конкретизована змінна при уніфікації поводиться подібно константі – вона не може змінити свого значення.

Якщо уніфікуються дві вільні змінні, то ніяких значень вони при цьому не одержують, але між ними встановлюється зв'язок таким

чином, що надалі вони будуть виступати як синоніми. Такі змінні називають *зчепленими* або *зв'язаними*.

Розібравшись із уніфікацією, можна розглянути безпосередньо сам процес пошуку розв'язання завдання Пролог-системою.

Як відзначалося раніше, *питання* до програми є *мета*, до якої прагне система. Досягти мети – значить показати, що твердження, що містяться в питанні, *логічно* випливають із фактів і правил програми. Якщо запит містить змінні, то система повинна знайти такі їхні значення, які забезпечують досягнення мети. У випадку, якщо ціль може бути досягнута (відповідь на *питання* позитивна), говорять, що вона *досяжна* або *успішна*; у противному випадку – *недосяжна* або *має неуспіх*.

Опишемо процес пошуку відповіді на питання для загального випадку. Допустимо, система отримує запит з декількох підцілей, зв'язаних кон'юнкцією та/або диз'юнкцією:

$A_1, A_2, \ldots, A_N; B_1, \ldots, B_N; \ldots; Z_1, \ldots Z_N.$

Спочатку по черзі викликаються цілі A_n. Якщо вони *всі* виявляються успішними, то знайдено одне або кілька розв'язків. Як тільки одна з них виявиться недосяжною (при будь-яких значеннях змінних), Пролог вважає всю групу цілей A неуспішною (інші цілі не викликаються). У кожному разі система переходить до цілей B_n і т. д.

Для того, щоб досягти поточної мети, система переглядає програму від початку до кінця, намагаючись знайти речення, заголовок якого співставляється з метою. Якщо таке речення не знайдено, то ціль терпить неуспіх, у противному випадку Пролог уніфікує аргументи мети й голови знайденого речення. При уніфікації змінні можуть *конкретизуватися* або *зв'язуватися* з іншими змінними. Зв'язки не руйнуються і змінні не міняють значень, поки не відбудеться *повернення*.

Механізм *повернень* (бектрекінг, backtracking) є специфічним механізмом Прологу і дозволяє знаходити всі варіанти розв'язання завдання. Коли досягається поточна підціль, система запам'ятовує свій стан і створює *точку повернення* для того, щоб потім повернутися і перевірити інші варіанти розв'язання. Після розгляду головної мети до кінця (вона може завершитися успіхом або неуспіхом), Пролог повертається до найближчої точки повернення, і звертається повторно до предиката, на якому вона стояла.

Якщо зіставлення поточної мети відбувається з фактом, то ціль негайно досягається.

Якщо ж зіставлення відбувається із заголовком правила, то ціль досягається тільки тоді, коли буде досягнута кожна підціль у тілі цього правила. Цілі з тіла правила розглядаються так само, як цілі з питання. При цьому змінні-аргументи можуть вже мати значення або зв'язки з іншими змінними, отримані при уніфікації з поточною метою. Саме в такий спосіб відбувається передача параметрів у правило.

Якщо підціль не допускає зіставлення або більше немає цілей для розгляду (досягнута основна мета), то система здійснює *повернення* для спроби повторного співставлення попередньої підцілі. Змінні, які одержували значення при виклику попередньої підцілі, звільняються. При спробі повторного зіставлення система відновляє перегляд бази з речення, що безпосередньо слідує за тим, що забезпечувало зіставлення *мети* раніше.

Як правило, програми пишуться виходячи із процедурної семантики Пролога, а потім злегка прикрашаються з метою надати їм декларативного змісту. Але для великих програм краще застосовувати протилежний підхід: спочатку програма записується як набір чисто логічних тверджень, а потім переписується з урахуванням процедурної семантики. Часто модифікація програми являє собою переупорядкування послідовності правил або цілей у правилах. У той же час варто піклуватися про те, щоб не порушити логічну семантику.

Механізм перебору дозволяє Прологу знаходити всі варіанти вирішення задачі і звільняє програміста від необхідності писати такий механізм самому. Однак нічим не обмежений перебір може стати джерелом неефективності програми. Тому іноді потрібно його обмежити або виключити зовсім. Для цього в Прологу передбачений предикат «!» – *відсікання*. Він не має логічного змісту і завжди завершується успішно. Після того, як до нього дійшла черга, він встановлює «забір», що не дає «відкотитися назад», щоб вибрати альтернативні розв'язки для вже «спрацювавших» підцілей. Тобто для тих, які розташовані лівіше відсікання. На цілі, розташовані правіше, відсікання не впливає. Крім того, відсікання відкидає всі речення предиката, розташовані після речення, у якому знаходиться відсікання. Таким чином, для підцілей, які розглядалися до відсікання, буде отримане одне рішення (знайдене першим).

Відсікання часто порушують відповідність між декларативним і процедурним змістом Пролог-програми, і їх слід використовувати з обережністю.

За допомогою відсікання в Пролозі можна змоделювати таку конструкцію імперативних мов, як розгалуження. Процедура

```
S :- <умова>, !, P.
S :- P2.
```

буде відповідати оператору `if <умова> then P else P2`. Якщо умова буде істинною, Пролог викличе мету P, а відсікання не дасть викликати S у другий раз і знайти друге рішення. Якщо ж умова хибна, то Пролог не дійде до відсікання, а відразу буде вважати перше правило неуспішним і перейде до альтернативного розв'язку для S – P2.

Часто буває потрібно мати предикат, що відповідає логічному запереченню. Для цієї мети служить вбудований предикат `not`. Ціль `not(Goal)` істинна, якщо `Goal` не істинна, і навпаки. `not` неявно визначений у такий спосіб:

```
not(P):- P, !, fail; true.
```

`fail` – вбудований предикат, що завжди терпить неуспіх й ініціює повернення, `true` завжди завершується успішно. Замість `not(Goal)` можна писати `not Goal`.

Завдяки тому, що not визначається за допомогою відсікання, він іноді може повертати несподівані (на перший погляд) відповіді, тому застосовувати цей предикат треба з обережністю.

Одним з потужних засобів керування ходом виконання програми на Пролозі є *рекурсія*. На відміну від імперативних мов програмування, *рекурсія* для Прологу є основним прийомом програмування.

Рекурсивна процедура містить у своєму визначенні сама себе, тобто предикат, що стоїть в лівій частині правила, входить до підцілей у правій частині правила. Рекурсивний предикат буде викликати сам себе доки не виконається умова виходу з рекурсії (якщо, звісно, програміст не забув її записати).

Як правило, для того, щоб означити рекурсивне відношення, треба спочатку визначити *базис* рекурсії – початкову ситуацію або ситуацію, на якій рекурсія повинна зупинитися. Зазвичай базис описує найпростіший випадок, за якого відповідь можна отримати одразу без рекурсії.

Крім того треба записати *шаг* рекурсії – власне рекурсивне правило, яке перед викликом самого себе може містити *умову виходу* з рекурсії. Параметри, з якими викликається рекурсивний предикат, повинні змінюватися у цьому правилі для того, щоб програма не зациклилась і спрацював або *базис*, або *умова виходу*.

Вид рекурсії, коли тіло правила починається з рекурсивного виклику означуваного предиката, називається *лівосторонньою рекурсією*. Лівостороння рекурсія часто приводить до зациклення програми, тому потрібно намагатися, якщо можливо, уникати її використання, на відміну від *правосторонньої* або *хвостової рекурсії*. Для її здійснення рекурсивний виклик означуваного предиката повинен бути останньою підметою в тілі рекурсивного правила й до моменту рекурсивного виклику не повинно залишитися точок повернення (неперевірених альтернатив). Тобто в підцілей, розташованих лівіше рекурсивного виклику предиката, не повинно залишатися якихось неперевірених варіантів й у предиката не повинно бути речень, розташованих після рекурсивного правила.

Практична частина

Перевірити дію предиката уніфікації «=».

1. Запустіть інтерпретатор та спробуйте задати такі питання:

```
1 = 1.
True
 1 Solution
 a = b.
No solutions
 1 + 2 = 1 + 2.
True
 1 Solution
 2 + 1 = 1 + 2.
No solutions
```

При уніфікації основних термів вони просто перевіряються на тотожність і, залежно від результату, повертається відповідь «так» або «ні».

Помітьте, вирази вигляду «`1 + 2`» теж є термами, і тому останнє зіставлення завершилося невдачею. Таке дивне поводження Прологу

пояснюється тим, що він розроблявся насамперед як мова для обробки символьної інформації. Для обчислення значень виразів потрібно використовувати предикат is.

2. Перевірте відповіді системи на наступні запити:

```
X = 1.
X = 1
1 Solution
2 + 3 = X.
X = 2 + 3
1 Solution
```

Якщо терми містять змінні, робиться спроба присвоїти значення змінним так, щоб обидва терми стали ідентичними. Якщо таке означування змінних можливе, уніфікація успішна і значення змінних видаються як відповідь.

Як видно, змінні можуть знаходитися як ліворуч, так і праворуч від знака «=».

3. Перевірте відповіді системи на наступні запити:

```
X = 1, X = 1.
X = 1
1 Solution
X = 1, X = 2.
No solutions
```

У першому прикладі змінна X спочатку співставляється з 1 і дістає значення 1, потім це значення успішно зіставляється з 1 (1 = 1). У другому прикладі X отримує значення 1, яке потім зіставляється з 2 (тобто фактично робиться спроба уніфікації 2 = 1), що, звичайно, завершується невдачею.

Можливо одночасне означування декількох змінних:

```
X + 2 = 3 + Y.
X = 3, Y = 2
1 Solution
```

Будь-яка змінна співставляється сама із собою і не отримує при цьому ніякого значення:

```
X = X.
X = X$0
1 Solution
```


Зіставлення двох різних змінних завжди успішне.

```
X = Y.
X = X$1, Y = X$1
1 Solution
```

Зверніть увагу, що «значення» в X та Y те саме – вони є *зв'язаними*.

```
X = Y, Y = Z, Z = a.
X = a, Y = a, Z = a
1 Solution
```

Тут X зв'язується з Y, Y зв'язується з Z і, нарешті, Z отримує значення «a». Після цього всі три змінні мають те саме значення «a».

Вивчити процес досягнення мети у програмі про родинні відносини, побудувати дерево виводу.

1. Закрийте всі відкриті у вікні інтерпретатора програми, ініціалізуйте систему (Engine►Reset). Відкрийте програму про сімейні відносини, яку ви створили в першій роботі. Розглянемо процес знаходження відповіді на поставлене питання. Задайте цій програмі таке питання:

```
батько(рома, даша).
```

Отримавши цю мету, Пролог-система починає шукати в програмі відношення, голова яких уніфікується з метою. Єдиним таким реченням є *факт* `батько(рома, даша)`. Знайшовши його, система буде вважати мету досягнутою і сповістить про це.

Далі запитайте:

```
батько(X, маша).
```

Із цією метою уніфікуються вже два факти. При зіставленні по черзі з кожним з них X одержує значення `рома` і `юля`.

Подумайте самі, як шукається відповідь на запит `батько(X, Y)`.

2. Наступне питання було таким:

```
батько(юля, X), батько(рома, X).
```

Тут ми маємо дві підцілі, розділені комою. Нагадаємо, що кома означає кон'юнкцію цілей (для досягнення головної мети всі її підцілі повинні бути успішними), а крапка з комою – диз'юнкцію (для досягнення головної мети хоча б одна з підцілей повинна бути успішною). Пролог буде намагатися досягти кожної з підцілей по черзі в порядку

їхнього слідування. Розглядати цей процес зручно в режимі трасування. Він перемикається командою Engine►Trace Calls (Ctrl+T). Увімкніть його й запустіть програму з даним питанням. У вікні Dialog ви побачите, як саме виводиться відповідь. Розглянемо послідовно весь процес:

```
Trace: >> CALL: батько(юля,X$2)
Trace: >> RETURN: батько(юля,даша)
```

Спочатку викликається (CALL) перша підмета `батько(юля,X)`. Першою знайденою відповідністю для неї (RETURN) буде `батько(юля,даша)`. При цьому X одержить за допомогою уніфікації значення `даша`. Знайдене речення є *фактом*, тому Пролог вважає першу підмету успішною. Якби це речення була *правилом*, система уніфікувала б (*зв'язала* або *конкретизувала*) відповідні змінні з поточної мети й заголовка правила і почала б виклик цілей з тіла правила. Розгляд правил схожий на розгляд питань, тільки в результаті програма досягає (або не досягає) не головної мети програми, а поточної підцілі.

```
Trace: >> CALL: батько(рома,даша)
```

Перша підмета досягнута, Пролог переходить до другої, котра виглядає тепер так: `батько(рома,даша)`, оскільки X вже конкретизована.

```
Trace: >> RETURN: батько(рома,даша)
X = даша
```

Знайдено відповідність другій підцілі. Тепер всі підцілі досягнуті, отже, і головна мета успішна. Пролог виводить знайдені значення змінних.

```
Trace: >> REDO: батько(рому,даша)
```

Тут вступає в дію механізм повернень. Досягнувши поточної підмети (при трасуванні в цьому місці ми бачимо RETURN), система створює точку повернення. Після розгляду головної мети до кінця Пролог повертається до найближчої точки повернення, і викликає повторно (REDO) ціль

```
батько(рома,даша).
Trace: >> FAIL: батько(рома,даша)
```

Інших речень, що відповідають цій меті (крім вже розглянутого), немає, тому вона цього разу терпить неуспіх (FAIL).

```
Trace: >> REDO: батько(юля,даша)
```


Відкат до першої точки повернення: `батько(юля,X$2)`. Змінна X звільнюється.

```
Trace: >> RETURN: батько(юля,маша)
```

Знайдено друге рішення для першої підцілі – X = маша.

```
Trace: >> CALL: батько(рома,маша)
Trace: >> RETURN: батько(рома,маша)
X = маша
```

Знову основна мета досягнута.

```
Trace: >> REDO: батько(рома,маша)
Trace: >> FAIL: батько(рома,маша)
Trace: >> REDO: батько(юля,маша)
Trace: >> RETURN: батько(юля,петро)
```

Після звернення втретє до першої підмети `батько(юля,X$2)` знайдена третя дитина Юлі.

```
Trace: >> CALL: батько(рома,петро)
Trace: >> FAIL: батько(рома,петро)
```

Однак для третьої дитини друга підціль відразу терпить неуспіх, основна мета не досягається.

```
Trace: >> REDO: батько(юля,петро)
Trace: >> FAIL: батько(юля,X$2)
2 Solutions
```

Після третього звертання до першої підцілі всі діти Юлі вже знайдені, тому четвертий виклик неуспішний. Всі можливі альтернативи розглянуті, виконання програми завершується.

3. Перераховані кроки досягнення мети можна представити у вигляді *дерева виводу* (пунктиром показані *повернення*):

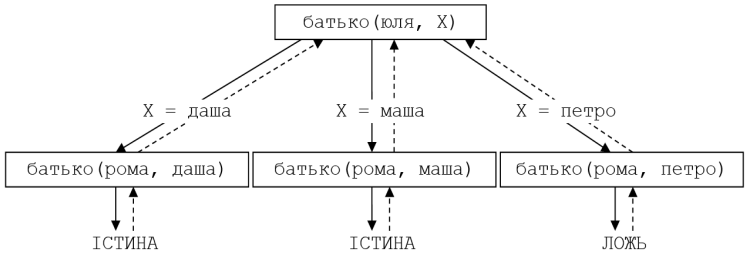


Рис. 11. Дерево виводу

Перевірити дію предиката відсікання.

1. Вимкніть режим трасування (ще раз виконайте команду Engine►Trace Calls), і задайте програмі таке питаня:

```
батько(X, даша), !, батько(X, Y).
X = рома, Y = даша
X = рома, Y = маша
2 Solutions
```

Без відсікання таке питання поверне п'ять варіантів відповіді (перша підціль знайде обох батьків Даші, друга – дітей кожного з них; діти будуть повторюватися). Завдяки відсіканню першою підметою буде знайдений тільки Рома (ціль `батько(X, даша)` не буде викликана повторно). Після відсікання перебір продовжується як звичайно – знайдені двоє дітей Роми.

2. За допомогою відсікання можна спростити предикат max:

```
max(X, Y, X):- X>=Y.
max(X, Y, Y):- X<Y.
```

Якщо перша умова (`X>=Y`) буде виконана, то система все одно викличе предикат max вдруге і перевірить умову `X<Y`, хоча очевидно, що це не так. Якщо ж X<Y, то після неуспішного завершення першого речення однаково ще раз перевіряється очевидне співвідношення X<Y. Підвищимо ефективність:

```
max(X, Y, X):- X>=Y, !.
max(_, Y, Y).
```

Тепер, якщо X>=Y, то відсікання не дасть Прологу розглянути альтернативне друге речення. Друге речення «спрацює» тільки у випадку, якщо умова виявиться помилковою – до відсікання Пролог не дійде. У цій ситуації в третій аргумент потрапить те ж значення, що знаходилося в другому аргументі. Зверніть увагу, що в цьому випадку нам уже не важливо, чому дорівнював перший аргумент, і його можна замінити анонімною змінною.

Створіть нову програму, запишіть в ній нову версію предиката max і перевірте його роботу.

Зауваження. Другий варіант предиката max не можна використовувати, якщо третій аргумент – константа або конкретизована змінна. Спробуйте, наприклад, запитати: `max(4, 3, 3)`. Подумайте, чому Пролог видає неправильну відповідь. Розгляньте також такі питання і спробуйте пояснити відповіді Пролог-системи:

```
not чоловік(ігор).
not чоловік(X).
X = маша, not чоловік(X).
```


Створити рекурсивний предикат предок. *Створити програму для обчислення факторіалу.*

1. Припустимо, нам потрібно визначити, чи не є дві людини з нашої бази даних віддаленими родичами через кілька поколінь. Запрограмуємо для цього відношення `предок/2`. Очевидно, що X буде предком Y, якщо X – батько Y:

```
предок(X, Y):-батько(X, Y). %базис рекурсії - найпростіший випадок
```

Крім того, якщо X – батько Z, і відомо, що Z є предком Y, то X теж буде предком Y – запишемо друге речення нашого предиката:

```
предок(X, Y):-батько(X, Z), предок(Z, Y). %шаг рекурсії - рекурсивний виклик
```

Завдяки використанню рекурсії написаний предикат може шукати як завгодно далеких предків і нащадків по генеалогічному древу.

Додайте ці два речення до нашої програми і збережіть її. Поставте запити до нового предикату: перевірте, чи є Віктор предком Даші, Маші та Роми, знайдіть всіх предків Петра.

Зауваження. Якщо поміняти місцями два речення предиката предок або дві підцілі в другому реченні, то може знизитися ефективність роботи програми, а в деяких випадках система може ввійти в нескінченний цикл (у цьому можна переконатися, побудувавши дерева виводу для різних варіантів предиката). У Пролозі важливий порядок речень і підцілей у реченнях, тому завжди варто перевіряти, як працює програма із процедурної точки зору.

2. Розглянемо завдання знаходження факторіала деякого цілого додатного числа. За означенням,

0! = 1 (Факторіал нуля є одиниця)
1! = 1
2! = 1 2
3! = 1 2 3
...
n! = 1 2 3 ... n
Зауважимо, що n! = (n-1)! · n

Тобто для того, щоб знайти факторіал деякого числа n, необхідно знайти факторіал від попереднього числа, а потім отримане значення помножити на дане n.

Створіть і збережіть таку програму:

```
fact(0,1):-!. %факторіал нуля дорівнює одиниці
fact(N,F):-
     N1 is N-1,
     fact(N1,F1), /* F1 дорівнює факторіалу числа,
                  на одиницю меншого вихідного
                  числа */
     F is F1*N. /* факторіал вихідного числа дорівнює
                  добутку F1 на саме число */
```

Запит треба ставити в такому вигляді:

```
fact(n, F). ,
```

де замість n треба підставити задане число, а на місці F отримаємо його факторіал.

3. Простежте в режимі трасування, як відбувається обчислення факторіала. При прямому ході рекурсії (поки N>0) мета fact викликає сама себе із всі меншими значеннями N. Як тільки N стає рівним 0, ціль `fact` повертає 1 і більше себе не викликає – починається зворотний хід рекурсії. Тепер відбувається повернення по ланцюжку предикатів `fact` у зворотному порядку з обчисленням факторіалів для всіх чисел від 0 до n. Відсікання в першому правилі необхідно для того, щоб дійшовши до нього, Пролог не викликав друге (як можливу альтернативу для мети `fact(0,1).`) – інакше це речення нескінченно викликало б себе й далі з негативними значеннями N.

Розглянута програма не вміє знаходити число по його факторіалу. Якщо N буде вільною змінною, то перший же виклик предиката `fact` зазнає невдачі, оскільки is не зможе обчислити число, що передує N, якщо N не задано. Напишемо іншу версію, використовуючи накопичувачі – змінні, у яких накопичуються деякі значення. У нашому випадку в Y – поточне значення факторіала, а в M – число, для якого він обчислений. Ідея методу полягає в припиненні накопичення при досягненні заданого значення (самого числа або факторіала). При цьому невідомий параметр уніфікується зі значенням відповідного накопичувача.


```
fact2(N, X, N, X):-!. /*зупиняємо рекурсію, коли третій
                        аргумент дорівнює першому*/
fact2(N, X, M, Y):-
     M1 is M+1, %M1 – наступне натуральне число після числа M
     Y1 is Y*M1, %Y1 – факторіал M1
     fact2(N, X, M1, Y1). /* рекурсивний виклик з новим
                          натуральним числом M1 і відповідним
                          йому обрахованим факторіалом Y1 */
```

Перші два параметри – число і його факторіал, другі – початкові значення. Для зручності можна додати в програму правило, що буде задавати початкові значення накопичувачів за нас:

```
fact2(N, X):-fact2(N, X, 0, 1).
```

Додайте в щойно створену програму нову версію предиката для знаходження факторіалу і спробуйте шукати факторіали деяких невеликих чисел та числа за їх факторіалами за допомогою обох версій предиката.

Завдання до лабораторної роботи 2

Завдання 1.

Задайте кілька питань із предикатом предок до розширеній вами бази даних з першої лабораторної роботи. Побудуйте дерево виводу для одного з них.

Завдання 2.

Виконайте наступні завдання, використовуючи рекурсію.

Варіант 1

Створіть предикат, що знаходить ступінь натурального числа.

Варіант 2

Створіть предикат, що знаходить суму натуральних чисел від 1 до N.

Варіант 3

Створіть предикат, що знаходить суму непарних чисел, які не перевищують N.

Варіант 4

Створіть предикат, що знаходить суму парних чисел, які не перевищують N.

Варіант 5

Створіть предикат, що знаходить добуток парних чисел, які не перевищують N.

Варіант 6

Створіть предикат, що знаходить добуток непарних чисел, які не перевищують N.

Варіант 7

Створіть предикат для знаходження N-го члену арифметичної прогресії із першим членом A і різницею D.

Варіант 8

Створіть предикат для знаходження N-го члену геометричної прогресії із першим членом B і знаменником Q.

Варіант 9

Створіть предикат, який знаходить N-ий член послідовності, перший член якої дорівнює 1, а кожний наступний визначається формулою $x_n=n\cdot x_{n-1}$.

Варіант 10

Створіть предикат для знаходження N-го числа ряду Фібоначчі. Перші два члени ряду Фібоначчі дорівнюють одиниці, а кожен наступний дорівнює сумі двох попередніх.

Контрольні запитання

1. Що являє собою програма з декларативної точки зору?
2. Що являє собою програма з процедурної точки зору?
3. Наведіть правила, за якими здійснюється уніфікація термів.
4. Що таке вільні, конкретизовані та зв'язані змінні?
5. У якій послідовності розглядаються підцілі головної мети?
6. Як перевіряється успішність поточної мети?
7. Як діє механізм повернень?
8. Як діє предикат відсікання?
9. Як на Пролозі реалізується розгалуження та логічне заперечення?
10. Що таке рекурсія?
11. Які види рекурсії вам відомі? Чим вони відрізняються?

2.5.3. Лабораторна робота 3. Робота зі списками

Мета: вивчити поняття списку, одержати і закріпити навички розробки та застосування основних процедур роботи зі списками.

Теоретичні відомості

У Пролозі основною складеною структурою даних є *список*.

Список – це послідовність довільної кількості елементів. *Список* записується у квадратних дужках:

```
[1, україна, дата(20, жовтень, 2007), Y]
```

Як бачите, *елементами* списку можуть бути будь-які терми. Список сам по собі також є термом і тому може слугувати елементом іншого списку. Елементи можуть повторюватися, їх порядок важливий – списки, які відрізняються порядком елементів, не еквівалентні. Допускається уніфікація списку зі змінними або іншими термами:

```
L = [[січень, березень, травень, липень, серпень, жовтень,
грудень], [квітень, червень, вересень, листопад], лютий]
```

Список може зовсім не містити елементів – *порожній список*:[]

Для обробки списків їх зручно розглядати як рекурсивну структуру. З цієї точки зору *список* – це структура даних, яка або порожня, або складається з *голови* та *хвоста*. *Голова* – це один або декілька початкових елементів списку, розділених комами, *хвіст* у свою чергу також є *списком* з останніх елементів основного списку. Такий спосіб визначення списків реалізується за допомогою операції «|»:

```
[Head|Tail]
```

Head – голова списку, Tail – хвіст. Наступні приклади ілюструють різні способи запису одного і того самого списку:

```
[a, b, c]  [a|[b, c]]  [a, b|[c]]  [a, b, c|[]]  [a|[b|[c]]]
```

Використання списків разом із застосуванням рекурсії є дуже потужним засобом розв'язування задач на Пролозі.

Напишемо предикати, які будуть виконувати основні операції над списками.

Визначимо відношення приналежності елемента списку. Предикат `належить(X, L)` буде перевіряти, чи входить X у список L. Якщо заданий елемент належить списку, то він або є його головою, або належить хвосту списка:

```
належить(X, [X|_]).
належить(X, [_|Tail]):-належить(X, Tail).
```

Якщо список починається з X, то одразу спрацює перше речення (факт) – предикат буде істинним. Інакше Пролог перейде до другого речення (правила), яке рекурсивно викличе предикат `належить` для пошуку у останній частині списку. При другому виклику буде передано не весь список, а лише його хвіст без першого елементу. Отже,

предикат `належить` буде викликати сам себе, відкидаючи кожен раз перший елемент, доки не закінчиться список. Якщо заданий елемент міститься у вихідному списку, то настане такий момент, коли він стане першим у списку, і тоді ціль стане успішною. У будь-якому разі Пролог перевірить весь список, і тому знайде всі входження X в нього.

Якщо треба знайти єдиний розв'язок (лише перше входження X), то потрібно скористатися відсіканням:

```
належить2(X, [X|_]):-!.
належить2(X, [_|Tail]):-належить2(X, Tail).
```

Як тільки буде знайдений перший X (тобто спрацює перше правило), предикат відсікання зупинить перебір.

Для зчеплення (конкатенації) двох списків в один визначимо відношення `конк(L1, L2, L3)`, де L3 – список, отриманий злиттям L1 та L2. Якщо перший список порожній, то L3 співпадає з L2. Інакше для того, щоб дописати елементи L2 до L1, треба зчепити хвіст L1 та весь L2, а потім до отриманого списку попереду приписати голову L1:

```
конк([], L, L).
конк([X|Tail1], L, [X|Tail2]):-конк(Tail1, L, Tail2).
```

Додавання елемента X у список L виконується дуже просто – достатньо записати результуючий список як `[X|L]`. Цю дію також можна оформити у вигляді предиката:

```
додати(X, L, [X|L]).
```

Предикат для видалення елемента зі списку також реалізується за допомогою рекурсії:

```
видалити(X, [X|Tail], Tail). %якщо X – голова списку, то
результатом видалення буде хвіст цього списку
видалити(X, [Y|Tail1], [Y|Tail2]):-видалити(X, Tail1, Tail2).
%інакше видалятимемо X у хвості списку
```

Тепер навчимо Пролог визначати, чи є один список підсписком другого (враховуючи порядок елементів). Якщо ми додамо до деякого списку L1 список S, а до них – ще деякий список L2 і у результаті отримаємо L (L1 + S + L2 = L), то S буде підсписком L. Запишемо цю ідею:

```
підсписок(S, L):- конк(Temp, L2, L), конк(L1, S, Temp).
```

Зауваження. Оскільки самі значення L1 і L2 для нас неважливі, для них слід було б використати анонімні змінні. Зараз ми так не зробили лише з міркувань зручності для пояснення роботи предиката.

Предикат `конк` можна використовувати ще для багатьох задач. Наприклад, можна записати даний список у зворотному порядку. Оберненим для порожнього списку буде порожній список – це буде базис рекурсії. А для того, щоб обернути непустий список, треба обернути його хвіст, а потім дописати до нього справа перший елемент вихідного списку – це буде шаг рекурсії.

```
обернений([], []).
обернений([X|Tail], L):-обернений(Tail, L1), конк(L1, [X], L).
```

Часто потрібно перевірити, чи повторяються деякі елементи в заданому списку. Запрограмуємо предикат всі_різні, який буде істинним, коли в списку немає повторів.

```
всі_різні([]).
всі_різні([X|Tail]):-not належить(X, Tail), всі_різні(Tail).
```

Тут ми перевіряємо, чи не зустрічається перший елемент в хвості списку, а потім аналогічно перевіряємо хвіст.

Для знаходження всіх перестановок заданого списку спочатку знайдемо перестановку хвоста списку, а потім внесемо перший елемент заданого списку в довільну позицію знайденої перестановки хвоста:

```
перестановка([], []).
перестановка([X|L], P):-перестановка(L, L1), видалити(X, P, L1).
```

Тут L1 – перестановка хвоста заданого списку, а предикат видалити вносить елемент X в список L1 і записує результат в список P. Цей предикат не можна використовувати, передаючи йому в якості першого аргументу неконкретизовану змінну через небезпеку нескінченного циклу.

Нарешті, часто треба знаходити кількість елементів списку:

```
довжина([], 0).
довжина([_|Tail], N):-довжина(Tail, N1), N is N1+1.
```

Цей предикат дуже простий, і ви можете самостійно зрозуміти принцип його роботи.

Для деяких задач потрібно вміти сортувати списки. Ми розглянемо один з досить ефективних методів сортування – так зване «швидке сортування». Його сутність полягає у наступному. Обирається деякий «бар'єрний» елемент, відносно якого вихідний список розбивається на два підсписки. У перший потрапляють елементи, які менші за бар'єрний, у другий – більші або рівні йому. Обидва підсписки сортуються

тим самим способом, після чого до вже відсортованого першого підсписку (з елементами, меншими за бар'єрний) дописується сам бар'єрний елемент, а потім – відсортований другий підсписок. В кінці-кінців отримаємо вихідний список, елементи якого розташовані у потрібному порядку.

Розділяти список буде предикат розділити/4. Його перший аргумент – список, що підлягає розбиттю, другий – бар'єрний елемент, третій і четвертий – підсписки, на які розбивається даний список.

```
розділити([], _, [], []). %базис рекурсії
розділити([X|Tail1], Y, [X|Tail2], Greater):-
     X<Y, !, розділити(Tail1, Y, Tail2, Greater). /*якщо X
                          менше бар'єрного елемента Y,
                          то віднесемо його до першого
                          підсписку і продовжимо
                          розбиття*/
розділити([X|Tail1], Y, Lesser, [X|Tail3]):-
     розділити(Tail1, Y, Lesser, Tail3). /*інакше – до
          другого підсписку і також рекурсивно
          продовжимо розбиття*/
```

Запишемо предикат, який і буде виконувати сортування за описаним принципом, беручи у якості бар'єрного перший елемент списку:

```
сорт([], []). %пустий список вважаємо вже відсортованим
сорт([X|Tail], Sorted):-
     розділити(Tail, X, Lesser, Greater), /*розділимо
вихідний список на два відносно першого елемента*/
     сорт(Lesser, LessSorted), %відсортуємо перший
підсписок...
     сорт(Greater, GreatSorted), %і другий
     конк(LessSorted, [X|GreatSorted], Sorted). %зберемо з
двох відсортованих підсписків результуючий список
```

Тепер ми вже знаємо достатньо, щоб розв'язати нескладну логічну задачу, у якій зручно використати списки – задачу про розфарбовування карти. Нехай ми маємо деяку карту. Треба розфарбувати її так, щоб жодні дві сусідні (які мають спільний кордон) країни не мали однакового кольору. Існує доведена теорема про те, що будь-яку двомірну карту можна розмалювати чотирма кольорами.

Простіше за все представити країни як змінні, яким будуть надаватися значення – їх кольори. Вся карта визначається списком сусідніх країн для кожної з них. Вона буде являти собою список з елементами вигляду `Country-Neighbors`, де `Country` – змінна, яка визначає країну, а Neighbors – список змінних країн-сусідів для `Country` (знак «–» в цьому випадку не означає операцію віднімання, а слугує лише для зв'язку двох термів). Підбирати кольори для країн буде рекурсивний предикат `розмалювати/2`. Він буде по черзі вибирати колір для кожної країни з карти-списку, перевіряючи, щоб жодна з країн-сусідів не мала такого ж кольору. Ми видалимо зі списку всіх кольорів колір поточної країни, тоді сусіди зможуть мати лише колір з отриманого списку. Ось повний текст програми:

```
/* Розфарбовування пласкої карти */
%предикат розмалювати/2 отримує карту та палітру – список
атомів-кольорів, які застосовуватимемо для розмалювання
розмалювати([Country-Neighbours|RestOfMap],Palette):-
     видалити(Country, Palette, Palette1), %виберемо колір
для першої країни. Змінна Country набуває значення зі списку
кольорів Palette, Palette1 містить всі останні кольори
     підмножина(Neighbours, Palette1), %кольори сусідів
повинні міститися в Palette1
     розмалювати(RestOfMap, Palette). %продовжимо з іншими
країнами
розмалювати([], _).

видалити(X, [X|L], L).
видалити(X, [Y|L], [Y|R]):-видалити(X, L, R).

підмножина([], _).
підмножина([X|Xs], Ys):-належить(X, Ys), підмножина(Xs, Ys ).

належить(X, [X|_]).
належить(X, [_|T]):-належить(X,T).
```

Неважко здогадатися, що предикат `підмножина/2` задає відношення підмножини без врахування порядку елементів.

Практична частина

Перевірити роботу предикатів для роботи зі списками.

1. Створіть нову програму і запишіть в ній всі предикати, розглянуті в теоретичних відомостях. Завантажте програму у пам'ять та збережіть у файлі lists.pro – він нам знадобиться в наступних лабораторних роботах.

2. Пряме застосування відношення `належить/2` – перевірка приналежності заданого елемента заданому списку. Спробуйте, наприклад, такі запити:

```
належить(3, [1, 2, 3, 5]).
належить(d, [a, b, 3, 5]).
належить([a, b], [a, b, 3, 5]).
належить([a, b], [[a, b], 3, 5]).
```

З іншого боку, ми можемо замість конкретного елемента вказати змінну – тоді отримаємо всі елементи списку:

```
належить(X, [a, b, c]).
```

Більш широкі можливості має предикат `конк/3`. По-перше, з його допомогою можемо злити два списки в один. Спробуйте такі запити і проаналізуйте відповіді:

```
конк([1, 2], [3, 4, 5], List).
конк([1, X, Y, 4], [4, X, 5], List).
```

По-друге, можна перевірити, чи є третій список об'єднанням двох перших:

```
конк([1, 2], [3, 4, 5], [1, 2, 3, 4, 5]).
```

Третє застосування обернене до першого – розбиття списку на підсписки:

```
конк([1, 2], Y, [1, 2, 3, 4, 5]).
конк(X, [3, 4, 5], [1, 2, 3, 4, 5]).
конк(X, [3, 5], [1, 2, 3, 4, 5]).
конк(X, Y, [1, 2, 3, 4, 5]).
```

По-четверте, можна знаходити елементи, які знаходяться лівіше і правіше заданого елемента. Наприклад, якщо нас цікавить, які елементи знаходяться до і після числа 2, можна поставити таке питання:

```
конк(L, [2|R], [1, 2, 3, 2, 5]).
```

Щоб знайти останній елемент списку, треба спитати:

```
конк(_, [Last], [1, 2, 3, 4, 5]).
```


Можна перевірити, чи є два елемента сусідніми у списку:

```
конк(_, [3, 4|_], [1, 2, 3, 4, 5]).
```

Є підозра, що різноманіття використань предиката конк цими прикладами не вичерпується. У реальних програмах відношення, подібні останнім трьом зручно запрограмувати у вигляді окремих універсальних предикатів.

Не дивно, що за допомогою предиката видалити можна як видаляти елементи зі списку, так і вставляти їх у список:

```
видалити(3, [1, 2, 3, 4, 5], List). %видалення
видалити(a, List, [1, 2, 3, 4, 5]). %включення
```

Спробуйте також такий запит:

```
видалити(X, [1, 2, 3, 4, 5], L).
```

Пролог буде вибирати по черзі всі елементи списку, а на місці L отримаємо результат видалення зі списку поточного значення X.

Роботу предиката підсписок можна розглянути на прикладі таких запитів:

```
підсписок([3, 4], [1, 2, 3, 4, 5]).
підсписок([3, 5], [1, 2, 3, 4, 5]).
підсписок(X, [1, 2, 3, 4, 5]). %знаходження всіх підсписків даного списку
підсписок([3, 4], X).
```

Останній запит змусить Пролог-систему знайти всі списки, підсписком яких може бути даний список. Зрозуміло, що розв'язків цієї задачі буде безліч, і врешті-решт програмі не вистачить пам'яті – виконання запиту закінчиться помилкою виділення пам'яті.

Перевірити роботу предикатів для знаходження оберненого списку, всіх перестановок заданого списку, обчислення довжини списків та сортування ви можете самостійно.

Перевірити роботу програми для розмальовування карти.

1. Створіть нову програму, надрукуйте текст програми для розмальовування карти (див. теоретичні відомості), збережіть її і завантажте у пам'ять.

2. Візьмемо частину карти Європи, схематично зображену на малюнку:

Рис. 12. Схематична карта

Ця мапа зображується таким списком:

```
[Ukraine-[Poland, Slovakia, Hungary, Rumania], Poland-[Ukraine, Slovakia], Slovakia-[Ukraine, Poland, Hungary], Hungary-[Ukraine, Slovakia, Romania], Romania-[Ukraine, Hungary]]
```

Насправді немає необхідності вказувати кожне сусідство двічі, тому опис карти можна скоротити:

```
[Ukraine-[Poland, Slovakia, Hungary, Rumania], Poland-[Slovakia], Slovakia-[Hungary], Hungary-[Romania]]
```

Запитання для програми буде, наприклад, таким:

```
розмалювати([Ukraine-[Poland, Slovakia, Hungary, Rumania], Poland-[Slovakia], Slovakia-[Hungary], Hungary-[Romania]], [червоний, синій, зелений, жовтий]).
```

В результаті ви повинні отримати 432 можливих розв'язки. Ви можете поекспериментувати з різними картами з невеликою кількістю країн. Спробуйте також змінювати кількість кольорів. На жаль, можливості інтерпретатора PIE обмежені, і запит з трохи складнішою мапою закінчиться нестачею пам'яті. Більш великі карти зможе обробляти програма, скомпільована на Visual Prolog.

Завдання до лабораторної роботи 3

Варіант 1

1. Створіть предикат, який замінює у списку перше входження заданого значення іншим.
2. Створіть предикат, який додає 1 до кожного додатного елемента та віднімає 1 від кожного від'ємного елемента.

Варіант 2

1. Створіть предикат, який замінює у списку всі входження заданого значення іншим.
2. Створіть предикат, який перетворює список цілих чисел у список їх абсолютних значень.

Варіант 3

1. Створіть предикат, який за заданим натуральним числом N створює список натуральних чисел від 1 до N у порядку зростання.
2. Створіть предикат, який знаходить елемент списку за його порядковим номером.

Варіант 4

1. Створіть предикат, який за заданим натуральним числом N створює список натуральних чисел від 1 до N у порядку зменшення.
2. Створіть предикат, який здійснює циклічний здвиг заданого списку вліво на один елемент ([1, 2, 3, 4] → [2, 3, 4, 1]).

Варіант 5

1. Створіть предикат, який обчислює суму елементів списку.
2. Дано список, елементами якого є цифри від 0 до 9. Створіть предикат, який переводить цей список у список назв відповідних цифр.

Варіант 6

1. Створіть предикат, який обчислює добуток елементів списку.
2. Дано список, елементами якого є цифри від 0 до 9. Створіть предикат, який переводить цей список у список відповідних римських чисел.

Варіант 7

1. Створіть предикат, який знаходить максимальний елемент списку.
2. Створіть предикат, який знаходить передостанній елемент списку.

Варіант 8

1. Створіть предикат, який знаходить мінімальний елемент списку.
2. Створіть предикат, який видаляє передостанній елемент списку.

Варіант 9

1. Створіть предикат, який обчислює кількість додатних елементів списку.
2. Створіть предикат, який перевіряє, чи є даний список впорядкованим за зростанням.

Варіант 10

1. Створіть предикат, який обчислює кількість від'ємних елементів списку.
2. Створіть предикат, який перевіряє, чи є даний список впорядкованим за убуванням.

Контрольні запитання

1. Що таке список у Пролозі? Як записуються списки? Наведіть приклади.
2. Опишіть роботу предикатів для визначення приналежності атома списку та для знаходження довжини списку.
3. Опишіть роботу предиката конкатенації списків.
4. Які застосування предиката конкатенації вам відомі?
5. Опишіть роботу предиката для видалення елемента зі списку.
6. Опишіть роботу предиката для знаходження підсписків.
7. Опишіть роботу предиката для знаходження оберненого списку.
8. Поясніть роботу програми для сортування списків.
9. Поясніть роботу програми розфарбовування карти.
10. *Описаний предикат сорт дозволяє сортувати списки чисел. Чи можете ви запропонувати ідею предиката, який реалізував би аналогічний спосіб сортування для списків об'єктів іншої природи (наприклад, дат, заданих структурами як у першій лабораторній роботі)?

2.5.4. Лабораторна робота 4. Логічні задачі. Задачі на задоволення обмежень

Мета: сформувати уявлення про логічні задачі та задачі задоволення обмежень та методи їх розв'язання на Пролозі, сформувати навички застосування методу «сформувати і перевірити»

Теоретичні відомості

Однією з особливостей Прологу є простота написання програм для розв'язування логічних задач у порівнянні з імперативними мовами.

Під логічними задачами ми будемо розуміти задачі, в яких за допомогою логічних тверджень задаються відношення між об'єктами і треба знайти таку комбінацію характеристик об'єктів, при якій задовольняються умови задачі. Оскільки Пролог оснований на логіці предикатів, то він пристосований до розв'язування логічних задач, однак іноді переклад задачі на Пролог є нелегким завданням.

Універсальним способом розв'язування логічних задач на Пролозі є так званий метод «сформувати та перевірити». Його суть полягає у формуванні множини можливих розв'язків та наступній перевірці їх на задоволення умов задачі.

Можна запропонувати такий алгоритм розв'язування логічної задачі на Пролозі за допомогою цього метода:

1. Проаналізувати умову задачі, визначити множину об'єктів, які фігурують в умові, їх властивості, взаємовідношення. Визначити, які саме властивості потрібно отримати у результаті.
2. Обрати спосіб зображення об'єктів та їх властивостей у програмі. Об'єкти можна подати у вигляді змінних або списку змінних, яким будуть надаватися значення – їх характеристики. Множина можливих значень для кожної властивості об'єкта зазвичай задається списком атомів або структур.
3. Записати предикат сформувати, який буде формувати можливі комбінації об'єктів та наборів їх характеристик, спираючись на множини можливих значень властивостей. Тут часто зручно застосувати предикати `належить/2, видалити/3, перестановка/2, всі_різні/1`.
4. Записати предикат перевірити, який буде застосовувати умови задачі до сформованого набору об'єктів і властивостей і перевіряти, чи виконуються умови. Для цього зазвичай достатньо переписати умови задачі з природної мови на Пролог.
5. Поставити запит до програми та перевірити її роботу. Запит матиме такий вигляд:

```
сформувати(...), перевірити(...). ,
```

Аргументами предикатів будуть об'єкти-змінні (або їх список). Завдяки роботі механізму перебору Пролог-система видасть всі набори шуканих характеристик об'єктів, які задовольняють умовам задачі.

Розглянемо декілька прикладів.

1. Перегони.

В автомобільних перегонах три перших місця посіли Петро, Микола та Олексій. Яке місце посів кожен з них, якщо Петро зайняв не друге та не третє місце, а Микола – не третє?

Об'єктами цієї задачі є гонщики Петро, Микола і Олексій – їх подамо у вигляді змінних `Petro, Mykola, Oleksiy`. Властивостями об'єктів є місця, ними займані – множину їх значень задамо списком `[1, 2, 3]`. Предикат сформувати буде надавати змінним-об'єктам можливі для них значення з цього списку. Предикат перевірити буде перевіряти значення змінних на відповідність умовам задачі.

```
місця([1, 2, 3]). %можливі місця
сформувати(Petro, Mykola, Oleksiy):-
     місця(Places), %отримаємо список місць
     видалити(Petro, Places, PlacesRest1), /*виберемо місце
для Петра. Обране місце видаляється зі списку доступних
місць, оскільки два гонщики не можуть посідати одне місце*/
     видалити(Mykola, PlacesRest1, PlacesRest2), %для Миколи
     видалити(Oleksiy, PlacesRest2, _). %для Олексія
перевірити(Petro, Mykola, Oleksiy):-
     not Petro=2, not Petro=3, %Петро посів не друге і не
третє місце
     not Mykola=3. %а Микола – не третє
```

Предикат видалити був описаний у третій лабораторній роботі.

2. Лицарі і брехуни.

На острові мешкають лицарі і брехуни. Лицарі завжди кажуть правду, брехуни завжди брешуть. Перехожий зустрів трьох мешканців острова A, B, і C та запитав у A: «Ви лицар чи брехун?». Той відповів, та дуже нерозбірливо, і перехожий не зміг нічого зрозуміти. Тоді незнайомець запитав у B: «Що сказав A?». «A сказав, що він брехун», – відповів B. «Не вірте B, він бреше!» – втрутився у розмову C. Хто з B і C лицар, а хто – брехун?

Об'єктами є три людини A, B, і C, а їх властивостями – належність до лицарів або брехунів. Розмову з незнайомцем можна запрограмувати за допомогою предиката `сказав(X, Sent)`, де X – людина, а Sent – твердження, яке вона сказала (записане у вигляді предиката). Крім того, потрібно пояснити Пролог-системі, що лицарі завжди кажуть правду, а брехуни – брешуть.

```
сказав(лицар, Sent):-Sent. %якщо лицар сказав Sent, то Sent
вірне
сказав(брехун, Sent):-not Sent. %якщо брехун сказав Sent, то
Sent не вірне
види([лицар, брехун]). %всі люди поділяються на лицарів і
брехунів
сформувати(A, B, C):-
     види(L), %отримаємо список видів
     належить(A, L), належить(B, L), належить(C, L). %та
виберемо вид для кожного співрозмовника
перевірити(A, B, C):-
     сказав(B, сказав(A, A=брехун)), %B сказав, що A сказав,
що A – брехун
     сказав(C, B=брехун). %C сказав, що B – брехун
```

3. Принцеса чи тигр?

Є дві кімнати, в кожній з яких можуть знаходитися або принцеса, або тигр. Лицар повинен вгадати, хто перебуває в кожній кімнаті. Якщо він вгадає, то жениться на принцесі, інакше його роздере тигр. На дверях кожної кімнати є таблички, на яких написано:

I. В цій кімнаті знаходиться принцеса, а в іншій – тигр.

II. В одній з цих кімнат знаходиться принцеса, крім того, в одній з цих кімнат сидить тигр.

Відомо, що на одній табличці написано правду, на іншій напис невірний.

Допоможіть лицарю знайти кімнату з принцесою, використовуючи Пролог.

Цього разу кімнати представимо у вигляді змінних First та Second, яким будуть надаватися значення принцеса або тигр.

```
напис1(принцеса, тигр). %напис на першій кімнаті
напис2(принцеса, тигр). %напис на другій кімнаті – два
варіанти
напис2(тигр, принцеса).
види([принцеса, тигр]).
```

```
сформувати(First, Second):-види(L), належить(First, L),
належить(Second, L).
перевірити(First, Second):-
      %або перший напис вірний, другий - невірний...
      напис1(First, Second), not напис2(First, Second).
перевірити(First, Second):-
      %або навпаки
      not напис1(First, Second), напис2(First, Second).
```

Одним з напрямків розвитку штучного інтелекту є розробка ефективних методів розв'язування задач задоволення обмежень. Такі задачі визначаються множиною змінних, множиною відповідних значень змінних та множиною обмежень на змінні. Розв'язком задачі є такі значення змінних, які задовілбняють всі обмеження. Багато задач з різних галузей науки і техніки можна віднести до цього класу задач, проте залишається актуальною проблема розробки універсальних алгоритмів розв'язування задач задоволення обмежень.

Для розв'язування цих задач можна використовувати розглянутий метод «сформувати та перевірити», проте його ефективність недостатньо висока для більш-менш складних задач, в яких простір перебору значень змінних може бути дуже великим. Розглянемо одну з класичних задач на задоволення обмежень.

На шахматній дошці NxN розмістити N ферзів так, щоб вони не били один одного.

Якщо позицію на дошці представити у вигляді пар координат ферзів (x, y), то задачу можна формалізувати наступним чином:

Знайти такий набір значень для $x_1, y_1, x_2, y_2, ..., x_N, y_N$ з множини [1..N], що для всіх різних i та j виконуються умови:

1) x_i =/= x_j (ферзі не стоять на одній вертикалі);
2) y_i =/= y_j (ферзі не стоять на одній горизонталі);
3) $|x_i - x_j|$ =/= $|y_i - y_j|$ (ферзі не стоять на одній діагоналі).

Можна, звичайно, перебирати всі можливі пари x та y для кожного ферзя та перевіряти отриману позицію на задоволення цих обмежень, проте в такому разі простір перебору навіть на невеликих дошках стане занадто великим. Спробуємо спростити умову. Очевидно, що ферзі завжди будуть займати всі вертикалі, тому координати x можна

представити у вигляді впорядкованого списку [1, 2, ... N]. Потрібно знайти відповідний йому список координат [$y_1, y_2, \ldots, y_N$], тобто перебирати можна лише координати y. Крім того, ферзі займатимуть всі горизонталі, і жодні два ферзя не знаходитимуться на одній горизонталі, тобто список y-координат складатиметься з всіх різних чисел від 1 до N. Такий список можна отримати перестановкою елементів списку [1, 2, ... N]. Список x-координат та відповідний йому список y-координат однозначно визначають позицію на дошці.

Запишемо програму для знаходження можливих розташувань ферзів для дошки 4x4 (для менших дошок задача не має розв'язку).

```
розвязок(P):-
      перестановка([1,2,3,4], P), %сформуємо список
у-координат
      безпечна([1,2,3,4], P). %і перевіримо отриману позицію

безпечна([], []).
безпечна([X|Xs], [Y|Ys]):-
      не_атакує(X, Y, Xs, Ys), %перший ферзь не атакує інших
      безпечна(Xs, Ys). %решта ферзів не б'ють один одного

не_атакує(_, _, [], []).
не_атакує(X, Y, [X1|Xs], [Y1|Ys]):-
      Y <> Y1, %різні горизонталі
      Y1-Y <> X1-X, Y1-Y <> X-X1, %різні діагоналі
      не_атакує(X, Y, Xs, Ys). %перевіримо решту ферзів
```

Предикат `безпечна/2` отримує список x-координат і список y-координат та перевіряє задану позицію на відповідність умовам задачі. Для того, щоб перевірити позицію, потрібно взяти одного ферзя, перевірити, чи не атакує він інших, а потім аналогічно перевірити решту ферзів. Предикат `не_атакує/4` отримує координати ферзя і списки координат решти ферзів та перевіряє, чи не атакує заданий ферзь хоча б одного з решти.

Практична частина

Перевірити роботу програм, описаних в теоретичних відомостях.

1. Запустіть середовище PIE, наберіть текст програми «Перегони» та збережіть її на диску, завантажте програму у пам'ять (Engine►Reconsult). Відкрийте також файл lists.pro, створений вами

у попередній лабораторній роботі (у цьому файлі описаний предикат видалити, який використовується в нашій програмі). Запит можна поставити у такому вигляді:

```
сформувати(Petro, Mykola, Oleksiy), перевірити(Petro, Mykola, Oleksiy).
```

В результаті ви повинні отримати місця, які посіли учасники перегонів як значення відповідних змінних. Перевірте, чи справді отриманий розв'язок відповідає умовам задачі.

2. Приведіть систему у вихідний стан (Engine►Reset), створіть нову програму, надрукуйте текст програми «Лицарі і брехуни», збережіть її та завантажте у пам'ять. Завантажте знов файл lists.pro. Запит матиме такий вигляд:

```
сформувати(A, B, C), перевірити(A, B, C).
```

Задача має два розв'язки, які відрізняються лише статусом людини A. Отже, на питання, поставлене в задачі, можна дати конкретну відповідь. Перевірте, чи справді отриманий розв'язок відповідає умовам задачі.

3. Приведіть систему у вихідний стан, створіть нову програму, надрукуйте текст програми «Принцеса чи тигр?», збережіть її та завантажте у пам'ять. Завантажте знов файл lists.pro. Самостійно сформулюйте запит до програми та перевірте його. Задача має єдиний розв'язок. Перевірте, чи справді отриманий розв'язок відповідає умовам задачі.

4. Приведіть систему у вихідний стан, створіть нову програму, надрукуйте текст програми для розв'язування задачі про ферзів, збережіть її та завантажте у пам'ять. Завантажте знов файл lists.pro. Поставте такий запит:

```
розвязок(L).
```

В результаті ви отримаєте розв'язки – списки з у-координатами ферзів. Перший елемент у списку буде номером горизонталі для ферзя на першій вертикалі, другий елемент – для ферзя на другій вертикалі і т. д. Ви можете накреслити отримані розв'язки на папері, щоб впевнитися, що ферзі в отриманих позиціях справді не б'ють один одного.

Знайдіть можливі розташування ферзів для дошок 5x5, 6x6. Для цього достатньо змінити відповідним чином списки x- та y-координат в предикаті `розвязок`, не забуваючи перезавантажувати програму у пам'ять після змін. Для дошки 4x4 задача має 2 розв'язки, 5x5 – 10 розв'язків, 6x6 – 4 розв'язки.

Завдання до лабораторної роботи 4

Створіть програму для розв'язування задачі:

Варіант 1

Принцеса чи тигр?

На цей раз на дверях кімнат написано:

I. Хоча б в одній з цих кімнат знаходиться принцеса.

II. Тигр сидить в іншій кімнаті.

Причому відомо, що або обидва твердження вірні, або обидва невірні. Яку б кімнату ви обрали?

Варіант 2

Принцеса чи тигр?

Цього разу на табличках такі написи:

I. Або в цій кімнаті сидить тигр, або принцеса знаходиться в іншій кімнаті.

II. Принцеса в іншій кімнаті.

Твердження або одночасно вірні, або одночасно невірні. Яку б кімнату ви обрали?

Варіант 3

Принцеса чи тигр?

На цей раз на дверях кімнат написано:

I. В кожній кімнаті сидить по принцесі.

II. В кожній кімнаті сидить по принцесі.

Якщо в першій кімнаті знаходиться принцеса, то твердження на табличці першої кімнати вірне, якщо тигр – невірне. Якщо в другій кімнаті знаходиться тигр, то твердження на табличці другої кімнати вірне, якщо принцеса – невірне. Яку б кімнату ви обрали?

Варіант 4

Принцеса чи тигр?

Цього разу на табличках такі написи:

I. Хоча б в одній кімнаті знаходиться принцеса.

II. Принцеса в іншій кімнаті.

Якщо в першій кімнаті знаходиться принцеса, то твердження на табличці першої кімнати вірне, якщо тигр – невірне. Якщо в другій кімнаті знаходиться тигр, то твердження на табличці другої кімнати вірне, якщо принцеса – невірне. Яку б кімнату ви обрали?

Варіант 5

Лицарі і брехуни.

Перед нами три мешканці острова A, B, і C. A сказав, що B – брехун. B сказав, що A і C однотипні. Хто такий C – лицар чи брехун?

Варіант 6

Лицарі і брехуни.

Перед нами два мешканці острова A і B. B сказав, що A сказав, що B сказав, що A сказав, що B – брехун. Хто з них хто?

Варіант 7

Лицарі і брехуни.

Перед нами два мешканці острова A і B. A сказав, що вони обидва брехуни. Хто з них хто?

Варіант 8

Вітя, Михайло та Юра сиділи на лавці. У якому порядку вони сиділи, якщо відомо, що Юра сидів зліва від Михайла та правіше за Вітю.

Варіант 9

Вітя, Михайло та Юра сиділи на лавці. У якому порядку вони сиділи, якщо відомо, що Михайло сидів зліва від Юри, а Вітя зліва від Михайла.

Варіант 10

На лавці сидять Марі, її мама, бабуся і лялька. Бабуся сидить поруч із внучкою, але не поруч із лялькою. Лялька не сидить поруч із мамою. Хто сидить поруч із мамою Марі?

Контрольні запитання

1. В чому полягає сутність методу «сформувати та перевірити»?
2. Опишіть основні етапи розв'язання логічної задачі методом «сформувати і перевірити».
3. Що розуміється під задачею задоволення обмежень?
4. Поясніть роботу програми «Перегони».
5. Поясніть роботу програми «Лицарі і брехуни».
6. Поясніть роботу програми «Принцеса або тигр?».
7. Поясніть роботу програми для розв'язування задачі про ферзів.
8. *Метод «сформувати і перевірити» часто не є найоптимальнішим. Наприклад, в задачі про ферзів поточна позиція завжди генерується до останнього ферзя і лише потім перевіряється, хоча іноді формувати всю позицію недоцільно, тому що вже поставлені ферзі можуть атакувати один одного. Запропонуйте ідеї підвищення ефективності цієї програми.

2.5.5. Лабораторна робота 5. Пошук у графах та деревах

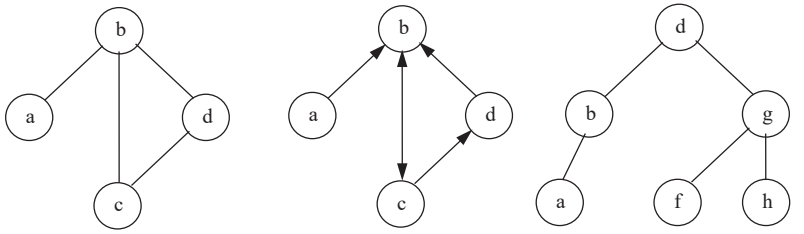
Мета: сформувати уявлення про графи і дерева, основні методи пошуку в них, навчитися застосовувати ці методи до розв'язування задач.

Теоретичні відомості

Граф – це структура даних, що складається з множини *вершин* та множини *ребер*. Кожне ребро сполучає дві вершини. Якщо для кожного ребра заданий напрямок, то граф називається *орієнтованим*, інакше – *неорієнтованим*. Ребра, що мають напрямок, називають *дугами*. Дві вершини орієнтованого графа, з'єднані дугою, називаються *батьком* і *сином* (або головною і підлеглою вершинами).

Шлях – це послідовність вершин, сполучених ребрами. Для орієнтованого графа напрямок шляху повинен збігатися з напрямком кожної дуги, через яку він проходить. Шлях, у якого збігаються початок і кінець, називається *циклом*. Якщо з однієї вершини орієнтованого графа існує шлях до іншої, то перша називається *предком*, друга – *нащадком*.

Дерево – це граф, в якому існує одна коренева вершина (вона не має батька), а всі інші вершини мають лише одного батька і є нащадками кореневої вершини. Дерево не містить циклів. *Листом* дерева називається його вершина, що не має синів. *Кроною* дерева називається сукупність всіх листів. *Висотою* дерева називається найбільша довжина шляху від кореня до аркуша.



Неорієнтований граф Орієнтований граф Дерево

Рис. 13. Графи

Ребро графа визначається парою вершин, які воно сполучає. Дуга орієнтованого графа визначається впорядкованою парою вершин. Весь граф визначається списком ребер.

Орієнтовані графи будемо задавати за допомогою відношення дуга/2, наприклад, множиною фактів вигляду

```
дуга(a, b). %в графі існує дуга від вершини a до вершини b
```

Для неорієнтованого графа ребра зобразимо відношенням ребро/2:

```
ребро(a, b). %ребро сполучає вершини a і b
```

Весь неорієнтований граф можна подати як орієнтований, поставивши кожному ребру у відповідність дві дуги:

```
дуга(A, B):-ребро(A, B); ребро(B, A).
```

Це правило означає, що існує дуга від A до B, якщо існує ребро від A до B або від B до A. Таким чином, ребру ребро(a, b) будуть відповідати дуги дуга(a, b) і дуга(b, a).

Однією з найважливіших задач для графів є пошук шляху від однієї вершини до іншої. Ми розглянемо два методи пошуку шляху: пошук в глибину і пошук в ширину. Вибір методу пошуку залежить від структури графу та розташування початкової і кінцевої вершин.

Пошук в глибину ґрунтується на такому принципі:

1. Візьмемо початкову вершину за поточну.
2. Якщо поточна вершина графу є цільовою, то шлях знайдений.
3. Інакше виберемо одну з вершин, з'єднаних з поточною і яку ми ще не розглядали, візьмемо її за поточну і спробуємо рекурсивно знайти шлях з неї до цільової вершини.
4. Якщо з поточної вершини ми не можемо перейти в жодну іншу ще не відвідану вершину, то повернімося до попередньої і випробуємо інші її вершини-наступниці.

Процедура пошуку в глибину обходить вершини в такому порядку, як показано на рис. 15.

Пунктиром показано процес пошуку, жирними стрілками – знайдений шлях, a – початкова, e – цільова вершина.

Пошук в глибину реалізуємо у вигляді предикату шлях1(A, B, Path),

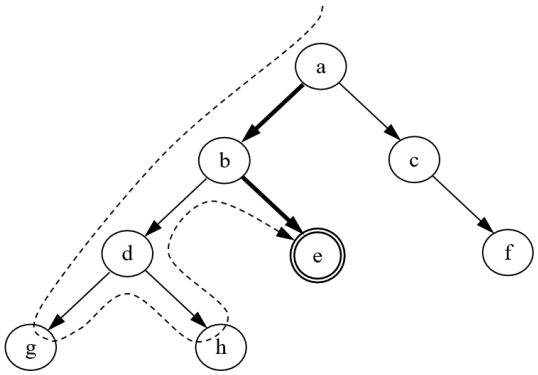


Рис. 14. Процедура пошуку в глибину

де A – початкова вершина, B – кінцева, Path – шлях від A до B. Власне пошуком буде займатися рекурсивний предикат в_глибину(A, B, Visited, Path), де Visited – список вже відвіданих вершин, а на місці Path отримаємо шлях від A до B.

```
шлях1(A, B, Path):-в_глибину(A, B, [A], Path).

в_глибину(A, A, _, [A]). /*прийшли в кінцеву вершину –
     зараз шуканий шлях складається лише з цієї вершини,
     на зворотному шляху рекурсії буде зібраний весь шлях в
потрібному порядку*/
в_глибину(A, B, Visited, [A|Path]):- %до шляху додається
поточна вершина
     дуга(A, C), %виберемо вершину-наступника
     not належить(C, [A|Visited]), %перевіримо, що цю вершину
ми ще не відвідували
     в_глибину(C, B, [C|Visited], Path). %запустимо пошук з C,
додамо її до списку перевірених вершин
```

Стратегія пошуку в ширину передбачає перехід у першу чергу до найближчих до стартової вершин, тому з всіх можливих шляхів першим знаходить найкоротший, рис. 16.

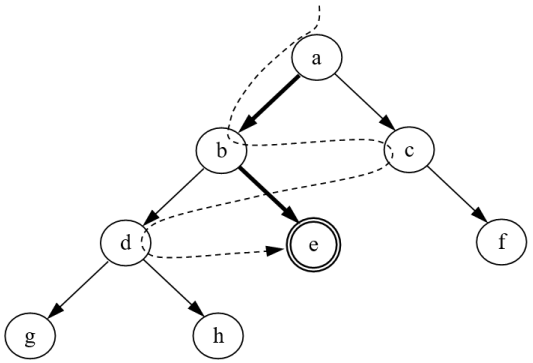


Рис. 15. Процедура пошуку в ширину

Для реалізації цього методу пошуку нам доведеться зберігати список шляхів-кандидатів. Кожен шлях являтиме собою список вершин,

перерахованих у зворотному порядку. Пошук буде починатися з одного шляху-кандидата, який складатиметься лише з початкової вершини:

```
[[A]]
```

На кожному кроці ми будемо продовжувати кожен із збережених шляхів. Таким чином, процес пошуку буде розвиватися рівномірно по всім напрямкам. Отже, для того, щоб виконати пошук в ширину при заданій множині шляхів-кандидатів, потрібно:

1. Якщо голова першого шляху – цільова вершина, то взяти цей шлях в якості розв'язку.

2. Інакше видалити перший шлях зі списку кандидатів і породити множину всіх можливих продовжень цього шляху на один шаг, додати цю множину в кінець списку кандидатів та виконати пошук в ширину з отриманим списком.

Ось програма, яка реалізує цей принцип:

```
шлях2(A, B, Path):-в_ширину([[A]], B, Path1),
обернений(Path1, Path).

в_ширину([[B|Path] | _], B, [B|Path]). /*поточний шлях-
кандидат закінчується в кінцевій вершині B - покладемо цей
шлях за розв'язок*/
в_ширину([[Node|Path] | Paths], B, Solution):- %[Node|Path] -
поточний шлях-кандидат
     findall(Cont, продовження(Cont, [Node|Path]), NewPaths),
          /*Знайдемо список NewPathes можливих продовжень
поточного шляху-кандидата.
          Для цього знайдемо всіх можливих наступників Node1
вершини Node, і додамо Node1 в початок поточного кандидата.*/
     конк(Paths, NewPaths, Paths1), /*додамо знайдений список
продовжень в кінець хвоста списку кандидатів*/
     в_ширину(Paths1, B, Solution). /*і запустимо пошук з
новим списком кандидатів*/

/*перехід від Node до Node1 можливий, якщо між ними існує
дуга і Node1 немає в списку вже відвіданих вершин*/
продовження([Node1, Node|Path], [Node|Path]):-дуга(Node,
Node1), not належить(Node1, [Node|Path]).
```


Предикат `шлях2(A, B, Path)` шукає шлях Path від вершини A до B за допомогою рекурсивного предиката `в_ширину([[Node|Path] | Paths], B, Solution)`, де [[Node|Path] | Paths] – список шляхів-кандидатів, [Node|Path] – поточний шлях-кандидат (за один крок знаходяться всі можливі продовження для цього шляху), B – кінцева вершина, Solution – шуканий шлях. Предикат `продовження(Cont, [Node|Path])` знаходить одне продовження для шляху [Node|Path].

Предикат `findall(X, G, L)` є стандартним для багатьох реалізацій Прологу і створює список L всіх об'єктів X, які задовольняють ціль G. В нашій програмі цей предикат збирає список NewPaths всіх продовжень поточного шляху-кандидату, кожне з яких генерується предикатом продовження/2. На жаль, інтерпретатор PIE не підтримує цей предикат, тому нам доведеться визначити його власноруч:

```
findall(X,Goal,Xlist):-
     call(Goal), assertz(queue(X)), fail;
     assertz(queue(bottom)), collect(Xlist).
collect(L):-
     retract(queue(X)), !, ((X == bottom, !, L = []);
     (L=[X|Rest], collect(Rest))).
```

Багато задач можна сформулювати наступним чином:

1. Дано деякий вихідний стан.

2. Визначено множину цільових станів. Ця множина може складатися з одного елемента, а може задаватися деяким правилом, яке визначає, що мета досягнута.

3. Визначено множину операцій, які можна використовувати для зміни поточного стану.

4. Потрібно знайти послідовність станів, які приводять до цільового стану.

Ми можемо подати простір станів задачі у вигляді орієнтованого графу, в якому станам відповідають вершини, а операціям – дуги. Таким чином, задача зводиться до пошуку шляху в графі.

Розглянемо наступну задачі.

Задача 1. Фермеру потрібно перевезти з лівого берега на правий вовка, козу і капусту. При цьому човен вміщує власне фермера і ще

лише один об'єкт, і не можна залишити на одному березі без нагляду вовка і козу або козу і капусту.

Нехай структура `ст(F, W, G, C)` визначає місцезнаходження фермера, вовка, кози і капусти відповідно. Кожна компонента структури набуває значення л, якщо відповідний об'єкт знаходиться на лівому березі, і п, якщо на правому. Всі можливі структури такого вигляду задають вершини графу. Початковою вершиною буде `ст(л,л,л,л)`, кінцевим – `ст(п,п,п,п)`. Можливими операціями є переміщення фермера або фермера з яким-небудь вантажем з одного берега на протилежний.

```
початок(ст(л,л,л,л)). %всі на лівому березі
кінець(ст(п,п,п,п)). %всі на правому березі
дуга(S,S1):-
      переміщення(S,S1), %перехід від стану S до S1
      безпечний(S1). %стан S1 дозволений
переміщення(ст(X,W,G,C), ст(Y,W,G,C)):-протилежний(X,Y).
%переїзд фермера
переміщення(ст(X,X,G,C), ст(Y,Y,G,C)):-протилежний(X,Y).
%переїзд фермера і вовка
переміщення(ст(X,W,X,C), ст(Y,W,Y,C)):-протилежний(X,Y).
%переїзд фермера і кози
переміщення(ст(X,W,G,X), ст(Y,W,G,Y)):-протилежний(X,Y).
%переїзд фермера і капусти
протилежний(л,п).
протилежний(п,л).
/*козу не можна залишати сам на сам з вовком або капустою,
отже, коза завжди повинна бути там, де і фермер*/
безпечний(ст(X,_,X,_)).
/*інакше разом з фермером повинні знаходитися вовк і капуста,
тоді коза може бути на якому завгодно березі*/
безпечний(ст(X,X,_,X)).

розвязок(Answer):-початок(S), кінець(E), шлях1(S, E, Answer).
```

Ця програма визначає граф станів нашої задачі за допомогою відношення `дуга/2`, а предикат `розвязок/1` використовує описаний раніше предикат `шлях1/3` для пошуку шляху.

Задача 2. Є дві посудини на 5 літрів і 2 літри. Можна повністю наповнювати посудини водою, виливати з них всю воду, а також переливати як завгодно воду з однієї посудини в іншу. Потрібно відміряти у другій посудині рівно 1 літр.

Можливі стани системи з двох посудин знову подамо у вигляді структури `ст(X, Y)`, де X і Y – кількість літрів води у першій і другій посудинах відповідно. Всі можливі переливання задаються відношенням `дуга/2`.

```
початок(ст(0,0)). %спочатку обидві посудини порожні
кінець(ст(_,1)). %потрібно отримати в другій 1 л

дуга(ст(_,X), ст(5,X)). %налити воду в першу посудину
дуга(ст(X,_), ст(X,2)). %налити воду в другу посудину
дуга(ст(_,X), ст(0,X)). %вилити воду з першої посудини
дуга(ст(X,_), ст(X,0)). %вилити воду з другої посудини
%перелити воду з першої посудини в другу:
%Rest2 – вільне місце в другій посудині
дуга(ст(X,Y), ст(X1,2)):-Rest2 is 2-Y, X>=Rest2, X1 is X-Rest2.
/*якщо в першій посудині більше води, ніж вільного місця в
другій, то друга наповнюється доверху, а з першої виливається
стільки, скільки вільного місця в другій*/
дуга(ст(X,Y), ст(0,Y1)):-Rest2 is 2-Y, X<Rest2, Y1 is Y+X.
/*якщо в першій посудині менше води, ніж вільного місця в
другій, то з першої вся вода переливається в другу*/
%перелити воду з другої посудини в першу:
%Rest1 – вільне місце в першій посудині
дуга(ст(X,Y), ст(5,Y1)):-Rest1 is 5-X, Y>=Rest1, Y1 is Y-Rest1.
/*якщо в другій посудині більше води, ніж вільного місця в
першій, то перша наповнюється доверху, а з другої виливається
стільки, скільки вільного місця в першій*/
дуга(ст(X,Y), ст(X1,0)):-Rest1 is 5-X, Y<Rest1, X1 is X+Y.
/*якщо в друій посудині менше води, ніж вільного місця в
першій, то з другої вся вода переливається в першу*/

розвязок(Answer):-початок(S), кінець(E), шлях1(S, E, Answer).
```

Практична частина

Створити програму для пошуку шляхів у графі методами пошуку в глибину і ширину.

Створіть нову програму і надрукуйте описані реалізації предикатів для пошуку в глибину і в ширину, включаючи предикат findall/3. Збережіть отриману програму у файлі search.pro.

Перевірити роботу програм, описаних в теоретичних відомостях.

1. Створіть новий файл і напишіть текст програми для розв'язку задачі про вовка, козу і капусту, збережіть її на диску. Завантажте у пам'ять цю програму, щойно створений вами файл search.pro і файл lists.pro з третьої лабораторної роботи.

2. Поставте запит до цієї програми:

```
розвязок(Solution).
```

Ви повинні отримати 4 розв'язки у вигляді списків станів, які визначають послідовність переходу з вихідного стану до кінцевого. Деякі з розв'язків збігаються, і всього задача має 2 різні розв'язки. Розглянемо один з них:

```
[ст(л,л,л,л) | [ст(п,л,п,л) | [ст(л,л,п,л) | [ст(п,л,п,п) | [ст(л,л,л,п) |
[ст(п,п,л,п) | [ст(л,п,л,п) | [ст(п,п,п,п) ]]]]]]]]
```

Список починається з вихідного стану – всі на лівому березі і закінчується станом, коли всі переправилися на правий берег. Аналізуючи перший перехід `ст(л,л,л,л) – ст(п,л,п,л)`, помітимо, що фермер (перший елемент структури) перевіз на правий берег козу (третій елемент структури), залишивши на лівому вовка з капустою (другий і четвертий елементи), що дозволяється умовами задачі. На наступному кроці фермер повертається на лівий берег і т. д. Проаналізуйте, що саме перевозив фермер на кожному кроці і перевірте проміжні стани на відповідність умовам задачі (чи не з'їсть вовк козу, а коза – капусту) для обох розв'язків.

3. Створіть і збережіть програму для розв'язування задачі про переливання води. Приведіть систему у вихідний стан і завантажте в пам'ять цю програму разом з файлами search.pro і lists.pro.

4. Поставте запит до цієї програми:

```
розвязок(Solution).
```

Задача матиме 27 розв'язків. Виберіть серед них найкоротші і відтворіть послідовність переливань, які приводять до розв'язку задачі.

Завдання до лабораторної роботи 5

Створіть програму для розв'язування задачі:

Варіант 1

Три місіонери і три канібали намагаються переправитися з лівого берега ріки на правий. Але човен уміщає не більше двох чоловік, а якщо на одному з берегів канібалів опиниться більше, ніж місіонерів, останнім загрожує доля Кука. Допоможіть їм у нелегкій справі спланувати переправу.

Варіант 2

Три дуже ревнивих чоловіки разом зі своїми дружинами бажають переправитися через ріку. Човен знову ж уміщає не більше двох чоловік, а жоден із чоловіків не може залишити свою дружину без догляду в товаристві інших чоловіків. Допоможіть їм у нелегкій справі спланувати переправу.

Варіант 3

Є три посудини, ємністю 10 л, 5 л і 3 л. Десятилітрова посудина доверху наповнена коштовною рідиною, дві інших порожні. Переливаючи рідину з однієї посудини в іншу, ви можете відміряти певну кількість рідини і випити її. За це ви повинні точно розподілити рідину, що залишилася, нарівно по трьох посудинах. Тривіальний варіант, коли випивається вся рідина, неприпустимий.

Варіант 4

Є три посудини ємністю 8, 5 і 3 літри, перша посудина наповнена водою повністю. Як, використовуючи переливання, одержати у двох посудинах по 4 літри (доливати ще воду не дозволяється)?

Варіант 5

Є три посудини ємністю 10, 7 і 2 літри, перша посудина наповнена водою повністю. Як, використовуючи переливання, одержати у двох посудинах по 5 літрів (доливати ще воду не дозволяється)?

Варіант 6

Маємо шестилітрову каструлю і чотирилітрову банку. Потрібно налити з бочки 1 літр води.

Варіант 7

Мавпа знаходиться біля входу в кімнату, посередині якої підвішений банан. З підлоги мавпа не дотягнеться до банана. Біля вікна кімнати стоїть ящик. Мавпа може переміщуватися по кімнаті, рухати ящик,

залазити і злазити з нього, схопити банан (якщо вона знаходиться на ящику посередині кімнати). Потрібно визначити послідовність дій, які повинна виконати мавпа, щоб дістати банан.

Варіант 8

Є 7-літрове відро меду, а також два порожніх відерця на 4 і 3 літри. Потрібно відміряти 1 л меду в трилітровому відерці, залишивши у великому відрі 6 літрів.

Варіант 9

Дехто має бочку соку ємністю 12 пінт та хоче подарувати половину своєму другові. В нього є також дві посудини на 5 і 8 пінт. Як можна налити 6 пінт в посудину на 8 пінт?

Варіант 10

Невеликий загін з N солдат потрібно переправити через річку. На березі знаходиться човен і два хлопця. Човен вміщає не більше одного солдата або двох хлопців. Якою повинна бути послідовність перевезень?

Контрольні запитання

1. Дайте визначення поняттям граф і дерево.
2. Дайте визначення поняттям шлях і цикл.
3. Дайте визначення поняттям лист, крона і висота дерева.
4. Які графи називаються орієнтованими і неорієнтованими?
5. Як можна подати орієнтовані і неорієнтовані графи в програмі на Пролозі?
6. Опишіть метод пошуку в глибину.
7. Опишіть метод пошуку в ширину.
8. Як подається у термінах графів задача про вовка, козу і капусту?
9. Як подається у термінах графів задача про переливання води?

Література до розділу

1. Гаврілова, Т. А. Бази знань інтелектуальних систем / Т. А. Гаврілова, В. Ф. Хорошевський. – СПб. : Пітер, 2001. – 384 с. (рос.)

2. Методи та моделі аналізу даних: OLAP и Data Mining / А. А. Барсегян та інш. – СПб. : БХВ-Петербург, 2004. – 336 с. (рос.)

3. Рассел, С. Штучний інтелект: сучасний підхід / С. Рассел, П. Норвиг. – 2-е изд. – М. : Вільямс, 2006. – 1408 с. (рос.)

4. Інтелектуальні інформаційні системи: навч.посібник / А. А. Смагін, С. В. Ліпатова, А. С. Мельниченко. – Ульяновськ: УлГУ, 2010. – 136 с. (рос.)

5. Інтелектуальні інформаційні системи. PROLOG – мова розробки інтелектуальних та експерних систем: навч. посібник / С. П. Хабаров. – СПб. СПбГЛТУ, 2013. – 138 с. (рос.)

6. Представлення знань в інформаційних системах. Використання середовища PIE при проектування баз даних та знань: навч. посібник / С. П. Хабаров, Л. Г. Пушкарева; / під ред. А. М. Заяц. – Санкт-Петербург: СПбГЛТУ, 2019. – 66 с. (рос.)

РОЗДІЛ ІІІ. ЕКСПЕРТНІ СИСТЕМИ

3.1. Експертні системи – системи, що базуються на знаннях

Експертна система (ЕС, англ. expert system) – комп'ютерна система, здатна частково замінити фахівця-експерта у вирішенні проблемної ситуації. ЕС почали розроблятися дослідниками штучного інтелекту в 1970-х роках, а в 1980-х отримали комерційне підкріплення. Термін «системи, грунтовані на знаннях» (knowledge based systems) з'явився в 1976 році одночасно з першими системами, що акумулюють досвід і знання експертів.

Експертні системи – це прикладні системи штучного інтелекту, в яких база знань є формалізованими емпіричними знаннями висококваліфікованих фахівців (експертів) в якій-небудь вузької предметної області, вони акумулюють ці знання і тиражують їх для консультації менш кваліфікованих фахівців.

У 1965 році Е. Фейгенбаум (колишній студент Герберта Саймона), Б. Б'юкенен (філософ за освітою) і Д. Ледербергом (лауреат Нобелівської премії в галузі генетики) почали роботи над першою експертною системою DENDRAL. У 1969 році Ст. Мартіном і Д. Мозесом була створена математична експертна система MACSYMA. Перша експертна система для медичної діагностики була створена в 1973 році Е. Шортлиффом і називалася MYCIN, вона спричинила за собою розвиток першого командного інтерпретатора EMYCIN (Ст. Мілле, Шортлифф і Б'юкенен). У 1976 році Дуда і Харт почали роботу над експертною системою PROSPECTOR, призначеної для розвідки корисних копалин, у 1984 році система точно передбачила існування місця народження молібдену, оціненого в багатомільйонну суму.

Ці експертні системи, розроблені у 60–70-х роках стали класичними. За походженням, предметних областях і по наступності застосовуваних ідей, методів та інструментальних програмних засобів їх можна розділити на кілька сімейств.

1. META-DENDRAL. Система DENDRAL дозволяє визначити найбільш ймовірну структуру хімічної сполуки за експериментальними даними (мас-спектрографії, даними ядерного магнітного резонансу та ін).

Meta-DENDRAL автоматизує процес отримання знань для DENDRAL. Вона генерує правила побудови фрагментів хімічних структур.

2. MYCIN-EMYCIN-TEIREIAS-PUFF-NEOMYCIN. Це сімейство медичних ЕС і сервісних програмних засобів для їх побудови.

3. PROSPECTOR-KAS. Система PROSPECTOR призначена для пошуку родовищ на основі геологічних аналізів. KAS – система знань для ГЕОЛОГА.

4. CASNET-EXPERT. Система CASNET – медична ЕС для діагностики видачі рекомендацій з лікування захворювань ока. На її основі розроблено мову інженерії знань EXPERT, з допомогою якої створено низку інших медичних діагностичних систем.

5. HEARSAY-HEARSAY-2-HEARSAY-3-AGE. Перші дві системи цього ряду є розвитком інтелектуальної системи розпізнавання злитої людської мови, слова якої беруться із заданого словника. Ці системи відрізняються оригінальною структурою, засновані на використанні дошки оголошень – глобальної бази даних, яка містить поточні результати роботи системи. Надалі на основі цих систем були створені інструментальні системи HEARSAY-3 і AGE (Attempt to Generalize – спроба спілкування) для побудови експертних систем.

6. Системи AM (Artifical Mathematician – штучний математик) і EURISCO були розроблені в Стенфордському університеті доктором Дугласом Ленатом для дослідних і навчальних цілей. У систему AM спочатку було закладено близько 100 правил виводу і більше ніж 200 евристичних алгоритмів навчання, що дозволяють будувати довільні математичні теорії та уявлення. EURISCO – це розвиток системи AM, з її допомогою у військово-стратегічній грі, що проводиться ВМФ США, була розроблена стратегія, що містить ряд оригінальних тактичних стратегій.

Крім розробки самих експертних систем дослідники зайнялися створенням інструментальних засобів для експертних систем: в 1983 році компанія IntelliCorp створила KEE, а в 1985 році агентство NASA випустило першу версію CLIPS.

Експертні системи швидко завоювали позиції на інформаційному ринку і набули широкого поширення. Уже в 1987 році опитування користувачів, проведене журналом Intelligent Technologies (США), показав, що приблизно:

– 25% користувачів використовують ЕС;
– 25% збираються придбати ЕС в найближчі 2-3 роки;
– 50% вважають за краще провести дослідження про ефективність їх використання.

В інформатиці експертні системи розглядаються спільно з базами знань як моделі поведінки експертів в певній галузі знань з використанням процедур логічного висновку та прийняття рішень. А бази знань – як сукупність фактів і правил логічного висновку в обраній предметній області діяльності.

Рішення спеціальних завдань вимагає спеціальних знань. Однак не кожна компанія може собі дозволити тримати у своєму штаті експертів по всім пов'язаним з її роботою проблемам або навіть запрошувати їх щоразу, коли виникає якась проблема. Головна ідея використання технології експертних систем полягає в тому, щоб отримати від експерта його знання і, завантаживши їх у пам'ять комп'ютера, використовувати кожного разу, коли в цьому виникне необхідність. Будучи одним з основних додатків штучного інтелекту, експертні системи є комп'ютерні програми, які трансформують досвід експертів з будь-якої галузі знань у форму евристичних правил (евристик). Евристики не гарантують отримання оптимального результату з такою ж упевненістю, як звичайні алгоритми, використовувані для вирішення завдань в рамках технології підтримки прийняття рішень. Однак часто вони дають в достатній мірі прийнятні рішення для їх практичного використання. Все це робить можливим використовувати технологію експертних систем у якості радників. Технологію побудови експертних систем часто називають інженерією знань. Як правило, цей процес вимагає специфічної форми взаємодії творця експертної системи, якого називають інженером знань, і одного або декількох експертів в деякій предметній області. Інженер знань «витягує» з експертів процедури, стратегії, емпіричні правила, які вони використовують при вирішенні завдань, і вбудовує ці знання в експертну систему. В результаті з'являється програма для ЕОМ, яка вирішує завдання багато в чому так само, як експерти – люди.

ЕС з успіхом застосовуються в тих областях, де крім застосування стандартних алгоритмічних методів, заснованих на точних обчисленнях, по суті знання і досвід конкретних експертів-аналітиків, а прийняття рішень формується в умовах неповноти даних і залежить скоріше від якісних, ніж кількісних оцінок. До таких предметних областей відноситься перш за все область аналізу фінансової діяльності, де ефективність прийнятих рішень залежить від зіставлення безлічі різних факторів, обліку складних причинно-наслідкових зв'язків, застосування нетривіальних логічних міркувань і т. п.

В даний час «класична» концепція експертних систем, що склалася в 70–80 роках минулого століття, переживає серйозну кризу, пов'язану з її глибокою орієнтацією на загальноприйнятий людино-машинний інтерфейс, який в даний час в користувацьких додатках майже повністю витіснена графічним (GUI). Крім того, «класичний» підхід до побудови експертних систем погано узгоджується з реляційною моделлю даних, що становить неможливо ефективне використання сучасних промислових СУБД для організації баз знань таких систем. Усі наведені в літературних та інтернет-джерелах приклади «відомих» або «поширених» експертних систем насправді відносяться до 80-х років минулого століття і в даний час давно не існують, або безнадійно застаріли і підтримуються лише нечисленними ентузіастами. З іншого боку, нерідко в якості маркетингового ходу експертними системами оголошуються сучасні програмні продукти, в «класичному» розумінні такими не є (наприклад, комп'ютерні довідково правові системи). Заходи, що вживаються ентузіастами спроби об'єднати «Класичні» підходи до розробки експертних систем з сучасними підходами до побудови призначеного для користувача інтерфейсу (проекти CLIPS Java Native Interface, CLIPS.NET та ін.) не знаходять підтримки серед великих компаній-виробників програмного забезпечення і з цієї причини залишаються поки в експериментальній стадії.

Технологію побудови експертних систем часто називають *інженерією знань*. Як правило, цей процес вимагає специфічної форми взаємодії творця експертної системи, якого називають інженером знань, і одного або декількох експертів в деякій предметній області. Інженер знань «витягує» з експертів процедури, стратегії, емпіричні правила, які вони використовують при вирішенні завдань, і вбудовує ці знання в експертну систему. В результаті з'являється комп'ютерна програма, яка вирішує завдання багато в чому так само, як експерти – люди.

Важливість експертних систем полягає в наступному:

– технологія ЕС є найважливішим засобом у вирішенні глобальних проблем традиційного програмування, перш за все, це – тривалість, і отже, висока вартість розробки складних додатків;

– висока вартість супроводу складних систем, яка часто в кілька разів перевершує вартість їх розробки;

– низький рівень повторного використовування програм.

При появі спеціальних інструментальних засобів побудови ЕС скоротилися терміни розробки, значно знизилася трудомісткість. Відповідно засоби побудови ЕС можна розділити на три основні типи:

– мови програмування;

– середовища програмування;

– порожні ЕС (оболонки).

Загальним недоліком мов програмування для створення експертних систем є:

– великий час розробки;

– необхідність залучення висококваліфікованих програмістів;

– труднощі з модифікацією готової системи.

Все це робить застосування мов програмування для реалізації ЕС досить дорогим і трудомістким.

Другий тип побудови ЕС дозволяє розробнику полегшити роботу тим, що з'являється можливість не програмувати деякі або всі компоненти ЕС, а вибирати їх із заздалегідь складеного набору.

Останній тип побудови ЕС (порожніх ЕС, або «оболонок») розробник ЕС повністю звільняється від робіт зі створення програм і займається лише наповненням бази знань.

Будь-яка експертна система включає в себе принцип накопичення знань фахівців (експертів), який будь-яким чином програмно реалізується. Після цього, на основі цих знань, користувачі ЕС, мають звичайну кваліфікацію, можуть вирішувати свої поточні завдання настільки ж успішно, як це зробили б самі експерти.

У експертної системи має бути два режими роботи:

– режим придбання знань;

– режим рішення задач.

У режимі придбання знань експерт спілкується з експертною системою за посередництвом інженера знань, в режимі вирішення завдань

в спілкуванні з експертною системою бере участь користувач, якого цікавить результат і спосіб його отримання. Експертні системи не є просто пасивним джерелом корисної інформації. Експертна система підказує необхідний напрям вирішення завдання, розвиває ланцюг умовиводів, пояснює свої дії.

3.2. Види систем та їх класифікація

Експерт – людина, яка ясно висловлює свої думки, вміє знаходити правильні рішення проблем в конкретній предметній області. *Інженер знань* – людина, що має пізнання в інформатиці та в штучному інтелекті, знає, як треба будувати експертні системи. З*асіб побудови експертної системи* – це програмні пакети, які використовує інженер знань.

Користувач:

1. Творець інструменту, відладжує засоби побудови експертної системи.
2. Інженер знань.
3. Експерт.
4. Клерк.
5. Фізичні або юридичні особи, а також програма.

На рис. 17 показані структура ЕС і система взаємодії користувачів і інженерів, що працюють з ЕС.

Серцевину експертної системи складає база знань, яка накопичується в процесі побудови ЕС. Знання повинні бути виражені в явному вигляді і організовані так, щоб спростити прийняття рішень. Важливість цієї особливості експертної системи неможливо переоцінити.

Наслідки цього процесу виходять за межі побудови програми, призначеної для вирішення певного класу задач. Причина в тому, що знання, як основа ЕС, є явними і доступними, що й відрізняє ці системи від більшості традиційних програм. БЗ мають таку ж цінність, як і будь-який великий обсяг знань, які можуть широко розповсюджуватися через книг і лекцій.

В області економіки нерухомості ця характеристика ЕС вельми актуальна, так як ринок нерухомості як об'єкт вивчення та аналізу характеризується дуже значним обсягом спеціальних знань, необхідних для раціональної діяльності. Накопичення цих знань можливо тільки протягом досить тривалого часу і з залученням широкого кола фахівців.

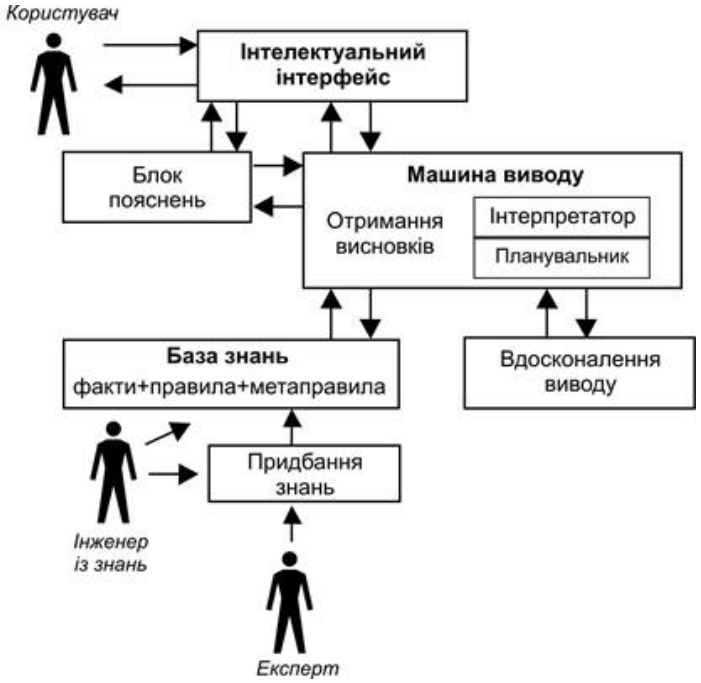


Рис. 16. Структура взаємодії ЕС і зовнішнього оточення

Найбільш корисною характеристикою ЕС є те, що вона застосовує для вирішення проблем тільки високоякісний досвід. Цей досвід може представляти рівень мислення найбільш кваліфікованих експертів в даній області, що веде до рішень творчим, точним і ефективним. Саме високоякісний досвід в поєднанні з умінням його застосовувати робить систему рентабельною, здатної заслужити визнання на ринку. Цьому сприяє також гнучкість системи. Система може нарощуватися поступово відповідно до потреб бізнесу або замовника. Це означає, що можна спочатку вкласти порівняно скромні кошти, а потім нарощувати можливості ЕС в міру необхідності.

Іншою корисною рисою експертних систем є наявність у них прогностичних можливостей. Експертна система може функціонувати в якості теорії обробки інформації або моделі рішення задачі в заданій області, даючи очікувані відповіді в конкретній ситуації і показуючи,

як зміняться ці відповіді в нових ситуаціях. Експертна система може пояснити докладно, яким чином нова ситуація привела до змін. Це дозволяє користувачеві оцінити можливий вплив нових фактів або інформації і зрозуміти, як вони пов'язані з рішенням. Аналогічно, користувач може оцінити вплив нових стратегій або процедур на рішення, додаючи нові правила або змінюючи вже існуючі.

Очевидно, що прогнозна складова методики оцінки є одночасно і найбільш важливою і найменш підкріпленою аналітичними методами. Саме прогноз зміни ціноутворюючих факторів представляється сьогодні найбільш перспективним напрямком застосування ЕС.

База знань, що визначає компетентність ЕС, може також забезпечити нову якість: інституційну пам'ять. Якщо БЗ розроблена в ході взаємодії з провідними фахівцями установи, відділу або штабу, то вона являє собою поточну політику або способи дії цієї групи людей. Цей набір знань стає зведенням дуже кваліфікованих думок і постійно оновлюється довідником найкращих стратегій і методів, використовуваних персоналом.

В області економіки нерухомості ЕС повинні надати засоби для більш стабільного державного і корпоративного управління, захистити інтереси замовника, підвищити якість послуг, що надаються. Високопрофесійні експерти зможуть зосередити свої зусилля на вирішенні найбільш складних творчих завдань, звільнившись від рутини.

Не менш важливою властивістю експертних систем є те, що їх можна використовувати для навчання і тренування керівних працівників і провідних фахівців. ЕС можуть бути розроблені з розрахунком на подібний процес навчання, так як вони вже містять необхідні знання та здатні пояснити процес свого міркування. Як інструмент навчання ЕС забезпечує нових фахівців великим багажем досвіду і стратегій, за якими можна вивчати рекомендовану політику і методи.

Загальноприйнята класифікація експертних систем відсутня, однак найбільш часто експертні системи розрізняють за призначенням, предметної області, методам представлення знань, динамічності і складності.

За призначенням класифікацію експертних систем можна провести наступним чином:

– діагностика стану систем, в тому числі моніторинг (безперервне відстеження поточного стану);

– прогнозування розвитку систем на основі моделювання минулого і сьогодення;

– планування та розробка заходів в організаційному і технологічному керуванні;

– проектування або вироблення чітких приписів щодо побудови об'єктів, які відповідають поставленим вимогам;

– автоматичне керування (регулювання);

– навчання користувачів та ін.

Класифікація експертних систем *за методами представлення знань* ділить їх на:

– традиційні;

– гібридні.

Традиційні експертні системи використовують, в основному, емпіричні моделі представлення знань і числення предикатів першого порядку.

Гібридні експертні системи використовують всі доступні методи, в тому числі оптимізаційні алгоритми і концепції баз даних.

За ступенем складності експертні системи ділять на поверхневі і глибинні. Поверхневі експертні системи представляють знання в вигляді правил «ЯКЩО-ТО». Умовою виводимості рішення є безобривність ланцюжка правил. Глибинні експертні системи мають здатність при обриві ланцюжка правил визначати (на основі метазнань) які дії слід зробити для продовження виконання завдання. Крім того, до складних належать предметні області, в яких текст запису одного правила на природній мові займає більше 1/3 сторінки.

Класифікація експертних систем *по динамічності* ділить експертні системи на:

– статичні;

– динамічні.

Предметна область називається статичною, якщо описують її вихідні дані, що не змінюються в часі. При цьому похідні дані (виведені з вихідних) можуть і з'являтися заново, і змінюватися.

Якщо вихідні дані, що описують предметну область, змінюються за час виконання завдання, то предметну область називають динамічною. В архітектуру динамічної експертної системи, в порівнянні зі статичною, вводяться два компонента підсистема моделювання зовнішнього світу і підсистема зв'язку із зовнішнім оточенням.

Підсистема зв'язку із зовнішнім оточенням здійснює зв'язок через систему датчиків і контролерів. Крім того, традиційні компоненти статичної експертної системи (база знань і механізм логічного висновку) зазнають суттєвих змін, щоб відобразити тимчасову логіку відбуваються в реальному світі подій.

3.3. Специфіка експертних системи

Особливості експертних систем, що відрізняють їх від звичайних програм, полягають в тому, що вони повинні володіти такими якостями.

1. *Компетентністю*, а саме:

– досягати експертного рівня рішень, тобто в конкретній предметній області мати той же рівень професіоналізму, що й експерти-люді;

– бути вмілою, тобто застосовувати знання ефективно і швидко, уникаючи, як і люди, непотрібних обчислень;

– мати адекватну робастність, тобто здатність лише поступово знижувати якість роботи у міру наближення до границь діапазону компетентності або припустимої надійності даних.

2. *Можливістю до символьних міркувань,* а саме:

– представляти знання в символьному вигляді;

– переформульовувати символьні знання. Мовою штучного інтелекту символ – це рядок знаків, стосується вмісту деякого поняття. Символи об'єднують, щоб висловити відносини між ними. Коли відносини представлені в експертній системі, вони називаються символьними структурами.

3. *Глибиною*, а саме:

– працювати в предметної області, що містить важкі завдання;

– використовувати складні правила, тобто використовувати або складні конструкції правил, або велику їх кількість.

4. *Самосвідомістю*, а саме:

– досліджувати свої міркування, тобто перевіряти їх правильність;

– пояснювати свої дії.

Існує ще одна важлива особливість експертних систем. Якщо звичайні програми розробляються так, щоб кожен раз породжувати правильний результат, то експертні системи розроблені з тим, щоб вести себе як експерти. Вони, як правило, дають правильні відповіді, але іноді, як і люди, здатні помилятися.

Переваги експертних систем перед людиною-експертом

1. Сталість.

Людська компетенція слабшає з часом. Перерва в діяльності людини-експерта може серйозно відбитися на його професійні якості.

2. Легкість передачі.

Передача знань від однієї людини іншій – довгий і дорогий процес. Передача штучної інформації – це простий процес копіювання програми або файлу даних.

3. Стійкість і відтворюваність результатів.

Експертні системи стійкі до «перешкод». Людина ж легко піддається впливу зовнішніх факторів, які безпосередньо не пов'язані з завданням. Експерт-людина може приймати в тотожних ситуаціях різні рішення через емоційні чинники. Результати експертної системи – стабільні.

4. Вартість.

Експерти, особливо висококваліфіковані, обходяться дуже дорого. Експертні системи, навпаки, порівняно недорогі. Їх розробка дорога, але вони дешеві в експлуатації.

Разом з тим розробка експертної системи не дозволяє повністю відмовитися від експерта-людини. Хоча експертна система добре справляється зі своєю роботою, тим не менш, в певних областях людська компетенція явно перевершує штучну. Однак і в цих випадках експертна система може дозволити відмовитися від послуг висококваліфікованого експерта, залишивши експерта середньої кваліфікації, використовуючи при цьому експертну систему для посилення і розширення його професійних можливостей.

Обмеженість застосування експертних систем

Існують три фактори, від яких залежить остаточна відповідь питання про здатність ЕС допомогти користувачеві, – природа проблеми, наявність певного досвіду в тій предметної області, до якої належить проблема, і можливість зіставлення результатів аналізу проблеми і наявного досвіду методом, доступним комп'ютерній програмі. Перед прийняттям рішення про створення експертної системи слід задуматися над наступним: чи доступний експерт, який: здатний вирішити проблему; знає, як вирішується проблема; здатний пояснити, як вирішується проблема; має часу, щоб пояснити,

як вирішується проблема; має достатні спонукальні мотиви до активної участі в створенні експертної системи.

Прикладом, може служити, прогноз погоди – це не те завдання, яке може вирішити будь-хто, навіть навчений великим досвідом експерт. Розпізнавання мови – це завдання, яке вирішує практично кожен, але ніхто з нас (включаючи і професійних лінгвістів) не може зрозуміло пояснити, як це робиться. А тому використовувати для вирішення цієї проблеми методи, засновані на аналізі знань, навряд чи вдасться. Тут більшого слід очікувати від статистичного моделювання. Навіть маючи на прикметі геніального експерта, який знає, як вирішується задача, не можна розраховувати на успіх, якщо цей експерт не може або не бажає детально і зрозуміло пояснити, як він це робить. Експерт може бути не розташований до спілкування зі сторонніми або занадто зайнятий, щоб втрачати час на тривалі співбесіди з інженером, якому доручено проектування бази знань. Як правило, експерт високого класу не відчуває нестачі в пропозиціях роботи в тій області, з якої він добре знайомий, а тому вважає за краще виконувати її, а не вести розлогі бесіди про те, як він це робить. Є ще й психологічний фактор – багато експертів досить ревниво ставляться до свого унікального досвіду і не схильні його розголошувати, оскільки вважають (і нам нема чого заперечити їм), що, передаючи досвід автоматизованих систем, вони «рубають сук», на якому сидять.

Але навіть якщо вдасться виконати обумовлені вище умови, в завданню можуть існувати чинники, що обмежують можливість «машинного» відтворення людського досвіду. наприклад:

– в процесі виконання завдання використовуються здатності органів почуттів людини, що є досить складною задачею для машини;

– в рішення задачі залучені міркування здорового глузду людства або великий обсяг знань, само собою зрозумілих для будь-якої людини.

Дуже важливо відокремити ті знання, володіння якими характерно саме для експерта в певній галузі, від тих знань, які відомі кожному, що виконує в цій області рутинну роботу. Керування автомобілем при їзді по забитим транспортом вулицями вимагає не стільки знань експерта, скільки вміння миттєво оцінювати ситуацію і швидко на неї реагувати.

3.4. Засоби розробки експертних систем

Існуючі засоби розробки експертних систем можна розділити на 3 класи:

1) Традиційні мови програмування (C ++, Java, Delphi) дозволяють побудувати експертні системи «з нуля» для конкретної задачі або предметної області, забезпечивши хороші показники якості і необхідну функціональність системи, але на розробку потрібні значні часові та фінансові ресурси. Так створюють експертні системи будь-якої стадії існування, особливо комерційні системи, продаж яких відшкодує витрати.

2) *Мови штучного інтелекту* (LISP, PROLOG, Рефал) були розроблені спеціально для представлення знань. Побудова з їх допомогою експертних систем дозволяє більш легко оперувати експертними знаннями, але обмежує спосіб їх подання структурою мови. За допомогою мов штучного інтелекту створюються дослідницькі та демонстраційні зразки.

3) Наступний клас засобів побудови експертних систем – спеціальний програмний інструментарій – орієнтований тільки на створення інтелектуальних інформаційних систем і ділиться на два під-класи: оболонки і середовища розробки інтелектуальних систем.

Середовище розробки є програмними комплексами, що дозволяють будувати системи з окремих готових блоків. На їх основі створюються демонстраційні і промислові зразки експертних систем.

Оболонка експертних систем – інструментальний засіб для проектування і створення експертних систем. До складу оболонки входять засоби проектування бази знань з різними формами представлення знань і вибору режиму роботи решателю завдань. Для конкретної предметної області інженер по знаннях визначає потрібне представлення знань і стратегії вирішення завдань, а потім, вводячи їх в оболонку, створює конкретну експертну систему.

Застосування оболонки дозволяє досить швидко і з мінімальними витратами створити дослідницьку, демонстраційну або промислову експертну систему. За ступенем роботи виділяють експериментальну (GPSI), дослідницьку (Expert) і комерційну (EXSYS) оболонки.

Знання в базі можуть бути представлені одним способом (EMYCIN, CLIPS) – семантичної мережею, продукціям, фреймами і т. п. або ж декількома (MINEVRA, EsWin) для створення більш повної, гнучкою і наочної моделі предметної області.

Використовувані в оболонці методи можуть бути традиційними (CubiCalc, NEXPERT, Алеф) – алгоритми, дерева виводу і т. п. і гібридними (FuzzyCLIPS, MultiNeuron), де спільно з традиційними використовуються нейронні мережі, нечітка логіка і т. п.

Існують статичні оболонки, призначені для вирішення статичних задач (1-st Clas, Еліс). Вони характеризуються використанням поверхневої технології, загальних правил і пошуку рішення від мети до даних, застосовуються для вирішення завдань аналізу.

Статичні оболонки, призначені для вирішення завдань аналізу та синтезу з поділом часу (KAPPA, Clips), використовують глибинний і структурний підходи, здійснюють пошук рішень від мети до даних і від даних до мети.

Оболонки для проектування динамічних систем (Framework, NExpert) застосовують поверхневий підхід, приймають рішення на основі правил загального вигляду.

Оболонки для розробки динамічних систем (G2, Rethink, RkWorks) мають підсистему моделювання, планувальника рішень, використовують змішану технологію, правила загального і приватного виду, рішення задачі аналізу та синтезу в реальному часі.

EMYCIN – перша оболонка, заснована на MYCIN. Принципи, розроблені для PROSPECTOR, були використані при створенні таких систем, як KAS, SAGE, SAVOIR.

Зміна принципів побудови веде до розвитку інструментарію. Тому оболонки пройшли той же еволюційний шлях, що і ЕС. Сучасні оболонки пропонують наступні можливості (в кожній конкретній оболонці представлені частково):

– гібридне представлення знань (EsWin);
– вибір з кількох стратегій виведення (G2, CLIPS);
– підключення бібліотек та інших систем (ACTIVATION FRAMEWORK);
– архітектура на основі «дошки оголошень» (HEARSAY-III);
– архітектура «клієнт-сервер» (JESS);

– інтеграція в Інтернет / Інтранет (Egg2Lite, Exsys Corvid);
– графічний інтерфейс (WindExS, WxCLIPS);
– підсистема моделювання (G2);
– модульне побудова системи (ReThink, G2);
– візуалізація структури БЗ (W.E.S.T.) і т. п.

3.5. Завдання для самоконтролю

1. Як називалася перша експертна система?
А) MACSYMA
Б) EMYCIN
В) PROSPECTOR
Г) нема правильної відповіді.

2. Яке завдання вирішувала експертна система PROSPECTOR?
А) визначення найбільш ймовірної структури хімічної сполуки.
Б) пошук родовищ на основі геологічних аналізів.
В) діагностика очних захворювань.
Г) розпізнавання злитої людської мови.
Д) немає правильної відповіді.

3. Які підсистеми є для експертної системи обов'язковими?
А) база знань
Б) інтерфейс системи із зовнішнім світом
В) алгоритмічні методи рішень.
Г) інтерфейс когнітолога
Д) контекст предметної області

4. Для вирішення яких завдань призначені статичні оболонки експертних систем?
А) для управління і діагностики в режимі реального часу.
Б) для розв'язування статичних задач
В) для розв'язування задач аналізу і синтезу з розподілом часу.
Г) для розробки динамічних систем
Д) нема правильної відповіді.

5. Гібридна експертна система це коли присутнє
А) використання декількох засобів розробки
Б) використання різних підходів до програмування
В) використання декількох методів представлення знань
Г) нема правильної відповіді

6. Хто створює базу знань експертної системи?
А) програміст
Б) користувач
В) когнітолог
Г) експерт

3.6. Лабораторний практикум

3.6.1. Лабораторна робота 6. Експертні системи

Мета: сформувати поняття про експертні системи і головні моделі подання знань, навчитися проектувати найпростіші експертні системи.

Теоретичні відомості

Експертна система (ЕС) – це програма, яка оперує знаннями деякої предметної галузі з метою вироблення рекомендацій або розв'язання проблем. Експертна система може взяти на себе функції, які зазвичай потребують залучення досвіду людини-експерта, або грати роль асистента для людини, яка приймає рішення. Для експертних систем є типовими такі задачі:

– виділення корисної інформації з первинних даних;
– діагностика несправностей механізмів та хвороб людини;
– структурний аналіз складних об'єктів;
– вибір конфігурації складних багатокомпонентних систем;
– планування послідовності виконання операцій.

В загальному випадку експертна система складається з таких компонентів:

– база знань – зберігає дані і правила, які описують предметну галузь;
– машина логічного виводу – розв'язує поставлену задачу, використовуючи базу знань;
– компонент набуття знань – дозволяє розширювати базу знань;
– компонент пояснення – забезпечує пояснення, як саме була отримана відповідь на питання і які знання при цьому використовувались;
– інтерфейс з користувачем.

Структурні компоненти повинні забезпечувати такі функції ЕС:

– набуття знань;
– подання (зображення) знань;

– розв'язування задач (можливо, використовуючи не повністю визначені, достовірні дані);

– пояснення прийнятого рішення.

Експертна система характеризується способом *представлення знань*. Теорія представлення знань – це окрема галузь досліджень, яка вивчає методи асоціативного зберігання інформації, подібні до тих, які існують у мозку людини. Розглянемо деякі моделі представлення знань.

Продукційна модель (модель, що ґрунтується на правилах) дозволяє описувати знання у вигляді правил (продукцій) вигляду Якщо <умова> то <висновок>. База знань складається з набору таких правил. Продукційна модель найчастіше використовується у промислових експертних системах. Вона характеризується наочністю, модульністю, простотою внесення доповнень і простотою механізму логічного виводу.

Семантична мережа – це орієнтований граф, вершинами якого є поняття, а дугами – відношення між ними. Поняттями зазвичай бувають абстрактні або конкретні об'єкти. Найчастіше відношення визначаються такими типами зв'язків:

– «частина-ціле» («клас-підклас», «елемент-множина»);
– функціональні (впливає);
– кількісні (більше, менше, дорівнює);
– просторові (близько, далеко, за, під, над);
– часові (раніше, пізніше, протягом);
– атрибутивні(мати властивість, мати значення);
– логічні (і, або).

Головною перевагою цієї моделі є відповідність сучасним уявленням про організацію пам'яті людини, недоліком – складність пошуку на семантичній мережі. Семантичні мережі використовуються в якості мови подання знань в таких ЕС, як PROSPECTOR, CASNBT, TORUS.

Фреймова модель була запропонована М. Мінським в 70-х роках. *Фрейм* – це структура даних для зображення стереотипних ситуацій. Всі знання про даний клас об'єктів або подій концентруються в єдиній структурі. Фрейм складається зі *слотів*, які зберігають значення атрибутів об'єкта. Кожен слот має ім'я і значення. Значенням слоту може бути практично будь-який об'єкт – числа, математичні вирази, тексти, посилання на інші слоти цього або іншого фрейму. Розрізняють

фрейми-прототипи, які зберігаються в базі знань, і фрейми-екземпляри, які створюються для відображення реальних ситуацій на основі отриманих даних. При переході від прототипу до екземпляру (конкретизації фрейму) самому фрейму і його слотам надаються конкретні імена та відбувається заповнення слотів.

Перевагами фреймів як моделі представлення знань є здатність відображати концептуальну основу організації пам'яті людини, гнучкість і наочність. Спеціальні мови представлення знань в мережах фреймів FRL (Frame Representation Language) та інші дозволяють ефективно будувати промислові ЕС. Відомими фрейм-орієнтованими експертними системами є ANALYST і МОДИС.

Практична частина

Напишемо просту експертну систему «Визначник тварин», яка буде за даними характеристиками визначати відповідну тварину.

Нам знадобляться наступні вбудовані предикати:

`write/?` – виводить передані аргументи у вікні Dialog. Число аргументів може бути будь-яким. Атоми, числа, структури виводяться так, як записані в програмі, замість аргументів-змінних виводиться їх значення.

`nl/0` – переводить курсор у вікні Dialog на новий рядок.

`assert/1` – додає переданий в якості аргументу *факт* в кінець програми. При цьому сам файл з програмою залишається незмінним, факти лише завантажуються в пам'ять.

`retract/1` – видаляє з пам'яті всі факти, які уніфікуються з переданим фактом-аргументом.

Предикати `assert/1` і `retract/1` дозволяють змінювати програму (або базу даних) в ході її ж виконання.

Наша ЕС буде використовувати продукційну модель подання знань. Кожна тварина описуватиметься правилами вигляду

```
тварина(пінгвін):-
клас(птах), %пінгвін – птах,
невірно("літає"), % що не літає,
вірно("плаває"), %а плаває
вірно("має чорно-біле забарвлення"), !. %до того ж чорно-
білий
```

```
клас(птах):-
вірно("має пір'я"), !. %птахи мають пір'я

клас(птах):-
вірно("літає"),
вірно("кладе яйця"), !. %або літають і кладуть яйця
```

Відсікання в кінці кожного правила потрібні для того, щоб Пролог-система не перевіряла зайвих альтернатив і знаходила лише одну тварину (наприклад, якщо стало відомо, що тварина має пір'я, то зрозуміло, що вона є птахом і немає сенсу перевіряти, чи кладе вона яйця).

Діалог з користувачем організуємо наступним чином: система буде ставити йому запитання щодо особливостей задуманої ним тварини, а користувач буде на них відповідати «так» або «ні». Оскільки різні тварини можуть мати однакові характеристики, то питання системи можуть повторюватися. Щоб запобігти цьому будемо зберігати відповіді користувача у вигляді фактів `відповідь_так(X)` (у випадку позитивної відповіді) та `відповідь_ні(X)` (для негативної відповіді). Тут X – деяка властивість тварини. Власне збереження відповідей в пам'яті реалізує предикат `запам'ятати/2`:

```
запамятати(X,так):-assert(відповідь_так(X)).  %запам'ятати позитивну відповідь – додамо відповідний факт
запамятати(X,ні):-assert(відповідь_ні(X)). %аналогічно з негативною
```

Процес відгадування задуманої тварини проходитиме наступним чином. Викликом мети тварина(X) запускатиметься процес перебору всіх тварин з бази знань. При перевірці кожної тварини викликатимуться відповідні предикати `вірно/1` або `невірно/1`. Ці предикати повинні визначити наявність або відсутність даної характеристики у тварини. Якщо людина вже дала відповідь (позитивну або негативну) стосовно цієї характеристики, то її можна знайти в базі знань, інакше потрібно задати відповідне питання і додати відповідь до бази. При цьому виклик `вірно/1` повинен завершитися успішно лише у випадку наявності даної характеристики. Виклик `невірно/1` навпаки, завершується успіхом, якщо відомо, що задумана тварина не має даної властивості.


```
вірно(X):-відповідь_так(X),!. %вже була дана позитивна відповідь, перевірка закінчена
вірно(X):-
%якщо була негативна відповідь, то перевірка буде неуспішною (not має неуспіх).
     not(відповідь_ні(X)),
%інакше (якщо ще не було ніякої відповіді), спитаємо людину
     спитати(X,так).
/*якщо відповідь була позитивною, то спитати(X, так) завершиться успішно,і весь предикат вірно/1 поверне істину; якщо ж відповідь негативна то вся перевірка буде неуспішною*/

%для невірно/1 все навпаки
невірно(X):-відповідь_ні(X),!. %була дана негативна відповідь
невірно(X):-
     not(відповідь_так(X)),  %у випадку наявності позитивної відповіді в базі знань перевірка завершується неуспішно
     спитати(X,ні). %інакше спитаємо користувача
```

Діалог з користувачем реалізує предикат `спитати/2`:

```
спитати(X,Answer):- %X – властивість, Answer – відповідь
     write("Воно ", X, "? (y/n)"),nl, %виведемо запитання
     readln("y"),!,  %якщо користувач ввів "y" (позитивна відповідь),
     запамятати(X,так),Answer=так; %то запам'ятаємо її
     запамятати(X,ні),Answer=ні. %інакше запам'ятаємо негативну відповідь
```

Після того, як тварина визначена або не визначена (не знайдена в базі), потрібно видалити всі факти – відповіді людини, щоб при наступному запуску ЕС пам'ять була чистою. Для цього запишемо предикат очистити/0:

```
очистити:-retract(відповідь_так(_)),fail. %видалимо всі факти відповідь_так
очистити:-retract(відповідь_ні(_)),fail. %видалимо всі факти відповідь_ні
очистити. %врешті-решт виклик очистити/0 закінчимо успішно
```

Retract/1 за один виклик видаляє один факт з бази, тому за допомогою fail задіємо механізм повернень. Тепер retract/1 буде викликатися доти, доки ще є відповідні факти в базі знань. Коли всі факти буде видалено, retract/1 завершиться невдачею, і система почне перевіряти інші альтернативи для очистити/0. Для того, щоб очистити/0 після видалення всіх фактів повернув істину, ми додали відповідний факт.

Вся програма контролюється такими предикатами:

```
старт:-
     тварина(X), %запустимо процес визначення тварини
     write("Мабуть, це ",X, "."),nl,nl, %виведемо відповідь
     очистити. %видалимо всі дані про відповіді людини
старт:-
/*якщо в базі немає тварини з вказаними характеристиками, то
виклик тварина(X) завершиться невдачею, і ми потрапимо сюди*/
     write("Я не знаю такої тварини."),nl,nl,
     очистити. %видалимо всі дані про відповіді людини

експерт:-
     старт,!, %почнемо діалог
     write("Спробуємо ще? (y/n)"),nl,
     readln("y"), експерт. %якщо людина хоче продовжувати, то
почнемо все з початку
```

Експертна система запускається запитом

```
експерт.
```

Отже, повний текст програми буде таким:

```
тварина(гепард):-
клас(ссавець),
клас("м'ясоїд"),
вірно("має рудий колір"),
вірно("має темні плями"),!.

тварина(тигр):-
клас(ссавець),
клас("м'ясоїд"),
вірно("має рудий колір"),
вірно("має темні смуги"),!.
```



```
тварина(жираф):-
клас(копитне),
вірно("має довгу шию"),
вірно("має довгі ноги"),
вірно("має темні плями"),!.

тварина(зебра):-
клас(копитне),
вірно("має темні смуги"),!.

тварина(страус):-
клас(птиця),
невірно("літає"),
вірно("має довгу шию"),
вірно("має довгі ноги"),
вірно("має чорно-біле забарвлення"),!.

тварина(пінгвін):-
клас(птиця),
невірно("літає"),
вірно("плаває"),
вірно("має чорно-біле забарвлення"),!.

тварина(альбатрос):-
клас(птиця),вірно("добре літає"),!.

клас(ссавець):-
вірно("має шерсть"),!.

клас(ссавець):-
вірно("годує дитинчат молоком"),!.

клас(птиця):-
вірно("має пір'я"),!.
```

```
клас(птиця):-
вірно("літає"),
вірно("кладе яйця"),!.

клас("м'ясоїд"):-
вірно("їсть м'ясо"),!.

клас("м'ясоїд"):-
вірно("має гострі зуби"),
вірно("має пазурі"),
вірно("має очі, спрямовані вперед"),!.

клас(копитне):-
клас(ссавець),
вірно("має копита"),!.

клас(копитне):-
клас(ссавець),
вірно("є жвачним"),!.

вірно(X):-відповідь_так(X),!.  %вже була дана позитивна відповідь, перевірка закінчена
вірно(X):-
%якщо була негативна відповідь, то перевірка буде неуспішною (not має неуспіх).
     not(відповідь_ні(X)),
%інакше (якщо ще не було ніякої відповіді), спитаємо людину
     спитати(X,так).
/*якщо відповідь була позитивною, то спитати(X, так) завершиться успішно, і весь предикат вірно/1 поверне істину; якщо ж відповідь негативна то вся перевірка буде неуспішною*/

%для невірно/1 все навпаки
невірно(X):-відповідь_ні(X),!. %була дана негативна відповідь
невірно(X):-
```



```
     not(відповідь_так(X)), %у випадку наявності позитивної відповіді в базі знань перевірка завершується неуспішно
     спитати(X,ні). %інакше спитаємо користувача

спитати(X,Answer):- %X - властивість, Answer - відповідь
     write("Воно ", X, "? (y/n)"),nl, %виведемо запитання
     readln("y"),!,  %якщо користувач ввів "y"  (позитивна відповідь),
     запамятати(X,так),Answer=так; %то запам'ятаємо її
     запамятати(X,ні),Answer=ні.     %інакше     запам'ятаємо негативну відповідь

запамятати(X,так):-assert(відповідь_так(X)).
запамятати(X,ні):-assert(відповідь_ні(X)).

очистити:-retract(відповідь_так(_)),fail. %видалимо всі факти відповідь_так
очистити:-retract(відповідь_ні(_)),fail. %видалимо всі факти відповідь_ні
очистити. %врешті-решт виклик очистити/0 закінчимо успішно

старт:-
     тварина(X), %запустимо процес визначення тварини
     write("Мабуть, це ",X, "."),nl,nl, %виведемо відповідь
     очистити. %видалимо всі дані про відповіді людини
старт:-
/*якщо в базі немає тварини з вказаними характеристиками, то виклик тварина(X) завершиться невдачею, і ми потрапимо сюди*/
     write("Я не знаю такої тварини."),nl,nl,
     очистити. %видалимо всі дані про відповіді людини

експерт:-
     старт,!, %почнемо діалог
     write("Спробуємо ще? (y/n)"),nl,
     readln("y"), експерт. %якщо людина хоче продовжувати, то почнемо все з початку
```

```
/*коли в базі ще немає жодного факту відповідь_так або відповідь_ні, звернення до цих предикатів викличе помилку. Тому потрібно записати наступні два речення*/
відповідь_так(_):-fail.
відповідь_ні(_):-fail.
```

Випробувати предикати для роботи з базою даних.

1. Додамо декілька фактів в базу даних. Запустіть інтерпретатор і, не відкриваючи ніякої програми, запустіть з вікна Dialog такі цілі:

```
assert(факт(1, "один")).
assert(факт(1, "ще один")).
assert(факт(2, "два")).
assert(факт(3, "три")).
```

2. Тепер перевіримо ці факти. Поставте такий запит:

```
факт(X, Y).
```

Ми додавали чотири факти, отже ви повинні отримати чотири розв'язки.

3. Тепер почнемо видаляти факти з пам'яті:

```
retract(факт(1, Y)).
```

Із аргументом уніфікуються два факти з бази, вони і будуть видалені – отримаємо два розв'язки. Тепер запит

```
факт(X, Y).
```

поверне значення лише з тих двох фактів, що залишилися.

4. Нарешті, видалимо всі інші факти:

```
retract(факт(X, Y)).
```

Спробуйте тепер спитати:

```
факт(X, Y).
```

Створити експертну систему «Визначник тварин» та перевірити її роботу.

1. Створіть нову програму, напишіть текст нашої експертної системи та збережіть її.

2. Запустіть експертну систему запитом

```
експерт.
```

Програма буде задавати питання у вікні Dialog. Для того, що відповісти на питання, потрібно у цьому вікні ввести «y» або «n» та натиснути Enter.

3. Перевірте, чи всі тварини правильно визначаються.

Завдання до лабораторної роботи 6

Розробіть свою експертну систему за обраною темою (у кожного студента своя). Можливі теми:

1. Визначник акваріумних риб.
2. Визначник рослин (квіток).
3. Вибір професії.
4. Вибір місця відпочинку.
5. Визначник марки автомобіля (мотоцикла або ін.).
6. Вибір і купівля автомобіля.
7. Діагностика несправності автомобіля (або іншого пристрою).
8. Вибір і купівля комп'ютера.
9. Діагностика захворювань людини.
10. Прогнозування погоди.
11. Тема за вашим вибором.

Контрольні запитання

1. Що таке експертна система?
2. Які завдання розв'язуються за допомогою експертних систем?
3. Яка структура експертної системи?
4. Які функції виконуються експертною системою?
5. Охарактеризуйте основні моделі подання знань.
6. Як відбувається процес визначення тварини в програмі «Визначник тварин»?
7. Як в програмі «Визначник тварин» запам'ятовуються відповіді людини і як видаляються дані про них?
8. Опишіть вбудовані предикати для роботи з базою даних.
9. Які факти знаходяться в пам'яті під час виконання запитів з першої частини практичних завдань? Поясніть відповіді інтерпретатора.

3.6.2. Лабораторна робота 7. Аналіз природньої мови

Мета: вивчити рівні та етапи аналізу природної мови, навчитися проектувати найпростіші синтаксичні аналізатори та інтерфейси на природній мові.

Теоретичні відомості

Проблема розуміння штучної мови виникає при реалізації інтерфейсу з базою даних, системи автоматичного перекладу, програми інте-

рактивного навчання, системи витягу інформації з тексту та автоматичного реферування.

Розуміння мови містить в собі такі різноманітні процеси, як розпізнавання звуків та друкованих літер, синтаксичний розбір, вивід семантик високого рівня. Виділяють наступні рівні аналізу природної мови:

1. Просодія – аналіз ритму та інтонації мови.
2. Фонологія – наука про звуки та їх комбінації в мовних структурах.
3. Морфологія – аналіз компонентів (морфем), з яких складаються слова.
4. Синтаксичний аналіз пов'язаний з вивченням правил сполучення слів в окремих фразах та реченнях, а також з використанням цих правил для розбору та генерації речень.
5. Семантика вивчає зміст слів, фраз та речень.
6. Прагматика – наука про способи використання мови та її вплив на слухача.
7. Знання про навколишній світ.

Стадії аналізу мови є такими:

1. Синтаксичний розбір – визначення лінгвістичної структури речення, формування *дерева розбору*. В синтаксичному аналізаторі використовуються знання синтаксису мови, морфології та, частково, семантики.

2. Семантична інтерпретація – формування представлення змісту тексту. В процесі семантичної інтерпретації використовуються знання про значення слів та лінгвістичну структуру.

3. Інтерпретація знань про навколишній світ – структури з бази знань додаються до внутрішнього представлення речення для формування розширеного представлення змісту речення.

В цій лабораторній роботі напишемо простий синтаксичний аналізатор, який буде аналізувати два види питань на українській мові та систему обробки проаналізованих речень, яка буде формувати запити до створеної в попередній роботі експертної системи. Таким чином, ми створимо найпростіший інтерфейс до експертної системи на природній мові.

Система працюватиме з двома типами питань:

1. `Який|Яка <назва_тварини>?` – система повинна вивести набір характеристик вказаної тварини.

2. `Хто <список_ознак>?` – система повинна визначити тварину, яка характеризується вказаними ознаками.

Найрозповсюдженим підходом до реалізації синтаксичного аналізу засобами Пролога є використання *граматик, що задаються означальними реченнями* (definite clause grammar, DCG). Ми будемо використовувати граматику, що визначається такими правилами:

```
речення→ група_підмета, група_присудка.
група_підмета→ «Хто»
група_підмета→ займенник, підмет.
група_присудка→ список_ознак.
список_ознак→ ознака, список_ознак.
ознака→ «...».
ознака→ «не», ознака.
займенник→ «Який».
займенник→ «Яка».
підмет→ тварина. (будь-яка відома програмі тварина)
```

Речення складається з групи підмета та групи присудка. Група підмета може бути займенником «Хто» (тоді група присудка є списком ознак деякої тварини) або складатися з займенника «Який» чи «Яка» та підмета – назви тварини.

На основі цих правил система будуватиме дерево розбору даного речення. Наприклад, для речення

Хто має хутро, має копита та має темні смуги?

дерево буде таким:

Рис. 17. Дерево розбору речення

Для розбиття речення, введеного з клавіатури, на слова використаємо вбудований предикат `str_toklist(String, TokenList)`, який повертає список слів та розділових знаків `TokenList` вхідного рядка `String`. Наприклад, для нашого прикладу отримаємо такий список:

```
[Хто,має,хутро,,,має,копита,,,має,темні,смуги,?,]
```

Подальший аналіз цього списку відбуватиметься за допомогою вищеназваних правил граматики, переписаних на Пролог. Кожне правило граматики на Пролозі матиме принаймні три аргументи: вхідне речення (список слів), залишок вхідного речення (те, що залишилося від вхідного речення після виділення з нього синтаксичної одиниці, що описується даним правилом) та частина дерева розбору, яка визначає виділену синтаксичну одиницю.

Наприклад, для синтаксичної одиниці «речення» правило буде таким:

```
%аналіз речення. Sent – вхідний рядок, Rest – те, що залишилось
від нього після виділення з нього одного речення
речення(Sent, Rest, речення(SubjectGroup, PredicateGroup)):-
    група_підмета(Sent, Rest1, SubjectGroup), %SubjectGroup –
група підмета, Rest1 – вхідний рядок без групи підмета
    група_присудка(Rest1, Rest, PredicateGroup). %тепер з того,
що залишилося від вхідного рядка, виділимо групу присудка
```

Група підмета задається двома правилами:

```
%підметом може бути займенник "Хто"
група_підмета(["Хто"|Rest], Rest, група_підмета("Хто")).
%група підмета також може складатися з займенника та підмета –
іменника, що визначає тварину
група_підмета([Pronoun, Animal|Rest], Rest, група_
підмета(займенник(Pronoun), підмет(Animal))):-тварини(Animal),
займенник(Pronoun).
```

В групі присудка потрібно знайти список ознак:

```
група_присудка([X|Rest], Rest, група_присудка(список_
ознак([]))):-кінець(X), !. %дійшли до кінця речення
група_присудка(Sent, Rest, група_присудка(список_
ознак(List))):-список_ознак(Sent, Rest, List). %побудуємо
список ознак
```


Ознаки у вхідному рядку розділяються комами або сполучниками «і» або «та». Кожна ознака може складатися з одного чи декількох слів. Перед ознакою може стояти частка «не». Рекурсивний предикат `список_ознак/3` виділятиме по одній ознаці за виклик. Рекурсія скінчиться, коли буде досягнутий кінець речення. На зворотному ході рекурсії буде побудований список зі всіх виділених ознак.

```
список_ознак([X|Sent], Sent, []):-кінець(X). %дійшли до кінця
речення
список_ознак(Sent, Rest, [X|List]):-
    ознака(Sent, Rest1, X, так), %спробуємо визначити ознаку,
вважаючи її позитивною
    список_ознак(Rest1, Rest, List). %продовжимо розбір речення
```

Власне ознака виділяється предикатом `ознака/4`. Предикат має чотири аргументи – вхідне речення, його залишок після виділення ознаки, вихідну структуру для дерева розбору і один додатковий аргумент для відстеження наявності частки «не» в ознаці. На початку визначення ознаки йому присвоюється значення «так». Якщо буде знайдена частка «не», то цьому аргументу надається значення «ні». Вихідна структура (третій аргумент) предикату має такий вигляд:

```
ознака(Text, Sign)
```

де Text – текст властивості тварини, Sign – «так» або «ні» в залежності від наявності частки «не» в ознаці в тексті речення.

Предикат `ознака/4` буде рекурсивно рухатися по реченню, доки не зустріне сполучник – роздільник ознак в реченні або кінець речення. На цьому етапі у вихідну структуру буде записано визначену в процесі рекурсії ознаку Sign та почнеться зворотній хід рекурсії, на якому з виділених слів буде побудовано повний текст ознаки.

```
%якщо зустріли частку "не", то продовжимо розгляд ознаки,
враховуючи її негативність
ознака(["не"|Sent], Rest, ознака(X, Sign), _):-ознака(Sent,
Rest, ознака(X, Sign), ні),!.
%якщо після поточного слова стоїть сполучник або кінець
речення, то закінчимо рекурсивний розгляд поточної ознаки,
%запишемо останнє слово ознаки та її позитивність/
негативність
ознака([X, Y|Sent], [Y|Sent], ознака(X, Sign), Sign):-
кінець(Y).
```

```
ознака([X, Y|Sent], Sent, ознака(X, Sign), Sign):-
сполучник(Y),!.
%розгляд ознаки
ознака([X|Sent], Rest, ознака(Y, Sign), CurrSign):-
     ознака(Sent, Rest, ознака(Y1, Sign), CurrSign),
%рекурсивно виберемо слова ознаки до першого сполучника або
кінця речення
     concat(X, " ", Str), concat(Str, Y1, Y). %додамо до
отриманої ознаки спереду поточне слово
```

Вбудований предикат `concat(String1, String2, String3)` додає до рядку String1 рядок String2, результат – рядок String3.

Залишилося визначити словник програми:

```
кінець("?"). %кінцівка речення
%сполучники для списку ознак
сполучник(",").
сполучник("і").
сполучник("та").
%займенники
займенник("Який").
займенник("Яка").
%відомі тварини
тварини(гепард).
тварини(тигр).
тварини(жираф).
тварини(зебра).
тварини(страус).
тварини(пінгвін).
тварини(альбатрос).
```

Після аналізу за допомогою цих правил нашого речення ми отримаємо його деревоподібну структуру, придатну для подальшої обробки:

```
речення(група_підмета(Хто),група_присудка(список_
ознак([ознака(має  хутро,так)|[ознака(має  копита,так)|[ознака
(має темні смути,так)|[]]]])))
```

Після аналізу структури речення потрібно визначити його зміст і спробувати знайти відповідь в базі знань на поставлене в ньому питання (база знань залишається такою самою, як і в попередній

лабораторній роботі). Ці дії виконуватиме предикат обробити/1, якому передається визначена структура речення.

Спочатку треба визначити тип поставленого питання. Це легко зробити, оскільки два види питань мають різну структуру.

Для знаходження відповіді на питання першого типу (які характеристики даної тварини) потрібно викликати предикат `тварина(X)` (де X – змінна, конкретизована значенням даної тварини) і потім при виклику відповідних предикатів `вірно/1` або `невірно/1` просто вивести відповідні ознаки на екран.

```
%питання першого типу – які ознаки тварини
обробити(речення(група_підмета(займенник(_), підмет(Animal)),
_)):-
  assert(тип_питання(1)), %запам'ятаємо тип питання для
використання в предикатах вірно/невірно
  write(Animal, " "), %виведемо назву тварини
  тварина(Animal),nl,nl. %та всі її властивості

%якщо розглядається перший тип питань, то просто виведемо
поточну ознаку
вірно(X):-тип_питання(1), write(X, ", ").
невірно(X):-тип_питання(1), write("не ", X, ", ").
Для другого типу питань (яка тварина має задані характеристики)
потрібно додати ознаки зі списку ознак до програми у вигляді
фактів відповідь_так/1 чи відповідь_ні/1 (так само, як
запам'ятовувалися відповіді людини в попередній версії нашої
експертної системи). Це зробить наступний предикат (предикат
запамятати/2 описаний у попередній лабораторній роботі):
додати_ознаки([]).
додати_ознаки([ознака(X, Y)|Signs]):-запамятати(X, Y), додати_
ознаки(Signs).
Після цього слід викликати предикат тварина(X), і на місці
X отримаємо шукану тварину. Предикати вірно/1 та невірно/1
аналогічні таким з попередньої лабораторної роботи крім того,
що тепер не потрібно запитувати людину про наявність тієї чи
іншої властивості у тварини, бо всі відповіді вже внесені до
```

```
бази.
 %для другого типу питань в базі обов'язково повинна бути відповідь щодо поточної ознаки
 вірно(X):-тип_питання(2), відповідь_так(X),!.
 вірно(X):-тип_питання(2), fail. %якщо відповіді немає, то поточна тварина нам не підходить
 %для невірно/1 все навпаки
 невірно(X):-тип_питання(2), відповідь_ні(X),!.
 невірно(X):-тип_питання(2), fail.

 %питання другого типу – яка тварина має дані ознаки
 обробити(речення(група_підмета("Хто"), група_присудка(список_ознак(Signs)))):-
      assert(тип_питання(2)), %для використання в предикатах вірно/невірно
      додати_ознаки(Signs), %додамо ознаки зі списку до бази даних
      (тварина(Animal),!, %визначимо тварину за доданими ознаками
      write("Мабуть, це ",Animal, "."); %тварину знайдено
      write("Я не знаю такої тварини.")),nl,nl. %тварину не знайдено
```

Всі етапи аналізу речення і діалогу з користувачем пов'язуються наступними предикатами:

```
 читати_речення(Str, Sentence):-
  str_toklist(Str, Sent), %розіб'ємо речення на слова
  речення(Sent, Rest, Sentence). %та побудуємо його структуру

 аналіз(Str):-читати_речення(Str, Sent), обробити(Sent),!.
%розбір речення та пошук відповіді на питання
 аналіз(_):-write("Речення не розпізнане!"),nl. %Якщо аналіз речення завершився неуспішно, то потрапимо сюди. Виконання програми продовжується.
 діалог:-
      repeat, %головний цикл
      write("Задайте питання:"),nl,
```



```
      readln(Str), %прочитаємо питання
      (Str="Вихід", !; %завершення програми – вихід з циклу
      аналіз(Str), очистити, %пошук відповіді на питання та очистка бази даних
      fail). %кінець циклу

 очистити:-retract(відповідь_так(_)),fail. %видалимо всі факти відповідь_так
 очистити:-retract(відповідь_ні(_)),fail. %видалимо всі факти відповідь_ні
 очистити:-retract(тип_питання(_)),fail. %видалимо всі факти тип_питання
 очистити. %врешті-решт виклик очистити/0 закінчимо успішно
```

Отже, повний текст програми буде таким:

```
 тварина(гепард):-
 клас(ссавець),
 клас("м'ясоїд"),
 вірно("має рудий колір"),
 вірно("має темні плями"),!.
 тварина(тигр):-
 клас(ссавець),
 клас("м'ясоїд"),
 вірно("має рудий колір"),
 вірно("має темні смути"),!.
 тварина(жираф):-
 клас(копитне),
 вірно("має довгу шию"),
 вірно("має довгі ноги"),
 вірно("має темні плями"),!.
 тварина(зебра):-
 клас(копитне),
 вірно("має темні смути"),!.
 тварина(страус):-
 клас(птах),
 невірно("літає"),
 вірно("має довгу шию"),
 вірно("має довгі ноги"),
```

```
вірно("має чорно-біле забарвлення"),!.
тварина(пінгвін):-
клас(птах),
невірно("літає"),
вірно("плаває"),
вірно("має чорно-біле забарвлення"),!.
тварина(альбатрос):-
клас(птах),вірно("добре літає"),!.
клас(ссавець):-
вірно("має хутро"),!.
клас(ссавець):-
вірно("годує дитинчат молоком"),!.
клас(птах):-
вірно("має пір'я"),!.
клас(птах):-
вірно("літає"),
вірно("кладе яйця"),!.
клас("м'ясоїд"):-
вірно("їсть м'ясо"),!.
клас("м'ясоїд"):-
вірно("має гострі зуби"),
вірно("має пазурі"),
вірно("має спрямовані вперед очі"),!.
клас(копитне):-
клас(ссавець),
вірно("має копита"),!.
клас(копитне):-
клас(ссавець),
вірно("є жвачною"),!.

%якщо розглядається перший тип питань, то просто виведемо
поточну ознаку
вірно(X):-тип_питання(1), write(X, ", ").
невірно(X):-тип_питання(1), write("не ", X, ", ").

%для другого типу питань в базі обов'язково повинна бути
відповідь щодо поточної ознаки
```



```
вірно(X):-тип_питання(2), відповідь_так(X),!.
вірно(X):-тип_питання(2), fail. %якщо відповіді немає, то
поточна тварина нам не підходить
%для невірно/1 все навпаки
невірно(X):-тип_питання(2), відповідь_ні(X),!.
невірно(X):-тип_питання(2), fail.

запамятати(X,так):-assert(відповідь_так(X)).
запамятати(X,ні):-assert(відповідь_ні(X)).

%аналіз речення. Sent – вхідний рядок, Rest – те, що залишилось
від нього після виділення з нього одного речення
речення(Sent, Rest, речення(SubjectGroup, PredicateGroup)):-
    група_підмета(Sent, Rest1, SubjectGroup), %SubjectGroup –
група підмета, Rest1 – вхідний рядок без групи підмета
    група_присудка(Rest1, Rest, PredicateGroup). %тепер з
того, що залишилося від вхідного рядка, виділимо групу присудка

%підметом може бути займенник "Хто"
група_підмета(["Хто"|Rest], Rest, група_підмета("Хто")).
%група підмета також може складатися з займенника та підмета –
іменника, що визначає тварину
група_підмета([Pronoun, Animal|Rest], Rest, група_
підмета(займенник(Pronoun), підмет(Animal))):-тварини(Animal),
займенник(Pronoun).

група_присудка([X|Rest], Rest, група_присудка(список_
ознак([]))):-кінець(X), !. %дійшли до кінця речення
група_присудка(Sent, Rest, група_присудка(список_
ознак(List))):-список_ознак(Sent, Rest, List). %побудуємо
список ознак

список_ознак([X|Sent], Sent, []):-кінець(X). %дійшли до кінця
речення
список_ознак(Sent, Rest, [X|List]):-
    ознака(Sent, Rest1, X, так), %спробуємо визначити ознаку,
вважаючи її позитивною
```

```
    список_ознак(Rest1, Rest, List). %продовжимо розбір речення
%якщо зустріли частку "не", то продовжимо розгляд ознаки, враховуючи її негативність
ознака(["не"|Sent], Rest, ознака(X, Sign), _):-ознака(Sent, Rest, ознака(X, Sign), ні),!.
%якщо після поточного слова стоїть сполучник або кінець речення, то закінчимо рекурсивний розгляд поточної ознаки,
%запишемо останнє слово ознаки та її позитивність/негативність
ознака([X, Y|Sent], [Y|Sent], ознака(X, Sign), Sign):-кінець(Y).
ознака([X, Y|Sent], Sent, ознака(X, Sign), Sign):-сполучник(Y),!.
%розгляд ознаки
ознака([X|Sent], Rest, ознака(Y, Sign), CurrSign):-
    ознака(Sent, Rest, ознака(Y1, Sign), CurrSign), %рекурсивно виберемо слова ознаки до першого сполучника або кінця речення
    concat(X, " ", Str), concat(Str, Y1, Y). %додамо до отриманої ознаки спереду поточне слово

кінець("?"). %кінцівка речення
%сполучники для списку ознак
сполучник(",").
сполучник("і").
сполучник("та").
%займенники
займенник("Який").
займенник("Яка").
%відомі тварини
тварини(гепард).
тварини(тигр).
тварини(жираф).
тварини(зебра).
тварини(страус).
тварини(пінгвін).
тварини(альбатрос).
читати_речення(Str, Sentence):-
```



```
    str_toklist(Str, Sent), %розіб'ємо речення на слова
    речення(Sent, Rest, Sentence). %та побудуємо його структуру

аналіз(Str):-читати_речення(Str, Sent), обробити(Sent),!. %розбір речення та пошук відповіді на питання
аналіз(_):-write("Речення не розпізнане!"),nl. %Якщо аналіз речення завершився неуспішно, то потрапимо сюди. Виконання програми продовжується.

%питання першого типу – які ознаки тварини
обробити(речення(група_підмета(займенник(_), підмет(Animal)), _)):-
    assert(тип_питання(1)), %для використання в предикатах вірно/невірно
    write(Animal, " "), %виведемо назву тварини
    тварина(Animal),nl,nl. %та всі її властивості
%питання другого типу – яка тварина має дані ознаки
обробити(речення(група_підмета("Хто"), група_присудка(список_ознак(Signs)))):-
    assert(тип_питання(2)), %для використання в предикатах вірно/невірно
    додати_ознаки(Signs), %додамо ознаки зі списку до бази даних
    (тварина(Animal),!, %визначимо тварину за доданими ознаками
    write("Мабуть, це ",Animal, "."); %тварину знайдено
    write("Я не знаю такої тварини.")),nl,nl. %тварину не знайдено

додати_ознаки([]).
додати_ознаки([ознака(X, Y)|Signs]):-запамятати(X, Y), додати_ознаки(Signs).

діалог:-
    repeat, %головний цикл
    write("Задайте питання:"),nl,
```

```
      readln(Str), %прочитаємо питання
      (Str="Вихід", !; %завершення програми – вихід з циклу
      аналіз(Str), очистити, %пошук відповіді на питання та
очистка бази даних
      fail). %кінець циклу

очистити:-retract(відповідь_так(_)),fail. %видалимо всі факти
відповідь_так
очистити:-retract(відповідь_ні(_)),fail. %видалимо всі факти
відповідь_ні
очистити:-retract(тип_питання(_)),fail. %видалимо всі факти
тип_питання
очистити. %врешті-решт виклик очистити/0 закінчимо успішно

відповідь_так(_):-fail.
відповідь_ні(_):-fail.
```

Запуск програми здійснюється запитом

```
діалог.
```

Практична частина

Створити експертну систему з інтерфейсом на природній мові.

1. Запустіть інтерпретатор PIE, створіть нову програму.
2. Напишіть текст програми експертної системи (повний текст програми див. в теоретичних відомостях).
3. Збережіть програму.

Перевірити роботу системи синтаксичного аналізу речення.

1. Синтаксичний аналіз речення в щойно створеній програмі виконує предикат читати_речення(Str, Sentence), де Str – вхідний рядок з реченням, Sentence – його деревоподібна структура. Поставте до програми такі запити:

```
читати_речення("Який жираф?", S).
читати_речення("Яка зебра?", S).
читати_речення("Який тигр.", S).
читати_речення("Який тигр", S).
читати_речення("Який літає?", S).
читати_речення("Хто літає?", S).
```



```
читати_речення("Хто має темні смуги, має рудий колір та годує
дитинчат молоком?", S).
читати_речення("Хто не літає і кладе яйця?", S).
```

2. Придумайте ще питання відповідно застосовуваній граматиці та перевірте правильність визначення їх синтаксичних структур.

Випробувати роботу інтерфейсу експертної системи на природній мові.

1. Запустіть програму запитом

```
діалог.
```

2. У відповідь на запрошення системи задайте, наприклад, такі питання (кожне питання завершуйте клавішею Enter):

```
Який жираф?
Яка зебра?
Який тигр.
```

3. Перевірте так само кожну відому системі тварину.
4. Далі спробуйте задати такі питання:

```
Хто має темні смуги, має хутро і є жвачною?
Хто не має хутро, літає, кладе яйця, добре літає?
Хто не літає?
```

5. Придумайте ще питання, аналогічні наведеним і задайте їх програмі.

Зауваження. Кожне питання обов'язково повинне закінчуватися знаком питання. Ознаки тварини слід писати точно так само, як вони написані в базі знань. Слід вказувати обов'язково всі характеристики тварини, які необхідні для її ідентифікації. Наша програма не здійснює морфологічний аналіз, тому іноді текст питання для неї не відповідатиме вимогам чистої української мови. Наприклад, для нашої програми слід писати «Хто не має хутро, ...» замість «Хто не має хутра, ...».

6. Для виходу з програми напишіть

```
Вихід
```

Завдання до лабораторної роботи 7

Реалізуйте аналогічний розглянутому інтерфейс на природній мові для вашої експертної системи (див. завдання до попередньої лабораторної роботи).

Контрольні запитання

1. Де застосовуються системи аналізу природної мови?

2. Які рівні аналізу природної мови вам відомі?
3. На які етапи розділяється аналіз мови?
4. Опишіть граматику, яка застосовується в нашій програмі.
5. Як відбувається виділення групі підмета і групи присудка з речення?
6. Як будується список ознак?
7. Як система знаходить відповідь на питання першого типу?
8. Як система знаходить відповідь на питання другого типу?
9. Яким вимогам повинні відповідати питання до нашої експертної системи?

3.6.3. Лабораторна робота 8. Компілятор Visual Prolog

Мета: вивчити основні особливості написання програм для Visual Prolog, навчитися перекладати програми, написані в класичному стилі, на Visual Prolog, сформувати навички виконання базових операцій в інтегрованому середовищі програмування Visual Prolog.

Теоретичні відомості

Досі вивчали класичний Пролог за допомогою інтерпретатора цієї мови. Проте інтерпретатор має низьку швидкість роботи, не дозволяє створювати програми, які можна запускати на інших комп'ютерах і мають розвинений інтерфейс користувача. Тому інтерпретатор виявляється придатним лише для навчальних цілей. Серед реалізацій Пролога, придатних для програмування систем промислового рівня, найрозвиненішою є Visual Prolog компанії PDC (Prolog Development Center). Ця система є прямим нащадком найпопулярнішої в свій час в СРСР версії Turbo Prolog. Її особливостями є:

– поєднання Пролога з об'єктно-орієнтованим програмуванням;
– жорстка типізація;
– зручне візуальне інтегроване середовище програмування;
– один з найефективніших компіляторів;
– наявність безкоштовної версії з обмеженими можливостями;
– робота в середовищі Windows;
– наявність великої кількості бібліотек PFC (Prolog Foundation Classes) для реалізації сучасних технологій – робота з COM-об'єктами,

створення графічного інтерфейсу користувача, робота з базами даних, HTML, CGI, та ін.
– підтримка багатозадачності;
– підтримка Unicode;
– автоматичне керування пам'яттю.

Через необхідність підвищення швидкості компіляції та виконання програм і застосування сучасних технологій (об'єктно-орієнтоване програмування) мова, реалізована у Visual Prolog досить сильно відрізняється від класичного Пролога.

Об'єктна орієнтованість Visual Prolog виражається у використанні *класів* і *об'єктів*. В класі описуються предикати, що реалізують логіку поведінки об'єкта. Кожна програма має принаймні один клас. При створенні нового проекту Visual Prolog автоматично створює базову структуру основного класу програми, тому зосередимось на реалізації внутрішньої логіки цього класу.

Всі внутрішні предикати, опис типів та константи класу розміщуються між ключовими словами implement та end implement (після слова implement вказується ім'я класу). Весь код програми всередині класу розділений на *секції*. Кожна секція починається з відповідного ключового слова. Секції можуть розташовуватися в довільному порядку, в програмі може бути декілька секцій одного типу.

Існують такі види секцій:

1. `constants`

Ця секція використовується для визначення декотрих часто вживаних значень:

```
constants
    програма = "Visual Prolog 7.0".
    пі = 3.14.
```

2. `domains`

В цій секції описуються типи даних (домени), які використовуються в програмі.

Системі вже відомі декотрі стандартні типи даних. Ось основні з них:

`integer` – ціле число;

`real` – дійсне число;

`char` – один символ (значення цього типу записується в одинарних лапках: `'a'`);

string – рядок символів (значення записується в подвійних лапках);

boolean – може приймати значення true() або false().

На основі стандартних типів можна визначати складніші типи даних. Способи визначення нових типів зрозумілі з наступних прикладів:

```
domains
    my_int = integer. %my_int - тип цілого числа
    my_str = string. %my_str - тип рядку символів
    int_list = integer*. %визначає новий тип int_list - список цілих чисел
    my_str_list = my_str*. %список рядків символів
    дата_стр = дата(my_int, my_str, my_int). %тип - структура з функтором дата. Перший аргумент структури - ціле число, другий - рядок, третій - ціле число
    тип_трикутник = трикутник(тип_точка, тип_точка, тип_точка). %структура, що описує трикутник
    тип_точка = точка2(integer, integer); %точка з цілими координатами на площині
        точка3(integer, integer, integer). %або точка в просторі
```

тип_точка – складений тип, дані цього типу можуть бути структурами точка2 або структурами точка3.

Ось кілька прикладів об'єктів цих типів даних

[3, 2, 1] – тип int_list;

дата(1, "квітень", 2007) – тип дата_стр;

точка2(1, 1) – тип тип_точка;

точка3(3, 3, 3) – тип тип_точка;

трикутник(точка2(0, 1), точка2(1, 0), точка2(1, 1)) – тип тип_трикутник;

трикутник(точка3(0, 1, 2), точка3(1, 0, -1), точка2(1, 1, 5)) – тип тип_трикутник.

Без особливих хитрувань неможливо створити тип для списку, елементи якого різнотипні або структури, яка може мати різну кількість аргументів.

3. class facts

В цій секції оголошуються факти, які можуть динамічно змінюватися в процесі виконання програми. Формат секції:

```
class facts
    <ім'я_факту> : (<тип_аргументу>, ...) <вид_факту>.
    ...
```

Вид факту може бути одним з наступних:

– nondeterm – недетермінований факт, може мати будь-яку кількість значень;

– determ – детермінований факт, може мати одне значення або не мати значень. Якщо факт не має значення, то його виклик завжди закінчується неуспішно;

– single – факт завжди має єдине значення, предикат assert/1 перевизначає це значення, не додаючи нових фактів. Такі факти неможна вилучати предикатом retract/1.

Якщо вид факту не вказаний, то він вважається nondeterm фактом.

Приклад:

```
class facts
    людина:(string). %факт з аргументом рядкового типу
    тварина:(string).
    кінець:() determ. %факт без аргументів
```

Факти, описані в секції class facts, можна в ході виконання програми можна додавати і вилучати з бази даних за допомогою розглянутих раніше предикатів assert/1 і retract/1. Початкові набори фактів можна задавати в секції clauses (див. далі). Для фактів виду single одне таке значення обов'язково повинне бути вказано.

4. class predicates

В цій секції оголошуються предикати програми. Формат секції:

```
class predicates
    <ім'я_предиката>:(<тип_аргументу>, ...) <вид_предиката> <шаблон_аргументів>.
    ...
```

Кожний предикат, що використовується в програмі, повинен бути оголошений в цій секції. Вид предикату може приймати такі значення:

– nondeterm – недетермінований предикат, може завершуватися успішно або неуспішно, завдяки механізму повернень може повертати декілька розв'язків;

– `determ` – детермінований предикат, може завершуватися успішно або неуспішно та завжди породжує єдиний розв'язок;

– `multi` – недетермінований предикат, який може породжувати багато розв'язків, але завжди повинен бути успішним;

– `procedure` – детермінований предикат, який завжди повинен повертати істину;

– `failure` – детермінований предикат, що завжди завершується неуспішно.

Якщо вид не вказано, то предикату надається вид procedure.

Шаблон аргументів (flow pattern) визначає, які з аргументів предикату можуть бути вхідними, а які – вихідними. При зверненні до предикату на місці вхідних аргументів повинні стояти конкретні об'єкти або конкретизовані змінні, а на місці вихідних – неконкретизовані змінні. В Visual Prolog 7.0 неможливо викликати предикат з аргументом, якому не передається ніякого значення і який не набуває значення в результаті виконання предикату.

Шаблон аргументів має такий вигляд:

```
(<вид_аргументу>, ...),
```

де вид аргументу приймає значення і для відповідних вхідних аргументів та `o` – для вихідних.

Предикат може мати декілька шаблонів аргументів, перед кожним шаблоном повинен стояти відповідний йому вид предикату. Якщо шаблон не вказано, то всі аргументи вважаються вхідними. Щоб не перераховувати всі можливі шаблони для кожного предикату, можна замість шаблону вказати ключове слово anyflow – предикат може застосовуватись з будь-яким набором вхідних і вихідних аргументів.

Приклад:

```
class predicates
    батько1:(string, string) nondeterm anyflow. %повертає всі можливі розв'язки для будь-якої комбінації вхідних і вихідних параметрів
    батько2:(string, string) determ (i,o). %дозволяє знаходити одну дитину даної людини
    рівнобічний:(тип_трикутник) nondeterm (i).
```

Якщо можна пожертвувати ефективністю програми, то всі предикати можна оголосити як `nondeterm anyflow`, тоді їх поведінка в цілому відповідатиме класичному Прологу.

5. `clauses`

В цій секції містяться речення, що власне визначають предикати, описані в попередній секції. Всі застосовані в цій секції предикати (крім стандартних) повинні бути описані в розділі опису предикатів. Речення, що визначають один предикат (речення з однаковим заголовком) повинні міститися в цій секції одне за одним.

6. `goal`

Ця секція міститься поза межами класу та визначає *головну точку входу* до програми. В ній розміщується *внутрішня ціль* програми – предикат, з якого починається її виконання. Внутрішня ціль повинна мати вид procedure, тобто завжди бути успішною.

Об'єктна орієнтованість спричинила деякі відмінності у використанні стандартних предикатів – тепер більшість з них належать до деяких класів. Практично це означає, що, наприклад, замість write(X) потрібно писати `console::write(X)`. Крім того, деякі предикати можуть повертати значення на зразок функцій в інших мовах програмування (наприклад, замість `readln(X)` потрібно писати `X=console::readLine()`). Нам знадобиться таблиця відповідності деяких найуживаніших стандартних предикатів.

Таблиця 7

Таблиця відповідності деяких найуживаніших стандартних предикатів

PIE	Visual Prolog	Призначення
`fail/0`	`fail/0`	неуспіх
`true/0`	`succeed/0`	істина
`not`	`not`	заперечення
`write/?`	`console::write/?`	вивід на екран
`nl/0`	`console::nl/0`	новий рядок
`readln/1`	`console::readLine/0 ->string`	Читає рядок символів з клавіатури. Використовується як функція, повертає значення типу `string`
`assert/1`	`assert/1`	додавання фактів в програму
`retract/1`	`retract/1`	вилучення фактів з програми

Закінчення таблиці 7

PIE	Visual Prolog	Призначення
repeat/0	std::repeat/0	повторення
concat/3	string::concat/2 ->string	об'єднує два рядки в третій
front-token/3	string::frontToken/3	виділяє з рядку (перший аргумент) перше слово чи розділовий знак (другий аргумент) та повертає залишок рядку (третій аргумент)

В Visual Prolog немає оператора присвоювання is. Замість нього слід використовувати звичайну уніфікацію =.

Отже, для того, щоб програму на класичному Пролозі (які ми створювали досі) перетворити на програму для Visual Prolog, потрібно:

1. Описати всі типи аргументів предикатів у розділі domains.
2. Описати всі предикати програми у розділі class predicates.
3. Записати весь текст програми у розділі clauses.
4. Записати запит до програми в розділі goal.
5. Замінити всі стандартні предикати на їх відповідники у Visual Prolog.

Розглянемо адаптацію нашої експертної системи.

Спочатку визначимо набір типів:

```
domains
    сп_речення = string*. %речення у вигляді списку слів
    сп_синтаксична_одиниця = синтаксична_одиниця*. %список синтаксичних одиниць
    синтаксична_одиниця = %синтаксична одиниця – це...
    речення(синтаксична_одиниця, синтаксична_одиниця); /* структура "речення", що складається з двох синтаксичних одиниць – групи підмета і групи присудка*/
    група_підмета(синтаксична_одиниця, синтаксична_одиниця); %або структура "група_підмета"
    підмет(string); %або найпростіші синтаксичні одиниці
    займенник(string);
    група_присудка(синтаксична_одиниця); %або група_присудка, що складається з списку ознак
```



```
    список_ознак(сп_синтаксична_одиниця); %або список_ознак, який складається зі списку синтаксичних одиниць, ...
    ознака(string, string); %кожна з яких є ознакою
    нічого(). %або порожня структура "нічого"
```

Речення, з якими працює програма, мають складні і різноманітні деревоподібні структури. Тому тип структури речення неможливо визначити простим описом типів її компонентів. Ми визначили універсальний тип синтаксична_одиниця для всіх можливих структурних елементів речення. Це дозволить обробляти речення різноманітних структур, користуючись лише цим типом даних.

В початковій програмі структура група_підмета могла мати або єдиний компонент «Хто», або два компоненти – займенник і підмет. В Visual Prolog неможливо описати тип для структури з різною кількістю компонентів, тому нам довелося трохи змінити представлення структури речення для другого типу питань і ввести додаткову синтаксичну одиницю – порожню структуру нічого(). Відповідні зміни було внесено до предикатів основної програми.

Факти, що змінюються динамічно, визначаються просто:

```
class facts
    відповідь_так:(string) nondeterm.
    відповідь_ні:(string) nondeterm.
    тип_питання:(integer) single. %тип питання завжди має єдине значення
Опишемо предикати програми:
class predicates
    клас:(string) nondeterm anyflow.
    тварина:(string) nondeterm anyflow.

    вірно:(string) nondeterm anyflow.
    невірно:(string) nondeterm anyflow.
    запамятати:(string, string) nondeterm anyflow.

    пр_речення:(сп_речення, сп_речення, синтаксична_одиниця) nondeterm anyflow.
    пр_група_підмета:(сп_речення, сп_речення, синтаксична_одиниця) nondeterm anyflow.
```

```
    пр_група_присудка:(сп_речення, сп_речення, синтаксична_
одиниця) nondeterm anyflow.
    пр_список_ознак:(сп_речення, сп_речення, сп_синтаксична_
одиниця) nondeterm anyflow.
    пр_ознака:(сп_речення, сп_речення, синтаксична_одиниця,
string) nondeterm anyflow.

    кінець:(string) nondeterm anyflow.
    сполучник:(string) nondeterm anyflow.
    пр_займенник:(string) nondeterm anyflow.
    тварини:(string) nondeterm anyflow.

    читати_речення:(string, синтаксична_одиниця) nondeterm
anyflow.
    аналіз:(string) nondeterm anyflow.
    обробити:(синтаксична_одиниця) nondeterm anyflow.
    додати_ознаки:(сп_синтаксична_одиниця) nondeterm anyflow.

    очистити:() nondeterm anyflow.
```

Було змінено назви декотрих предикатів, щоб вони не збігалися з назвами типів. Для простоти всі предикати оголошені як `nondeterm anyflow`.

В основну програму необхідно внести такі зміни:

1. Замінити всі атоми на рядки – відповідно до типів аргументів предикатів (заміть `гепард - "гепард"`).
2. Змінити порядок предикатів так, щоб однотипні предикати розташувались один за одним (згрупувати предикати `вірно/1` та `невірно/1`).
3. Змінити назви предикатів (див. опис предикатів).
4. Змінити предикати `пр_група_підмета/2` та `обробити/1` відповідно до нової структури речення:

```
пр_група_підмета(["Хто"|Rest], Rest, група_підмета(займенник
("Хто"), нічого())). %нова структура
обробити(речення(група_підмета(займенник("Хто"), _), група_
присудка(список_ознак(Signs)))):-
    assert(тип_питання(2)), додати_ознаки(Signs),
    (тварина(Animal),!, console::write("Мабуть, це ",Animal,
".");
```



```
    console::write("Я не знаю такої тварини.")),console::nl,
console::nl.
```

5. З предикату `очистити/0` прибрати видалення фактів `тип_питання/1`, оскільки факти виду `single` непотрібно видаляти.
6. Додати початкове значення для факту `тип_питання/1`:

```
тип_питання(1).
```

7. Замінити всі вбудовані предикати їх відповідниками для Visual Prolog. Предикат `str_toklist/2` потрібно визначити окремо:

```
class predicates
    str_toklist:(string, core::string_list) procedure (i,o).
clauses
    str_toklist(STRING, [H|TAIL]) :-
    string::frontToken(STRING, H, REST), !,
    str_toklist(REST, TAIL).
    str_toklist(STRING, [STRING]).
```

8. Записати предикат `run/0` – внутрішню мету програми:

```
run():-
    std::repeat, %головний цикл
    console::write("Задайте питання:"),console::nl,
    Str=console::readLine(), %прочитаємо питання
    (Str="Вихід", !; %завершення програми - вихід з циклу
    аналіз(Str), очистити, %пошук відповіді на питання та
очистка бази даних
    fail). %кінець циклу
run(). %забезпечимо успішне завершення
```

Останній рядок є необхідним тому що `run/0` повинен мати вид procedure і завжди завершуватися успішно.

Практична частина

Адаптувати експертну систему до Visual Prolog та скомпілювати її.

1. Запустіть середовище розробки Visual Prolog та створіть новий проект командою Project►New….
2. У вікні Project Settings введіть назву проекту (наприклад, expert) в полі Project Name, в полі Base Directory вкажіть каталог, в якому буде збережено проект. В списку UI Strategy виберіть Console,

оскільки ми будемо створювати консольну програму без графічного інтерфейсу. Всі останні опції залиште без змін та натисніть кнопку OK (див. рис. 18).

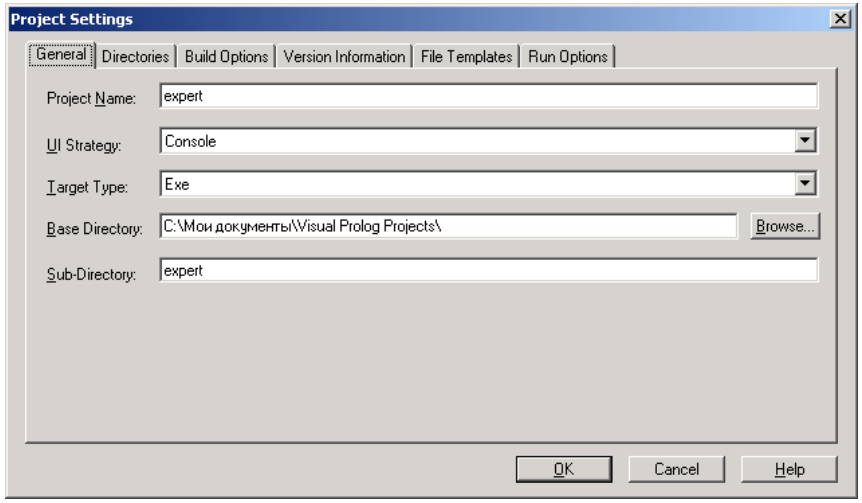


Рис. 18. Вікно Project Settings

3. Visual Prolog створить новий проект та відкриє поки що майже порожнє *дерево проекту*.

4. Запустіть програму командою Build►Execute. Система створить всі необхідні файли проекту, скомпілює програму та запустить її. Проект поки що порожній, тому програма лише створює консольне вікно та одразу завершує роботу.

5. У вікні з деревом проекту з'являться файли зі структурою найпростішої програми. Кожний файл можна відкривати прямо з дерева за допомогою миші.

6. Продивіться вміст файлу expert.cl. Цей файл містить опис класу expert, який буде основним класом нашої програми (назви файлу і класу можуть бути іншими в залежності від того, як ви назвали проект).

7. Файл expert.pro містить реалізацію класу `expert` – опис його внутрішніх предикатів, типів, фактів і констант:


```
implement expert
    open core
constants
    className = "com/visual-prolog/expert".
    classVersion = "$JustDate: $$Revision: $".

clauses
    classInfo(className, classVersion).

clauses
    run():-
    console::init(),
    succeed(). % place your own code here
end implement expert

goal
    mainExe::run(expert::run).
```

Ми бачимо опис двох констант класу і предикату `classInfo`. Внутрішньою метою є виклик предикату run класу `mainExe`. Цей предикат після виконання ініціалізації програми передасть керування предикату run класу `expert`, який і керує виконанням програми. В порожньому проекті цей предикат лише ініціалізує вікно виводу та успішно завершує виконання програми. Сам предикат `expert::run` оголошено в описі класу (файл expert.cl). Ми не будемо створювати інших класів (крім основного), тому всю програму запишемо у файлі expert.pro. Отже, нам потрібно вписати предикати нашої експертної системи у створену системою структуру класу expert.

8. Якщо в дереві проекту виділити файл, то в правій частині вікна дерева з'явиться структура класів, предикатів, фактів, констант і типів даних, описаних у даному файлі, рис. 19. Цю структуру зручно використовувати для швидкого переходу до потрібних об'єктів програми.

9. Відкрийте за допомогою дерева проекту файл expert.pro, видаліть з нього стандартну реалізацію предиката `run`.

10. Всередині класу `expert` (між рядками `implement expert` та `end implement expert`) вставте секції опису типів (секція domains), опису предикатів (секція `class predicates`), фактів (секція `class facts`) та опис предикату `str_toklist` (текст всіх цих секцій наведений в теоретичних відомостях).

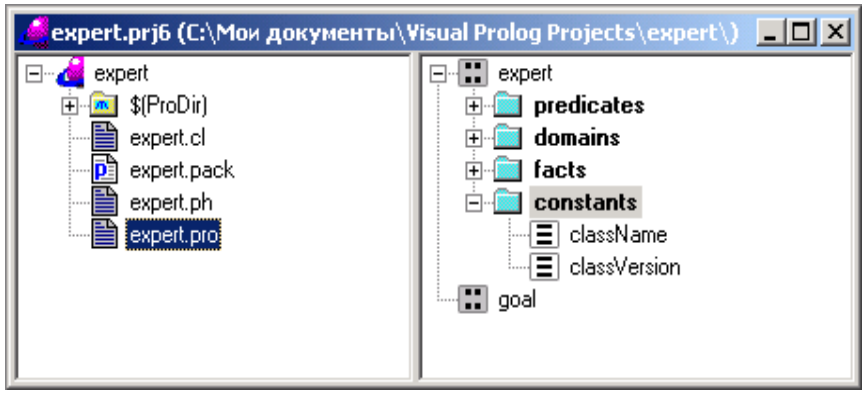


Рис. 19. Дерево проектів

11. Додайте секцію `clauses` із зміненими предикатами програми (див. теоретичні відомості):

```
clauses

тварина("гепард"):-
клас("ссавець"),
клас("м'ясоїд"),
вірно("має рудий колір"),
вірно("має темні плями"),!.

тварина("тигр"):-
клас("ссавець"),
клас("м'ясоїд"),
вірно("має рудий колір"),
вірно("має темні смути"),!.

тварина("жираф"):-
клас("копитне"),
вірно("має довгу шию"),
вірно("має довгі ноги"),
вірно("має темні плями"),!.
```



```
тварина("зебра"):-
клас("копитне"),
вірно("має темні смути"),!.

тварина("страус"):-
клас("птах"),
невірно("літає"),
вірно("має довгу шию"),
вірно("має довгі ноги"),
вірно("має чорно-біле забарвлення"),!.

тварина("пінгвін"):-
клас("птах"),
невірно("літає"),
вірно("плаває"),
вірно("має чорно-біле забарвлення"),!.

тварина("альбатрос"):-
клас("птах"),вірно("добре літає"),!.

клас("ссавець"):-
вірно("має хутро"),!.

клас("ссавець"):-
вірно("годує дитинчат молоком"),!.

клас("птах"):-
вірно("має пір'я"),!.

клас("птах"):-
вірно("літає"),
вірно("кладе яйця"),!.

клас("м'ясоїд"):-
вірно("їсть м'ясо"),!.
```

```
клас("м'ясоїд"):-
вірно("має гострі зуби"),
вірно("має пазурі"),
вірно("має спрямовані вперед очі"),!.

клас("копитне"):-
клас("ссавець"),
вірно("має копита"),!.

клас("копитне"):-
клас("ссавець"),
вірно("є жвачною"),!.

вірно(X):-тип_питання(1), console::write(X, ", ").
вірно(X):-тип_питання(2), відповідь_так(X),!.
вірно(X):-тип_питання(2), fail.

невірно(X):-тип_питання(1), console::write("не ", X, ", ").
невірно(X):-тип_питання(2), відповідь_ні(X),!.
невірно(X):-тип_питання(2), fail.

запамятати(X,"так"):-assert(відповідь_так(X)).
запамятати(X,"ні"):-assert(відповідь_ні(X)).

%аналіз речення. Sent – вхідний рядок, Rest – те, що залишилось
від нього після виділення з нього одного речення
пр_речення(Sent, Rest, речення(SubjectGroup, PredicateGroup)):-
    пр_група_підмета(Sent, Rest1, SubjectGroup), %SubjectGroup
– група підмета, Rest1 – вхідний рядок без групи підмета
    пр_група_присудка(Rest1, Rest, PredicateGroup). %тепер з
того, що залишилося від вхідного рядка, виділимо групу присудка

%підметом може бути займенник "Хто"
пр_група_підмета(["Хто"|Rest], Rest, група_підмета
(займенник("Хто"), нічого())).
%група підмета також може складатися з займенника та підмета –
іменника, що визначає тварину
```



```
пр_група_підмета([Pronoun, Animal|Rest], Rest, група_
підмета(займенник(Pronoun), підмет(Animal))):-тварини(Animal),
пр_займенник(Pronoun).

пр_група_присудка([X|Rest], Rest, група_присудка(список_
ознак([]))):-кінець(X), !. %дійшли до кінця речення
пр_група_присудка(Sent, Rest, група_присудка(список_
ознак(List))):-пр_список_ознак(Sent, Rest, List). %побудуємо
список ознак

пр_список_ознак([X|Sent], Sent, []):-кінець(X). %дійшли до
кінця речення
пр_список_ознак(Sent, Rest, [X|List]):-
    пр_ознака(Sent, Rest1, X, "так"), %спробуємо визначити
ознаку, вважаючи її позитивною
    пр_список_ознак(Rest1, Rest, List). %продовжимо розбір
речення
%якщо зустріли частку "не", то продовжимо розгляд ознаки,
враховуючи її негативність
пр_ознака(["не"|Sent], Rest, ознака(X, Sign), _):-пр_
ознака(Sent, Rest, ознака(X, Sign), "ні"),!.
%якщо після поточного слова стоїть сполучник або кінець речення,
то закінчимо рекурсивний розгляд поточної ознаки,
%запишемо останнє слово ознаки та її позитивність/негативність
пр_ознака([X, Y|Sent], [Y|Sent], ознака(X, Sign), Sign):-
кінець(Y).
пр_ознака([X, Y|Sent], Sent, ознака(X, Sign), Sign):-
сполучник(Y),!.
%розгляд ознаки
пр_ознака([X|Sent], Rest, ознака(Y, Sign), CurrSign):-
    пр_ознака(Sent, Rest, ознака(Y1, Sign), CurrSign),
%рекурсивно виберемо слова ознаки до першого сполучника або
кінця речення
    Str = string::concat(X, " "), Y = string::concat(Str,
Y1). %додамо до отриманої ознаки спереду поточне слово
```

```
кінець("?"). %кінцівка речення
%сполучники для списку ознак
сполучник(",").
сполучник("і").
сполучник("та").
%займенники
пр_займенник("Який").
пр_займенник("Яка").
%відомі тварини
тварини("гепард").
тварини("тигр").
тварини("жираф").
тварини("зебра").
тварини("страус").
тварини("пінгвін").
тварини("альбатрос").

читати_речення(Str, Sentence):-
     str_toklist(Str, Sent), %розіб'ємо речення на слова
     пр_речення(Sent, Rest, Sentence). %та побудуємо його
структуру

аналіз(Str):-читати_речення(Str,  Sent),  обробити(Sent),!.
%розбір речення та пошук відповіді на питання
аналіз(_):-console::write("Речення                    не
розпізнане!"),console::nl. %Якщо аналіз речення завершився
неуспішно, то потрапимо сюди. Виконання програми продовжується.

%питання першого типу - які ознаки тварини
обробити(речення(група_підмета(займенник(_), підмет(Animal)),
_)):-
     assert(тип_питання(1)), %для використання в предикатах
вірно/невірно
     console::write(Animal, " "), %виведемо назву тварини
     тварина(Animal),console::nl,console::nl.  %та  всі  її
властивості
```



```
%питання другого типу - яка тварина має дані ознаки
обробити(речення(група_підмета(займенник("Хто"),  _),  група_
присудка(список_ознак(Signs)))):-
     assert(тип_питання(2)), %для використання в предикатах
вірно/невірно
     додати_ознаки(Signs), %додамо ознаки зі списку до бази
даних
     (тварина(Animal),!,  %визначимо  тварину  за  доданими
ознаками
     console::write("Мабуть, це ",Animal, "."); %тварину знайдено
     console::write("Я не знаю такої тварини.")),console::nl,
console::nl. %тварину не знайдено

додати_ознаки([]).
додати_ознаки([ознака(X, Y)|Signs]):-запамятати(X, Y), додати_
ознаки(Signs).

очистити():-retract(відповідь_так(_)),fail.  %видалимо  всі
факти відповідь_так
очистити():-retract(відповідь_ні(_)),fail. %видалимо всі факти
відповідь_ні
очистити(). %врешті-решт виклик очистити/0 закінчимо успішно

тип_питання(1). %початкове значення
```

12. Нарешті, в секцію clauses додайте реалізацію предикату run, описану в теоретичних відомостях.

Зауваження. Не змінюйте та не видаляйте константи і предикати (окрім run), створені середовищем розробки при створенні проекту.

13. Збережіть проект (Project ►Save), відкомпілюйте і запустіть його (Build ►Execute). В підкаталозі Exe\ каталогу проекту повинен з'явитися файл expert.exe, який можна запускати безпосередньо.

14. Якщо програма не скомпілювалася, то перевірте помилки, що повинні з'явитися у вікні Errors/Warnings. Перевірте, чи правильно описані та оголошені всі предикати, факти і типи даних, чи всі

речення входять до відповідних секцій, чи всі нові секції містяться в описі класу.

15. Випробуйте роботу експертної системи з інтерфейсом на природній мові, задаючи їй питання так само, як і на попередній лабораторній роботі.

Завдання до лабораторної роботи 8

Адаптуйте до Visual Prolog та скомпілюйте експертну систему, написану вами при виконанні завдань до двох попередніх лабораторних робіт.

Контрольні запитання

1. Чим вирізняється Visual Prolog серед інших Пролог-компіляторів?
2. Чим відрізняється мова Visual Prolog від мови інтерпретатора PIE?
3. Опишіть структуру програми на Visual Prolog.
4. Для чого застосовується і який формат має секція constants?
5. Які стандартні типи даних вам відомі?
6. Як визначаються нові типи даних?
7. Як оголошуються факти, які можна динамічно змінювати під час виконання програми?
8. Які види фактів вам відомі? Опишіть особливості кожного виду.
9. Як оголошуються предикати програми?
10. Які види предикатів вам відомі? Опишіть особливості кожного виду.
11. Для чого застосовується секція clauses?
12. Що таке внутрішня ціль програми? Як вона визначається?
13. Опишіть процес адаптації програми на класичному Пролозі до Visual Prolog.

Література до розділу

1. Експертні системи: навч. посібник / сост. А. Н. Никулин. – Ульяновськ: УлГТУ, 2015. – 78 с. (рос.)

2. Джексон Пітер. Вступ до експертних систем / Джексон Пітер – СПб.: Видавництво дім «Вільямс», 2012. (рос.)

3. Нейлор К. Як побудувати свою експертну систему / К. Нейлор, – М.: Энергоатомиздат, 2012. – 286 с. (рос.)

4. Інтелектуальні інформаційні системи. PROLOG – мова розробки інтелектуальних та експерних систем: навч.посібник / С. П. Хабаров. – СПб. СПбГЛТУ, 2013. – 138 с (рос.)

5. Представлення знань в інформаційних системах. Використання середовища PIE при проектування баз даних та знань: навч. посібник / С. П. Хабаров, Л. Г. Пушкарева; / під ред. А. М. Заяц. – Санкт-Петербург: СПбГЛТУ, 2019. – 66 с. (рос.)

РОЗДІЛ IV. НЕЙРОННІ МЕРЕЖІ

4.1. Основні елементи нейромереж

Так як основне завдання штучного інтелекту – моделювання міркувань, а природне «пристрій, здатний думати» – це мозок, то очевидною завданням є створення «штучного мозку» «за образом та подобою» людського. Дослідженням цього питання стали займатися в рамках напрямку штучного інтелекту «нейрокібернетика».

Основною ідеєю нейрокібернетики стало відтворення «в залізі» клітини мозку – нейрони.

Нейрони – спеціалізовані клітини, здатні приймати, обробляти, кодувати, передавати і зберігати інформацію, організовувати реакції на подразнення, встановлювати контакти з іншими нейронами, клітинами органів. Унікальною особливістю нейрона є здатність генерувати електричні розряди і передавати інформацію за допомогою спеціалізованих закінчень – синапсів.

Число нейронів мозку людини наближається до 1011. На одному нейроні може бути до 10 000 синапсів. Якщо тільки ці елементи вважати осередками зберігання інформації, то можна прийти до висновку, що нервова система може зберігати 1019 од. інформації, тобто здатна вмістити практично всі знання, накопичені людством.

1943 рік став роком народження теорії штучних нейронних мереж. Дж. Маккалок і У. Пітт запропонували модель формального нейрона (рис. 20) та описали основні принципи побудови нейронних мереж.

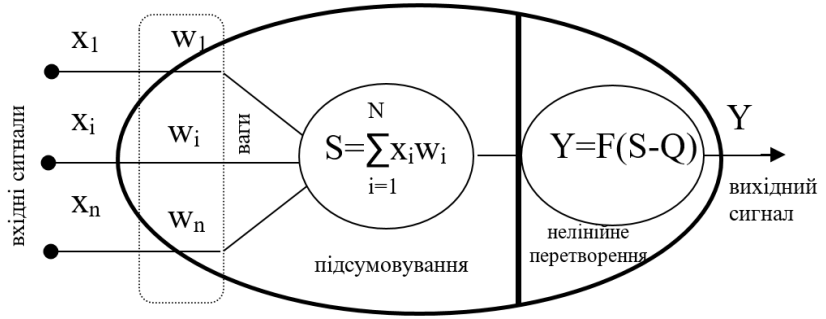


Рис. 20. Модель формального нейрону

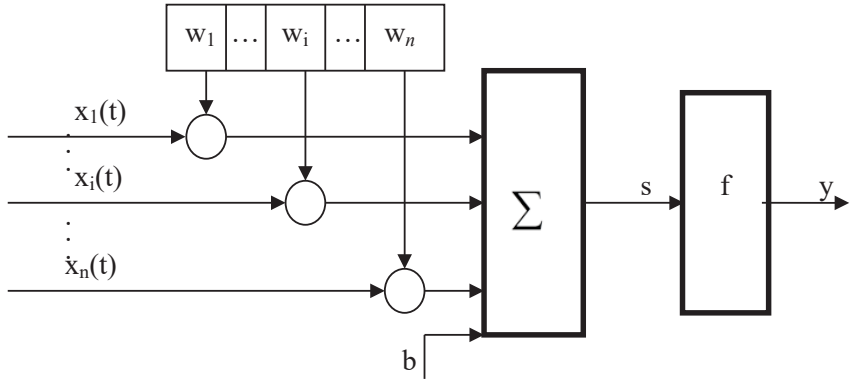


Рис. 21. Структура штучного нейрона з зсувом

Синапси нейрона – місця контактів нервових волокон – відповідно до цієї моделі, передають сигнали, визначаючи силу впливу сигналу з цього входу на вихідний сигнал, що надходить на аксон. Для цього кожному входу ставиться у відповідність ваговий коефіцієнт w_i. Дендрити отримують вхідний сигнал, представлений вектором x_i. Потім нейрон обробляє сигнал, що надійшов, виробляючи зважене підсумовування і нелінійне перетворення, використовуючи для цього активаційну функцію (функції активації можуть бути різними, найбільш поширені – лінійна, порогова і сігмоїдна), аргументом якої буде результат підсумовування мінус граничне значення. Це значення визначає рівень сигналу, на який нейрон буде реагувати.

Математична модель нейрона описується таким співвідношенням:

$$s = \sum_{i=1}^{n} x_i \cdot w_i + b\,,\ y = f(s)\,, \qquad (1)$$

де b – значення зсуву.

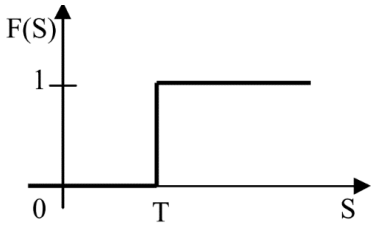
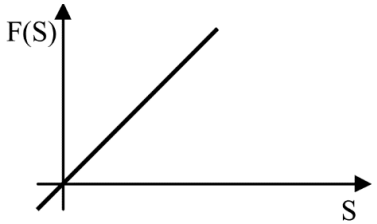
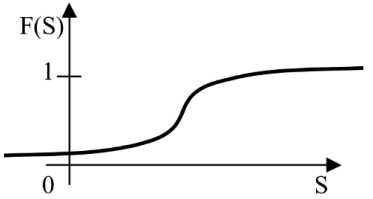
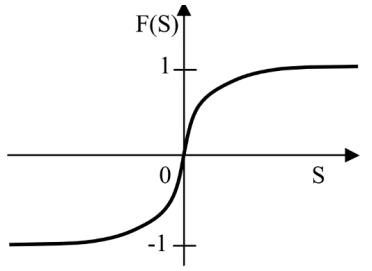
Синаптичні зв'язки з додатними вагами називають *збуджувальними*, з від'ємними вагами – *гальмувальними*.

Функції активації можуть бути різними, найбільш часто використовувані представлені в таб. 8.

Головними властивостями біологічного нейрона є його здатність до навчання, універсальність, здатність вирішувати різні завдання. Описана вище модель не здатна до цього. Навченою вона стала лише

Таблиця 8

Основні функції активації

Назва	Формула	Графік (приклад)
Порогова	$FS = \begin{cases} 1, & при\ S \geq T, \\ 0, & при\ S < T, \end{cases}$ $T = const.$	
Лінійна	$F(S) = k \cdot S,$ $k = const.$	
Сигмоїдальна	$F(S) = \frac{1}{1 + e^{-S \cdot k}},$ $k = const.$	
Гіперболічний тангенс	$F(S) = \frac{e^{\frac{S}{k}} - e^{-\frac{S}{k}}}{e^{\frac{S}{k}} + e^{-\frac{S}{k}}},$ $k = const.$	

в 1949 році завдяки Д. Хеббу (D. Hebb), який, спираючись на фізіологічні та психологічні дослідження, висунув гіпотезу про навчання біологічних нейронів. Його метод навчання став відправною точкою для алгоритмів навчання нейронних мереж без вчителя.

У 1957 році в світовому науковому світі сталася друга значима в історії нейронних мереж подія: американський фізіолог Ф. Розенблатт розробив модель зорового поширення та розпізнавання – персептрон (percеptron), а потім і побудував перший нейрокомп'ютер «Марк-1».

Нейронні мережі були прості, зрозумілі і багатообіцяючі. Складні процеси мислення, здавалося, були готові розкритися перед людиною, для їх опису були потрібні лише елементарні математичні операції: додавання, множення і лінійна функція. На персептрон, а потім і багатошаровий персептрон покладалися великі надії, тому бурхливий ентузіазм, з яким він був прийнятий, досить швидко змінився жорсткою критикою.

У 1969 році вийшла в світ книга «Персептрони» М. Мінського та С. Паперті, яка ознаменувала закінчення першого етапу в історії нейронних мереж. У цій книзі було проведено аналіз можливостей одношарових нейронних мереж та їх обмежень.

Обмеження, виявлені Мінським, були характерні одношаровим нейронним мережам, але не багатошаровим нейронним мережам, яким була властива інша проблема – їх навчання.

Запропонований в 1986 році Д. Румельхардом та іншими вченими алгоритм зворотного поширення помилки став одним з провідних чинників, що породив сучасний нейромережевий бум, так як був ефективним способом навчання нейронної мережі досить довільної структури.

Нейронні мережі можуть реалізовуватися як програмно, так і апаратно. Поступово напрямок нейрокібернетики перетворилося в нейрокомпьютинг – шосте покоління комп'ютерів, що базується на нейронних мережах.

Задачі, які вирішуються за допомогою нейронних мереж:

Класифікація образів. Задача полягає у визначенні належності вхідного образу (наприклад, мовного сигналу або рукописного символу), представленого вектором ознак, до одного або декількох попередньо визначеним класам. До відомих додатків відносяться розпізнавання букв, розпізнавання мови, класифікація сигналу електрокардіограми, класифікація клітин крові.

Кластеризація / категоризація. При вирішенні задачі кластеризації навчальна множина не має міток класів. Алгоритм кластеризації заснований на подобі образів і поміщає схожі образи в один кластер. Відомі випадки застосування кластеризації для видобутку знань, стиснення даних і дослідження властивостей даних.

Апроксимація функцій. Припустимо, що є навчальна вибірка ((x_1,y_1), (x_2,y_2), ..., (x_n,y_n)) (пари даних вхід-вихід), яка генерується невідомою функцією F, спотвореної шумом. Завдання апроксимації полягає в знаходженні невідомої функції F. Апроксимація функцій необхідна при вирішенні численних інженерних і наукових задач моделювання.

Передбачення / прогноз. Нехай задані n дискретних відліків $\{y(t_1), y(t_2), ..., y(t_n)\}$ в послідовні моменти часу t_1, t_2, ..., t_n. Завдання полягає в передбаченні значення $y(t_n+1)$ в наступний момент часу t_n+1. Передбачення / прогноз мають велике значення для прийняття рішень в бізнесі, науці і техніці (передбачення цін на фондовій біржі, прогноз погоди).

Оптимізація. Численні проблеми в математиці, статистиці, техніці, науці, медицині та економіці можуть розглядатися як проблеми оптимізації. Завданням алгоритму оптимізації є знаходження такого рішення, яке задовольняє системі обмежень і максимізує або мінімізує цільову функцію.

Управління. Розглянемо динамічну систему, задану сукупністю $\{u(t), y(t)\}$, де $u(t)$ – вхідний керуючий вплив, а $y(t)$ – вихід системи в момент часу t. У системах управління з еталонною моделлю метою управління є розрахунок такого вхідного впливу $u(t)$, при якому система діє по бажаної траєкторії, заданої еталонною моделлю. Прикладом є оптимальне управління двигуном.

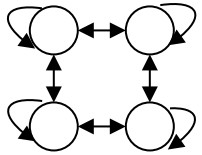
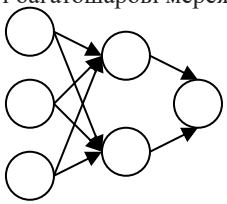
4.2. Класифікація нейромереж

Існує безліч нейронних мереж, які класифікуються за кількома ознаками (таб. 9). Найбільшого поширення набули шаруваті мережі прямого поширення.

Для вирішення конкретної задачі потрібно обрати відповідну нейронну мережу. При цьому потрібно враховувати не тільки перераховані в таблиці критерії, а й архітектуру мережі. Вибір архітектури має на меті визначення кількості слоїв і нейронів у цих слоях. Не існує формального алгоритму за визначенням потрібної архітектури, тому на практиці вибирають або свідомо маленьку мережу та поступово її збільшують або свідомо велику і поступово виявляють невикористовувані зв'язки і скорочують мережу. Нейронна мережа, перш ніж використовуватися на практиці для вирішення будь-якої задачі, повинна бути навчена. Існує безліч алгоритмів, орієнтованих на певні типи мереж і на конкретні завдання, розглянемо алгоритми для одношарової та багатошарової мереж.

Таблиця 9

Класифікація штучних нейронних мереж

Тип	Опис
За топологією (архітектурою)	
Повнозв'язні	Кожен нейрон пов'язаний з іншим нейроном в мережі (через високу складності навчання не використовується). 
Слоїсті (шаруваті)	Нейрони розташовуються слоями, кожен нейрон наступного слою пов'язаний з нейронами попереднього. Є одношарові і багатошарові мережі. 
За типом зв'язків	
Прямого поширення	Всі зв'язки між нейронами йдуть від виходів нейронів попереднього шару до входів нейронів наступного (без зворотних зв'язків між нейронами; до таких мереж відносяться одношаровий і багатошаровий персептрони, мережа радіальних базисних функцій).
Рекурентні	Допускаються зв'язки виходів нейронів наступних шарів з входами нейронів попередніх (зі зворотним зв'язком, від виходів нейронів до входів; до таких мереж відносяться змагальні мережі і мережу Хопфілда).
За організації навчання	
З вчителем	При навчанні використовуються навчальні вибірки, в яких визначені необхідні від мережі вихідні значення, такі мережі використовують для вирішення задач класифікації.
Без вчителя	Нейронна мережа сама в процесі роботи виділяє класи об'єктів і відносить об'єкт до певного класу, такі мережі використовують для задач кластеризації
За типом сигналу	
Бінарні	На вхід нейронних мереж подають тільки нулі або одиниці.
Аналогові	Сигнали, що подаються на входи нейронів можуть бути довільними (числами).

Закінчення таблиці 9

Тип	Опис
За типом структур	
Однорідна	Всі нейрони в нейронної мережі використовують одну функцію активації.
Неоднорідна	Нейрони в нейронної мережі мають різні функції активації.

4.2. Навчання нейромереж

Навчання нейронної мережі – це процес налаштування синаптичних ваг. Іншими словами, необхідно щоб досягався мінімум сумарної функції оцінки:

$$\sum_{i=1}^{N} \|D_i - Y_i\| \to \min \qquad (2)$$

де D_i – бажане вихідне значення на i-нейроні, Y_i – реальне значення на i-нейроні.

Як функція норми виступає покомпонентна сума квадратів елементів вектора Y-Y' (оцінка МНК), або більш спеціалізована. Застосування більш спеціалізованих оцінок прискорює процес навчання мережі. Так, можна використовувати оцінки, що дозволяють задавати вимоги до точності розв'язку задачі. Це дозволяє припиняти процес навчання, коли досягнута задовольняюча користувача точність. Для вирішення задач класифікації можна будувати спеціалізовані оцінки.

Весь процес навчання можна відобразити таким чином (рис. 22).

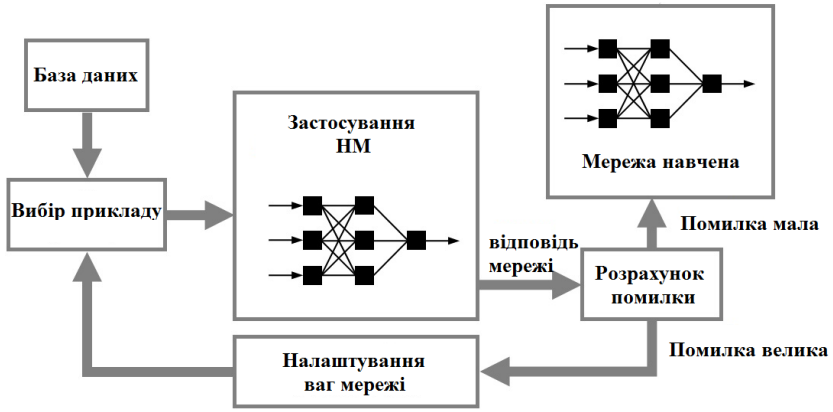


Рис. 22. Процес навчання нейромережі

4.2.1. Навчання нейромереж за Δ-правилом

Найпростіша нейронна мережа – одношарова (рис. 23), що являє собою розташовані паралельно нейрони, які отримують на входи однакові сигнали, але мають різні синаптичні зв'язки. Кількість входів і виходів такої нейронної мережі відповідає кількості нейронів. Такі нейронні мережі можна вивчати за допомогою алгоритму навчання за Δ-правилом.

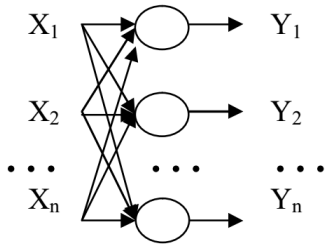


Рис. 23 Одношарова нейронна мережа

Алгоритм навчання за Δ-правилом:

1. крок. Ініціалізація матриці ваг (та порогів, у разі використання порогової функції активації) випадковим чином.

2. крок. Пред'явлення нейронній мережі образу (на вхід подаються значення з навчальної вибірки – вектор X), береться відповідний вихід (вектор D).

3. крок. Обчислення вихідних значень нейронної мережі (вектор Y).

4. крок. Обчислення для кожного нейрона величини розбіжності реального результату з бажаним:

$$\varepsilon_i = d_i - y_i$$

де d_i – бажане вихідне значення на і-нейроні, y_i – реальне значення на і-нейроні.

5. крок. Зміна вагів (і порогів при використанні порогової функції) за формулами:

$$w_{ij}(t+1) = w_{ij}(t) - \eta \cdot \varepsilon_j \cdot x_i$$
$$\theta_i(t+1) = \theta_i(t) - \eta \cdot \varepsilon_i$$

де t – номер поточної ітерації циклу навчання, w_{ij} – вага зв'язку j-входу з і-нейроном, η – коефіцієнт навчання, задається від 0 до 1, x_j – вхідне значення, θ_i – порогове значення і-нейрона.

6. крок. Перевірка умови продовження навчання (обчислення значення помилки і / або перевірка заданого кількості ітерацій). Якщо навчання не завершено то 2 крок, інакше закінчуємо навчання.

Приклад розв'язку задачі

Задача. Прорахувати одну ітерацію циклу навчання за Δ-правилом одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: гіперболічний тангенс (k = 1) і порогову функцію (T = 0,7). В якості навчальної вибірки використовувати таблицю істинності для операцій еквівалентності та диз'юнкції (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

Опис процесу розв'язку

Для навчання нейронної мережі за Δ-правилом необхідно:

1) Графічно відобразити структуру нейронної мережі. Визначити розмірність матриці синаптичних ваг.

2) Визначити навчальну вибірку, представивши її в табличному вигляді.

3) Вибрати вхідні дані, на яких буде розглядатися ітерація циклу навчання.

4) Дотримуючись алгоритму навчання за Δ-правилом, прорахувати одну ітерацію циклу і представити нові синаптичні ваги в матричному вигляді.

Розв'язок

1) За умовою нейронна мережа складається з двох нейронів, значить, входів у одношарової нейронної мережі буде 2 і виходів 2, а синаптичних ваг 4 Перший нейрон має порогову функцію активації, другий – гіперболічний тангенс.

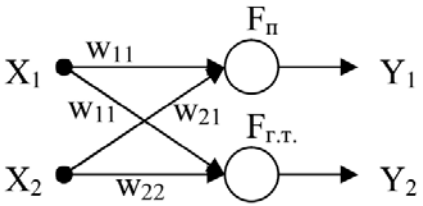


Рис. 24. Нейронна мережа

2) За умовою нейронна мережа бінарна, тому на її входи можуть подаватися тільки нулі і одиниці, так як входів 2, то можливих комбінацій

вхідних значень буде 4 (навчальна вибірка буде складатися з 4 векторів). Вихід першого нейрона відповідно до умови відповідає оператору еквівалентності, а другого – диз'юнкції. Тому таблиця з навчальною вибіркою буде виглядати наступним чином:

X1	X2	D1	D2
0	0	1	0
0	1	0	1
1	**0**	**0**	**1**
1	1	1	1

3) Нехай в якості вектора навчання буде розглядатися 3-й рядок таблиці.

4) Дотримуючись алгоритму навчання за Δ-правилом виконаємо 6 кроків

1. крок: задамо матрицю ваг випадковим чином з інтервалом [0,1]:

Wij(1)	1	2
1	0.7	1
2	0.5	0.2

2. крок: вектор X = {1,0}, вектор D = {0,1}.

3. крок: обчислення вихідних значень нейронної мережі (вектор Y). T=0.7;

$$S_1 = x_1 \cdot w_{11} + x_2 \cdot w_{21} = 1 \cdot 0.7 + 0 \cdot 0.5 = 0.7;$$

$$Y_1 = \begin{cases} 1, при\ S_1 \geq T \\ 0, при\ S_1 < T \end{cases} = \begin{cases} 1, при\ 0.7 \geq 0.7 \\ 0, при\ 0.7 < 0.7 \end{cases} = 1$$

K=1,

$$S_2 = x_1 \cdot w_{12} + x_2 \cdot w_{22} = 1 \cdot 0,9 + 0 \cdot 0,2 = 0,9;$$

$$Y_2 = th\left(\frac{S_2}{k}\right) = \frac{e^{0.9} - e^{-0.9}}{e^{0.9} + e^{-0.9}} \approx 0.71.$$

4. крок:

$$\varepsilon_1 = \left(d_1 - y_1\right) = \left(0 - 1\right) = -1$$

$$\varepsilon_2 = (d_2 - y_2) = \left(1 - 0.71\right) = 0.29$$

5. крок: задаєм η – коефіцієнт навчання від 0, до 1 і змінюємо ваги: Η=0.8

$$w_{11}(2) = w_{11}(1) - 0.8 \cdot \varepsilon_1 \cdot x_1 = 0.7 - 0.8 \cdot (-1) \cdot 1 = 1.5,$$

$$w_{21}(2) = w_{21}(1) - 0.8 \cdot \varepsilon_1 \cdot x_2 = 0.5 - 0.8 \cdot (-1) \cdot 0 = 0.5,$$
$$\theta_1(2) = \theta_1(1) - 0.8 \cdot \varepsilon_1 = 0.7 - 0.8 \cdot (-1) = 1.5,$$
$$w_{12}(2) = w_{12}(1) - 0.8 \cdot \varepsilon_2 \cdot x_1 = 0.9 - 0.8 \cdot 0.29 \cdot 1 = 0.668$$
$$w_{22}(2) = w_{22}(1) - 0.8 \cdot \varepsilon_2 \cdot x_2 = 0.2 - 0.8 \cdot 0.29 \cdot 0 = 0.2,$$

Wij(1)	1	2
1	1.5	0.668
2	0.5	0.2

6. крок: обчислимо середньоквадратичну помилку (можна вибрати інші методи оцінки помилки)

$$\varepsilon_2 = \sum_{i=1}^{H} \left(d_i - y_j\right)^2 = \sum_{i=1}^{H} \varepsilon_i^2 = \varepsilon_1^2 + \varepsilon_2^2 = (-1)^2 + 0.29^2 = 1.0841$$

де H – кількість нейронів. Так як ми розглядаємо одну ітерацію циклу навчання в будь-якому випадку виходимо з циклу.

4.2.2. Алгоритм зворотнього поширення помилки

Багатошарова штучна нейронна мережа (рис. 25) може містити довільну кількість слоїв/шарів (K), кожен слой складається з декількох нейронів, число яких також може бути довільно (H_k – кількість нейронів в шарі), кількість входів n, кількість виходів H = H_K – числу нейронів у вихідному (останньому) шарі.

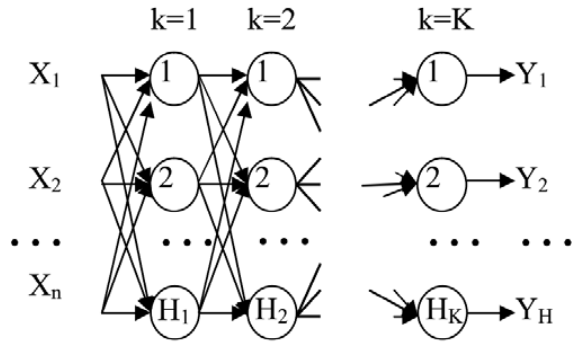


Рис 25. Багатошарова нейронна мережа прямого поширення

Шари між першим і останнім називаються проміжними або прихованими. Ваги в такій мережі мають три індекси i – номер нейрона наступного шару, для якого зв'язок вхідний, j – номер входу або нейрона поточного шару, для якого зв'язок вихідний, k – номер поточного шару в нейронної мережі (для входів, вектора X, $k = 0$)

Багатошарові нейронні мережі прямого поширення навчаються методом зворотного поширення помилки.

Алгоритм навчання методом зворотного поширення помилки:

1. крок. Ініціалізація матриць ваг випадковим чином (в циклах).

2. крок. Представлення нейронній мережі образу (на вхід подаються значення з навчальної вибірки – вектор X) і береться відповідний вихід (вектор D).

3. крок. Прямий прохід: обчислення в циклах виходів всіх слоїв та отримання вихідних значень нейронної мережі (вектор Y).

$$y_i^k = \int\left(\sum w_{ij}^k \cdot y_j^k\right), \qquad (3)$$

$$y_j^0 = x_j,$$
$$y_0^{k-1} = 1,$$
$$x_0 = 1,$$

де y_i^k – вихід i-нейрона k-шару, $\int$ – функція активації, w_{ij}^k – синаптичний зв'язок між j-нейроном шару k-1, та i-нейроном шару k, x_j – вхідне значення.

4. крок. Зворотний прохід: зміна вагів в циклах за формулами:

$$w_{ij}^k(t+1) = w_{ij}^k(t) + \eta \cdot \delta_i^k \cdot y_j^{k-1},$$
$$\delta_i^k = \left(d_i - y_i\right) \cdot y_i \cdot \left(1 - y_i\right) -$$

для останнього (вихідного) слою,

$$\delta_i^k = y_i \cdot \left(1 - y_i\right) \cdot \sum_{l=1}^{H_{k+1}} \delta_l^{k+1} \cdot w_l^{k+1} -$$

для проміжних слоїв, де t-номер поточної ітерації циклу навчання (номер епохи), η – коефіцієнт навчання задається від 0 до 1, y_i^k – вихід інейрона k-шару, w_{ij}^k – синаптична зв'язок між j-нейроном шару k-1, та i-нейроном шару k, d_i – бажане вихідне значення на i-нейроні, y_i – реальне значення на i-нейроні вихідного шару.

5. крок. Перевірка умови продовження навчання (обчислення значення помилки та / або перевірка заданої кількості ітерацій). Якщо навчання не завершено, то 2 крок, інакше закінчуємо навчання. Ефективне значення помилка обчислюється таким чином:

$$\varepsilon = \frac{1}{Q} \cdot \sum_{q=1}^{Q} \sum_{i=1}^{H} \left(d_i - y_i\right)^2,$$

де Q – загальна кількість прикладів, H- кількість нейронів у вихідному шарі, d_i - бажане вихідне значення на інейроне, y_i – реальне значення на інейроне вихідного шару.

Приклад розв'язку задачі

Задача. Прорахувати одну ітерацію циклу навчання методом зворотного поширення помилки багатошарової бінарної неоднорідної нейронної мережі, що складається з 2 шарів, причому в першому шарі знаходиться 2, нейрона і використовується сигмоїдальна функція активації (k = 0.9), а в другому – 1, лінійна (k = 0.7) функція. В якості навчальної вибірки використовувати таблицю істинності для операції «штрих Шеффера» (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

Опис процесу розв'язку

Для навчання нейронної мережі методом зворотного поширення помилки необхідно:

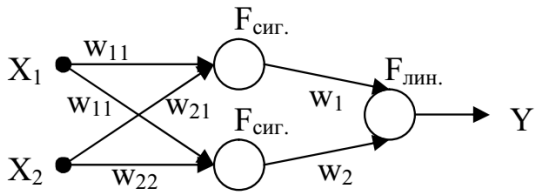
1) Графічно відобразити структуру нейронної мережі. Визначити розмірність і кількість матриць синаптичних ваг (для кожного шару своя матриця).

2) Визначити навчальну вибірку, представивши її в табличному вигляді.

3) Вибрати вхідні дані, на яких буде розглядатися ітерація циклу навчання.

4) Дотримуючись алгоритми навчання методом зворотного навчання помилки прорахувати одну ітерацію циклу і представити нові синаптичні ваги в матричному вигляді.

Розв'язок



1) За умовою нейронна мережа складається з трьох нейронів, два вхідних, один вихідний, значить синаптичних вагів 6, Перший шар нейронів має сигмоїдальну функцію активації, другий – лінійну.

2) За умовою нейронна мережа бінарна, тому на її входи можуть подаватися тільки нулі і одиниці, так як входів 2, то можливих комбінацій вхідних значень буде 4, (навчальна вибірка буде складатися з 4, векторів). Вихід нейронної мережі відповідно до завдання відповідає оператору «штрих Шеффера». Тому таблиця з навчальною вибіркою буде виглядати наступним чином:

X1	X2	D
0	0	1
0	**1**	**1**
1	0	1
1	1	0

3) Нехай в якості вектора навчання буде розглядатися 2-ий рядок таблиці.

4) Дотримуючись алгоритму навчання методом зворотнього поширення помилки, виконаємо 5, кроків:

1. крок: задамо матрицю вагів випадковим чином з інтервалу [0,1]

Wg(1)	**1**	**2**
	0.3	0.8

Wij(1)	**1**	**2**
1	0.6	0.9
2	0.1	0.5

2. крок: вектор X = {0,1}, D = {1}.

3. крок. Прямий прохід: обчислення в циклах виходів всіх слоїв та отримання вихідних значень нейронної мережі (вектор Y).

$$S_1 = x_1 \cdot w_{11} \cdot x_2 \cdot w_{21} = 0 \cdot 0.6 + 1 \cdot 0.1 = 0.1$$

$$S_2 = x_1 \cdot w_{12} \cdot x_2 \cdot w_{22} = 0 \cdot 0.9 + 1 \cdot 0.5 = 0.5$$

$$k = 0.9;$$

$$Y_1 = \frac{1}{1 + e^{-s_1 \cdot k}} = \frac{1}{1 + e^{-0.1 \cdot 0.9}} = 0.5224$$

$$Y_2 = \frac{1}{1 + e^{-s_2 \cdot k}} = \frac{1}{1 + e^{-0.5 \cdot 0.9}} = 0.61$$

$$S_3 = Y_1 \cdot w_1 \cdot Y_2 \cdot w_2 = 0.5224 \cdot 0.3 + 0.61 \cdot 0.8 = 0.64472$$

$$l = 0.7;$$

$$Y = l \cdot S = 0.4513$$

4. крок. Зворотний прохід: зміна вагів:

$$\eta = 0.7;$$

$$\delta^2 = (d - Y) \cdot Y \cdot (1 - Y) = (1 - 0.4513) \cdot 0.4513(1 - 0.4513) = 0.3587$$

$$w_1(2) = w_1(1) + \eta \cdot \delta^2 \cdot Y_1 = 0.3 + 0.7 \cdot 0.3587 \cdot 0.5224 = 0.431$$

$$w_2(2) = w_2(1) + \eta \cdot \delta^2 \cdot Y_2 = 0.8 + 0.7 \cdot 0.3587 \cdot 61 = 0.953$$

$$\delta_1^1 = Y_1 \cdot (1 - Y_1) \cdot \sum_{l=1}^{H_{k+1}} \delta_1^{k+1} \cdot w_1^{k+1} = Y_1 \cdot (1 - Y_1) \cdot \delta^2 \cdot w_1 =$$

$$= 0.5224 \cdot (1 - 0.5224) \cdot 0.3587 \cdot 0.3 = 0.0268$$

$$\delta_2^1 = Y_2 \cdot (1 - Y_2) \cdot \sum_{l=1}^{H_{k+1}} \delta_1^{k+1} \cdot w_1^{k+1} = Y_2 \cdot (1 - Y_2) \cdot \delta^2 \cdot w_2 =$$

$$= 0.61 \cdot (1 - 0.61) \cdot 0.3587 \cdot 0.8 = 0.0682$$

$$w_{11}(2) = w_{11}(1) + \eta \cdot \delta_1^1 \cdot x_1 = 0.6 + 0.7 \cdot 0.0268 \cdot 0 = 0.6$$

$$w_{12}(2) = w_{12}(1) + \eta \cdot \delta_2^1 \cdot x_1 = 0.9 + 0.7 \cdot 0.0682 \cdot 0 = 0.9$$

$$w_{21}(2) = w_{21}(1) + \eta \cdot \delta_1^1 \cdot x_2 = 0.1 + 0.7 \cdot 0.0268 \cdot 1 = 0.119$$

$$w_{12}(2) = w_{22}(1) + \eta \cdot \delta_2^1 \cdot x_2 = 0.5 + 0.7 \cdot 0.0682 \cdot 1 = 0.548$$

Wij(1)	**1**	**2**
1	0.6	0.9
2	0.119	0.548

Wg(1)	**1**	**2**
	0.431	0.953

5. крок:

$$\varepsilon = \sum_{i=1}^{H} (d_i - y_i)^2 = (1 - 0.4513)^2 = 0.237$$

Так як ми розглядаємо одну ітерацію циклу навчання, в будь-якому випадку виходимо з циклу.

4.2. Системи індуктивного моделювання

4.3.1. Загальна постановка задачі моделювання за даними спостережень

Нехай маємо *n* результатів спостережень за функціонуванням об'єкта чи процесу, тобто задано вибірку даних, що містить інформацію про *m* вхідних *X*[*n*×*m*] та одну вихідну змінну y[*n*×*1*], яка,як правило, містить випадковий шум.

У загальному випадку задача індуктивного моделювання зводиться до формування за вибіркою експериментальних даних деякої множини моделей-кандидатів *Ф* різної структури:

$$\hat{y}_f = f(X, \hat{\theta}_f) \tag{4}$$

та пошуку оптимальної моделі з цієї множини як розв'язок задачі дискретної оптимізації за умовою мінімуму зовнішнього критерію селекції *CR*(·):

$$f^* = \arg\min_{f \in \Phi} CR(y, f(X, \hat{\theta}_f)), \tag{5}$$

де оцінки параметрів $\hat{\theta}_f$ для кожної $f \in \Phi$ є розв'язком іще однієї задачі неперервної оптимізації:

$$\hat{\theta}_f = \arg\min_{\theta \in R^{s_f}} QR(y, X, \theta_f), \tag{6}$$

де $QR(\cdot) \neq CR(\cdot)$ – критерій якості розв'язання задачі параметричної ідентифікації кожної окремої моделі, що генерується в процесі структурної ідентифікації, s_f – складність моделі *f* (число оцінюваних параметрів).

Постановка задачі (4) – (6) повністю відповідає завданням моделювання з застосуванням перебірних алгоритмів МГУА, коли скінченну множину Ф можна повністю описати і організувати тим чи іншим чином пошук моделі оптимальної складності. Специфіка ітераційних алгоритмів полягає в тому, що для такої множини – взагалі кажучи, нескінченної – можна лише вказати правило побудови, проте його не можна переглянути, оскільки його формування відбувається в процесі розв'язання задачі (5).

4.3.2. Метод групового урахування аргументів та його основні структурні елементи

Розвиток теорії МГУА сприяв виникненню спектру алгоритмів, кожен з яких, згідно з деякими специфічними умовами конкретного застосування, має свої переваги та недоліки. Алгоритми МГУА можуть відрізнятися за типом базисних функцій, способом ускладнення структури моделі, зовнішніми критеріями тощо. Вибір алгоритму залежить від рівня шуму в даних та їх достатності.

Розглянемо стисло основні алгоритми МГУА, що застосовуються для розв'язання задачі (5) пошуку кращої моделі, та їхні структурні елементи.

Множина Φ формується за допомогою різних генераторів ускладнюваних структур моделей. Усі генератори структур, подібно до методів оптимізації, природним чином поділяються на дві групи – перебірні та ітераційні, що відрізняються способами формування моделей та організацією пошуку мінімуму заданого критерію.

Перебірні алгоритми призначені для розв'язання задач перебором моделей зі скінченної множини Φ, всі елементи якої можуть бути обчислені незалежно за допомогою оцінки параметрів різних варіантів моделей. При цьому (5) є аналогом задачі дискретного програмування, що може бути розв'язана повним або спрямованим перебором. Повний перебір формує всі можливі варіанти структур моделей (4), що містять від одного до m регресорів (незалежних змінних). При спрямованому переборі формується скорочене число моделей з метою отримати результат повного перебору при істотно менших затратах.

Ітераційні алгоритми подібні до класу методів оптимізації за допомогою послідовних наближень, але використовують принцип неостаточних рішень (свободу вибору). В залежності від способу організації послідовних наближень ітераційні алгоритми МГУА поділяються на дві основні групи: багаторядні та релаксаційні. Багаторядні алгоритми побудовані за аналогією з біологічною селекцією живих організмів: ускладнення моделей від r-го ряду до $(r+1)$-го відбувається за рахунок попарного «схрещування» виходів моделей попереднього ряду з наступною селекцією кращих варіантів отриманих моделей. У таких алгоритмах процедура ускладнення припиняється після початку зростання значень критерію *CR*.

Алгоритми МГУА релаксаційного типу відповідають методу послідовних наближень за допомогою групових релаксацій: на кожному ряді (кроці наближення) кращі моделі ускладнюються тільки за рахунок схрещування виходів моделей попереднього ряду з початковими аргументами. У цих алгоритмах момент зупинки розв'язання задачі визначається за фактом початку зростання значень критерію *CR* або його стабілізації.

Для оцінювання параметрів у алгоритмах МГУА застосовується перш за все МНК, де в якості *QR* в (6) виступає сума квадратів помилок:

$$QR(\theta_f) = \left\| y - \hat{y}_f \right\|^2 = \left\| y - X\hat{\theta}_f \right\|^2. \qquad (7)$$

МНК є найкращим методом оцінювання при виконанні припущення про незалежний шум з нульовим середнім і скінченною дисперсією, застосування якого дає розв'язок задачі оцінювання (7) у вигляді

$$\hat{\theta}_f = (X_f^T X_f)^{-1} X_f^T y, \qquad (8)$$

де X_f – підматриця матриці X, що містить стовпці, які відповідають частинній моделі $f \in \Phi$, яка розглядається.

Критерії порівняння якості моделей, які генеруються у процесі перебору за МГУА, описано нижче.

Класи моделей які застосовуються в МГУА. У практиці моделювання зустрічаються такі типи задач:

1) побудова регресійних моделей статичних об'єктів;
2) моделювання часових рядів або процесів;
3) моделювання динамічних об'єктів, процесів та систем.

Коротко розглянемо ці типи задач.

Моделювання статичних об'єктів. У цьому випадку необхідно побудувати багатовимірну регресійну модель виду

$$y_i = b^T g(x_i) + \xi_i = \sum_{j=1}^{m} b_j g_j(x_i) + \xi_i. \qquad (9)$$

де $i = \overline{1 \ldots n}$ – номер точки спостереження, $g(x)$ – відома m-вимірна вектор-функція від MX вхідних змінних, b – невідомий m-вимірний вектор параметрів, ξ_i – незалежний некорельований випадковий вектор з нульовим середнім і скінченною невідомою дисперсією.

У якості базисних функцій $g(x)$ в алгоритмах МГУА застосовуються перш за все поліноми:

$$b^T g(x) = \sum_{j=1}^{m} b_j \prod_{v=1}^{MX} x_v^{c_{jv}}, \qquad (10)$$

тоді кожен компонент $g(x)$ є одночленом цього полінома. Якщо при цьому степені c_{jv} приймають цілі значення 0, 1, 2, ... і враховується обмеження

$$\sum_{v}^{MX} c_{jv} = C, \qquad C = 0, 1, ..., ST, \qquad (11)$$

то поліном (10) (порядок слідування і структура одночленів) однозначно задається числами *MX* і *ST*, а число його членів дорівнює

$$m = \prod_{j=1}^{MX} \frac{ST + j}{j}. \qquad (12)$$

Моделі часових рядів. Для моделювання процесів як часових рядів в алгоритмах МГУА застосовуються поліноміальні, тригонометричні, експоненціальні, логарифмічні та логістичні базисні функції. Очевидно, що можна моделювати тренди процесів з урахуванням присутності тільки однієї вхідної змінної – часу.

Крім вказаних класів, для моделювання часових рядів застосовується також клас динамічних моделей у вигляді авторегресійних залежностей:

$$y_k = \sum_{i=1}^{LY} a_i y_{k-1} + \xi_k \ , \qquad (13)$$

де *LY* – порядок авторегресії (число врахованих запізнюваних значень).

Моделі динамічних об'єктів. Третій тип задач моделювання динаміки розв'язується перш за все у класі лінійних динамічних моделей. При цьому в моделі є також частина, що відображає вплив вектора зовнішніх незалежних впливів x. Тоді загальний вигляд моделі динаміки у припущенні її використання для прогнозування на крок уперед подається таким рівнянням:

$$y_k = \sum_{r=1}^{MY} \sum_{\alpha=1}^{LY} a_\alpha^r y_{k-\alpha}^r + \sum_{j=1}^{MX} \sum_{v=1}^{LY} b_v^j x_{k-v+1}^j + \xi_k, \qquad (14)$$

де LY, LX – число враховуваних минулих значень (запізнень) для вихідних і вхідних змінних відповідно, $a_1,..., a_{LY}$, $b_1,..., b_{LX}$ – вектори невідомих параметрів, ξ_k має смисл вхідного процесу типу білого шуму.

В кожному алгоритмі МГУА є блок перетворення даних відповідно до заданого класу моделей, в результаті чого формується стандартна задача структурно-параметричної ідентифікації, де специфіка типу початкової задачі вже у явному вигляді не враховується. Наприклад, вирази (9) та (14) зводяться до вигляду звичайної регресійної моделі

$$y_i = f(x_i, \theta) + \xi_i = x_i^T \theta + \xi_i, \qquad (15)$$

якщо в (9) позначити $\theta \overset{\Delta}{=} b$, $x \overset{\Delta}{=} g(x)$, а в (14) $\theta \overset{\Delta}{=} (a_1,...,a_{LY}, b_1,...,b_{LX})^T$, $x = (y_{k-1},..., y_{k-LX}, x_k,..., x_{k-LX+1})^T$, тоді задача (5) полягає у визначенні оптимальної структури і значень параметра θ, або – в термінах регресійного аналізу – побудові моделі з оптимальною підмножиною регресорів.

Критерії, що застосовуються в алгоритмах МГУА. Вони базуються на розбитті вибірки даних на частини і за рахунок цього неявно (автоматично) забезпечують «штрафування» надмірного ускладнення моделі. При цьому обчислення оцінок параметрів і значень критеріїв відбувається на різних частинах вибірки. Найпростішим випадком є розбиття на дві підвибірки:

$$W = \left[X \vdots y \right] = \begin{bmatrix} A \\ B \end{bmatrix} = \begin{bmatrix} X_A \vdots y_A \\ X_B \vdots y_B \end{bmatrix}, \quad n = n_A + n_B$$

Де *A* – навчальна, *B* – перевірочна, *W* – робоча вибірки, $W = A \cup B$.

Наведемо обчислювальні формули для основних зовнішніх критеріїв груп *точності* та *узгодженості*. Критерій вибору моделі може бути названий зовнішнім, якщо він отриманий за допомогою додаткової інформації, яка не міститься в даних, що використовувалися при обчисленні параметрів моделей. Наприклад, така інформація міститься в перевірочній вибірці.

Критерії точності. Найбільш уживаним серед критеріїв точності є критерій *регулярності*, який розраховується для заданої складності моделі *S*:

$$AR_B(s) = AR_{B|A}(s) = \left\| y_B - \hat{y}_{B|A}(s) \right\|^2 = \left\| y_B - X_{Bs} \hat{\theta}_{As} \right\|^2, \qquad (16)$$

де запис $AR_{B\,B}\ (S)$ означає «помилка на *B* моделі складності *S*, коефіцієнти якої отримано на *A*».

Надалі опустимо в формулах залежність від *S*, пам'ятаючи, що значення критеріїв розглядаються для конкретної складності *S*.

Критерій регулярності, розрахований на тій же вибірці, де отримана модель, є залишковою сумою квадратів (*RSS* – Residual Sum of Squares):

$$AR_{A|A} = RSS_A;\ \ AR_{B|B} = RSS_B, \tag{17}$$

де *RSS* обчислюється як в (7) або, що те ж саме, як $RSS = y^T y - y^T\ \hat{y}$, але на відповідній підвибірці.

В якості критерію точності може бути також критерій стабільності:

$$AS = AR_{W|A} + AR_{W|B} = \left\| y_W - X_W \hat{\theta}_A \right\|^2 + \left\| y_W - X_W \hat{\theta}_B \right\|^2. \tag{18}$$

Критерії узгодженості (незміщеності). Критерії цієї групи відображають вимогу, щоб моделі на *A* і *B* мінімально відрізнялись.

Найвідоміша форма цього типу критеріїв – критерій незміщеності рішень:

$$CB = CB_{W|A,B} = \left\| \hat{y}_W(A) - \hat{y}_W(B) \right\|^2 = \left\| X_W \hat{\theta}_A - X_W \hat{\theta}_B \right\|^2. \tag{19}$$

Точність і незміщеність – різні показники якості моделей, не взаємно замінні, особливо при неточних даних. За відсутності шуму всі критерії (внутрішні чи зовнішні) рівнозначні, бо виявляють фізичну модель. Достатня точність моделі на всій наявній вибірці даних не гарантує задовільної точності на довільній наступній, що свідчить про важливість забезпечення незміщеності.

4.3.3. Класичний багаторядний алгоритм

У найбільш відомому класичному багаторядному алгоритмі МГУА завдання побудови оптимальної моделі (4) – (6) вирішується індуктивним шляхом: будуються моделі поступово ускладнюваної структури, причому процес ускладнення має характер ітерацій, коли кращі попередні результати використовуються на наступному ряді (ітерації). Ускладнення відбувається за єдиним правилом, що дозволяє будувати як завгодно складну модель від великого числа змінних (аргументів), що характеризують об'єкт моделювання.

Означення довільного ітераційного алгоритму МГУА, як і взагалі будь-якої ітераційної процедури, передбачає визначення:

1) матриці початкового наближення;
2) оператора переходу до наступної ітерації;
3) правила зупинки.

З урахуванням цього означимо базовий класичний багаторядний алгоритм МГУА, який був історично першим і в сучасній термінології називається *багаторядний ітераційний алгоритм* БІА МГУА.

1. Матрицею початкового наближення алгоритму БІА є початкова матриця експериментальних чи статистичних даних, що містить $\overline{m}$ стовпців вимірювань вхідних змінних $X = (x_1, \ldots, x_m),\ x_j[n \times 1],\ j = \overline{1,m}$.

2. Загальним виглядом оператора переходу до наступної ітерації є деяка функція $z = f(u,v)$, де *u*, *v* – довільна пара векторів розв'язків попередньої ітерації, *z* – вектор виходу моделі, який буде вхідною змінною для наступної ітерації. Ця перехідна функція $z = f(u,v)$, яка в МГУА називається *частинним описом*, може бути лінійною, білінійною або квадратичною:

$$\begin{gathered} z = f(u,v) = a_0 + a_1 u + a_2 v; \\ z = f(u,v) = a_0 + a_1 u + a_2 v + a_3 uv; \\ z = f(u,v) = a_0 + a_1 u + a_2 v + a_3 uv + a_4 u^2 + a_5 v^2. \end{gathered} \tag{20}$$

Стандартним частинним описом базового алгоритму МГУА була квадратична функція, тобто він призначався для побудови моделей переважно складних нелінійних об'єктів.

3. Алгоритм зупиняється при виконанні умови $CR_{r+1} \geq CR_r$, де *CR* – критерій якості моделі, *r* – номер ітерації (ряду). Зазвичай застосовується критерій регулярності:

$$AR_{B|A} = \left\| y_B - \hat{y}_B \right\|^2 = \left\| y_B - X_B \hat{\theta}_A \right\|^2, \tag{21}$$

що базується на розбитті вибірки даних $W = (X\ y)$ на дві неперетинні підвибірки – навчальну *A* і перевірну *B*, $W^T = (A^T B^T)$, де $\hat{y}_B$ – оцінка виходу на *B* за моделлю, вектор параметрів $\hat{\theta}_A$ якої обчислено на *A*.

Зазначені основні характеристики БІА не описують повністю роботу алгоритму, тому розглянемо детальніше операції, що виконуються на 1-му та довільному (*r* +1)-му рядах.

Перша ітерація (перший ряд). З множини входів $X = \{x_1, x_2, ..., x_m\}$ вибираються пари аргументів x_i, x_j, і формуються частинні описи виду (20), тобто $y_l^1 = f(x_i, x_j)$, $l = 1,2,\ldots,C_m^2$, та за МНК для кожного опису за даними навчальної вибірки знаходяться оцінки невідомих параметрів

(коефіцієнтів) $\hat{a}_0, \hat{a}_1, \hat{a}_2, \ldots$ За обраним критерієм на перевірочній вибірці обирають F кращих моделей $\hat{y}_k^1, k = \overline{1,F}$, тобто реалізують процедуру селекції, де F називається свободою вибору. Виходи цих моделей служать аргументами-входами для конструювання моделей наступного ряду. Далі знаходиться мінімальне $CR_{\min}^1$ серед усіх F значень критерію на першому ряді.

Довільна (r+1)*-ша ітерація* (*ряд* r+1): З векторів-аргументів $\hat{y}_k^r, k = \overline{1,F}$ попереднього r-го ряду формуються всі можливі частинні описи виду (20), тобто

$$y_l^{r+1} = f(y_i^r, y_j^r),\ l = 1,2,\ldots,C_F^2,\quad i,j = \overline{1,F} \qquad (22)$$

та за МНК на A знаходяться оцінки параметрів. Потім за критерієм (21) відбираються F кращих моделей $\hat{y}_k^{r+1}$, $k = \overline{1,F}$, серед яких знаходиться $CR_{\min}^{r+1}$ та перевіряється умова $CR_{\min}^{r+1} \geq CR_{\min}^r$, при виконанні якої ітераційний процес зупиняється, інакше перехід до наступного ряду. У випадку зупинки процесу оптимальною приймається модель, яка відповідає значенню $CR_{\min}^r$ на попередньому r-му ряді. Структуру цього алгоритму подано на рис. 26.

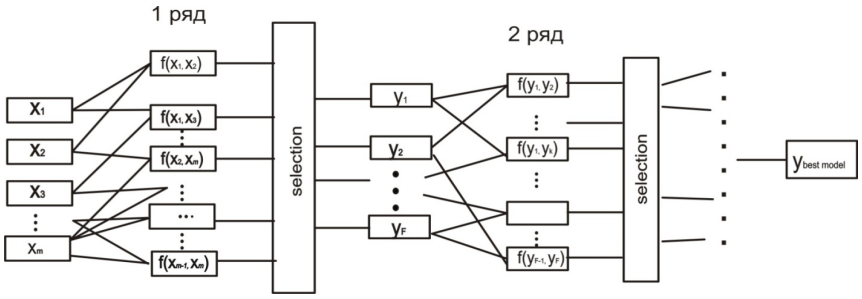


Рис. 26. Структура багаторядного ітераційного алгоритму БІА МГУА

Відзначимо, що така багаторядна процедура послідовної генерації структур моделей імітує процес біологічної селекції з попарним схрещуванням батьківських індивідуумів, тобто цей алгоритм відноситься до групи еволюційних «інспірованих природою» (nature inspired) методів обчислювального інтелекту.

Високу ефективність ітераційних алгоритмів демонструють численні приклади розв'язання практичних задач моделювання для цілей прогнозу та управління. Вони працездатні при дуже великих m – порядку 1000 – і дозволяють будувати лінійні та нелінійні моделі, причому навіть у вироджених задачах при $n < m$, тобто коли довжина вибірки менша від кількості аргументів.

Проте цей класичний алгоритм БІА МГУА має досить істотні недоліки:

1) можливість втрати інформативних аргументів, якщо вони були виключені з процесу селекції на початкових рядах перебору;

2) можливість закріплення неінформативних аргументів, якщо вони були включені на початку перебору;

3) при квадратичному частинному описі експоненційно зростає степінь полінома: 1, 2, 4, 8, ...;

4) зі зростанням числа рядів вектори виходів кращих моделей стають дедалі більш корельованими, що погіршує обумовленість систем рівнянь для оцінювання параметрів.

Далі розглянемо шляхи подолання цих недоліків.

4.3.4. Узагальнений ітераційний алгоритм, УІА

Узагальнений ітераційний алгоритм (рис. 27) поєднує в собі наступні модифікації:

– пари формуються лише з проміжних аргументів (багаторядний ітераційний алгоритм);

– пари формуються лише з проміжних та початкових аргументів (релаксаційний ітераційний алгоритм);

– пари формуються як лише з проміжних аргументів, так і з проміжних та початкових (комбінований ітераційний алгоритм);

– у всіх трьох варіантах застосовується комбінаторна оптимізація складності частинних описів – як лінійних, так і білінійних чи квадратичних.

Формально в загальному випадку для ряду r визначимо УІА МГУА так:

1) вхідною матрицею є $X_{r+1} = (y_1^r, \ldots, y_F^r, x_1, \ldots, x_m)$;

2) застосовуються оператори переходу виду (22) та $y_l^{r+1} = f(y_i^r, x_j), l = 1,2,\ldots,Fm, i = \overline{1,F}, j = \overline{1,m}$ з квадратичним частинним описом;

3) для кожного опису знаходиться оптимальна структура:

$$f(u,v) = a_0 d_1 + a_1 d_2 u + a_2 d_3 v + a_3 d_4 uv + a_4 d_5 u^2 + a_5 d_6 v^2, \quad (23)$$

$$d_k = \{0,1\},\ d_{opt} = \arg\min_{l=1,q} CR_l,\ q = 2^p - 1,\ f_{opt}(u,v) = f(u,v,d_{opt}).$$

4) алгоритм зупиняється при виконанні умови $CR^{r+1}_{\min} \geq CR^{r}_{\min}$ і оптимальна модель відповідає значенню $CR^{r}_{\min}$ на *r*-му ряді.

Запропонований варіант гібридизації структур дає змогу істотно вдосконалити архітектуру класичного багаторядного алгоритму МГУА і завдяки цьому забезпечити такий комплекс нових функціональних властивостей:

– збереження інформативних аргументів, що могли бути відсіяні на початкових етапах моделювання;

– відсіювання неінформативних аргументів, що могли бути включені на початкових етапах;

– запобігання переускладненню побудованої моделі за рахунок комбінаторної оптимізації складності частинних моделей.

– усунення «виродження» частинних моделей.

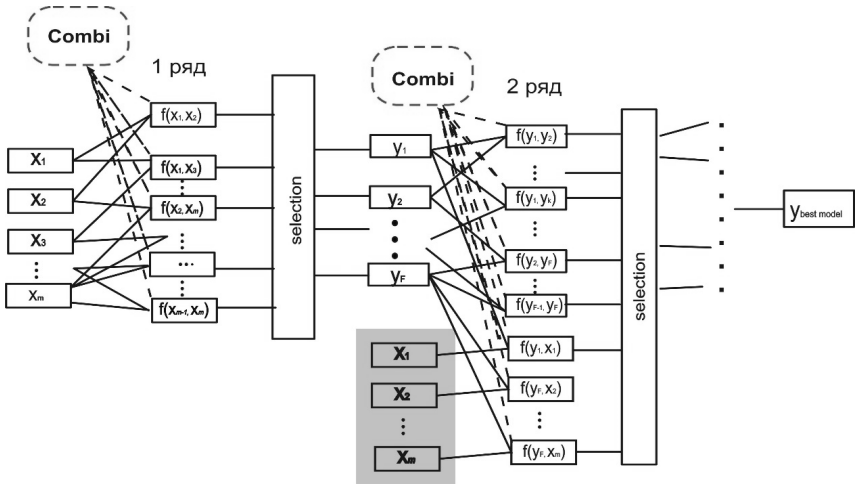


Рис. 27. Схема роботи узагальненого ітераційного алгоритму УІА МГУА

Тобто ця алгоритмічна структура дозволяє не тільки узагальнити основні структури розроблених раніше ітераційних алгоритмів МГУА і одночасно отримати нові їх варіанти, але і в комплексі усунути всі чотири зазначених вище недоліки класичного багаторядного алгоритму МГУА.

Зазначимо, що узагальнений алгоритм є втіленням ідеї О.Г. Івахненка про нейромережу з активними нейронами, описано нижче – тут у кожному вузлі багаторядної мережі МГУА структура перехідної функції оптимізується за допомогою комбінаторного алгоритму МГУА.

4.3.5. МГУА – як поліноміальна нейромережа

Метод групового урахування аргументів (МГУА) нині широко використовується при вирішенні різноманітних завдань і активно застосовується там, де звичайні алгоритмічні рішення виявляються неефективними або зовсім неможливими. З метою підвищення точності та розширення горизонтів застосування багатьма фахівцями було досліджено деякі аспекти МГУА і запропоновано та розроблено так звані *гібридні алгоритми*. Зараз активно розвивають МГУА-подібні системи на основі багаторядного алгоритму вітчизняні та закордонні вчені в Україні, Чехії, Японії тощо. Зокрема, описано використання генетичного алгоритму для оптимізації структури багаторядного МГУА, тоді як коефіцієнти моделей знаходять за допомогою методу сингулярного розкладу (SVD − Singular Value Decomposition).

У 1990-х роках, коли штучні нейрнні мережі отримали широку популярність, О.Г. Івахненко і користувачі його методу почали називати типову структуру алгоритму МГУА також нейромережею. Більше того, в останні роки фахівці з нейромереж за кордоном найчастіше називають алгоритм МГУА Polinomial Neural Network (PNN), тобто Поліноміальна Нейронна Мережа (ПНМ). При цьому основний елемент ітераційних алгоритмів − частинний опис − є елементарним нейроном ПНМ МГУА, його структуру для квадратичного опису зображено на рис. 28. Оригінальність і ефективність нейромережі з таких нейронів полягає у швидкості процесу локального налаштування ваг нейронів та автоматичної глобальної оптимізації (самоорганізації) структури мережі − числа вузлів і кількості рядів (прихованих шарів).

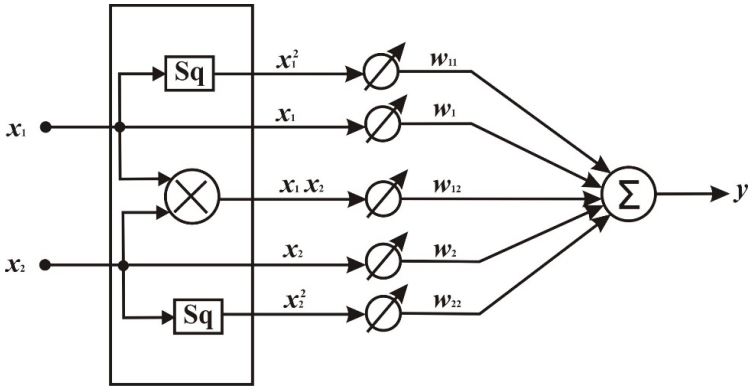


Рис. 28. Структура нейрону МГУА з квадратичним частинним описом

Ідея нейромережі з активними нейронами. На ранніх етапах розробки теорії МГУА було відзначено подібність між нейронними мережами і багаторядним алгоритмом МГУА. О. Г. Івахненко в одній зі статей стверджував, що відмінності між перцептроном і МГУА не фундаментальні, тому цілком прийнятно називати МГУА-системи «системами перцептронного типу».

Перш ніж дати означення активного нейрона, треба встановити, що таке пасивний нейрон. У таких нейронах не закладено процес оптимізації множини вхідних змінних. Вони встановлюються загалом у складному процесі самоорганізації всієї системи з багатьох однотипних нейронів. Структурна оптимізація за зовнішнім критерієм та отримання оптимальних нефізичних моделей в перцептронах не передбачені. Здійснюється тільки параметрична оптимізація за допомогою зворотного перерахунку помилки («back propagation»). За точністю перцептрони майже не поступаються поліноміальним алгоритмам МГУА, якщо навчальна вибірка досить довга, дисперсія шуму невелика, а вибірка містить змінні, дискретизовані на мале число рівнів, тобто у випадках, коли оптимальною є фізична модель. Перевага в точності моделей МГУА досягається за коротких вибірок неперервних зашумлених змінних, тобто у разі, коли оптимальною є нефізична модель. Переваги перцептронів і поліноміальних алгоритмів МГУА об'єднуються в нейромережах з активними нейронами.

Подібність між структурою нейромереж і алгоритмів МГУА спонукала дослідників вивчити, як вони можуть бути об'єднані. О.Г. Івахненко запропонував комбінований метод, який поширює теорію самоорганізації з ізольованих моделей на активні нейронні мережі. Запропонований алгоритм відомий як «нейронні мережі з активними нейронами» застосовується замість пасивного нейрона в багатьох алгоритмах МГУА.

Як багаторядні, так і комбінаторні алгоритми МГУА можуть бути використані в якості активних нейронів. О. Г. Івахненко запропонував модифікувати комбінаторний алгоритм у алгоритм з активними нейронами, що веде до збільшення точності та скорочення часу обчислень.

Перевага нейромережі з активними нейронами у порівнянні зі звичайною нейромережею з пасивними нейронами полягає в тому, що самоорганізація мережі спрощується: кожен нейрон знаходить необхідні зв'язки та свою складність (степінь нелінійності) сам у процесі своєї самоорганізації.

Нейромережі з активними нейронами були застосовані для прогнозування економічних та екологічних систем.

МГУА-подібна нейромережа зі зворотним зв'язком – МГУА-подібна нейронна мережа зі зворотним зв'язком, яка може сама обирати оптимальну нейромережеву архітектуру. МГУА-подібна нейронна мережа може автоматично організувати архітектуру, використовуючи метод самоорганізації, який є основою алгоритму МГУА.

Архітектура такої МГУА-подібної нейронної мережі має цикл зворотного зв'язку. У цьому алгоритмі виходи нейронів поєднані зі вхідними змінними системи (зворотний зв'язок) для подальшого обчислення. Таким чином, складність нейронної мережі зростає поступово. В цьому алгоритмі нейронна мережа автоматично обирає одну з трьох оптимальних архітектур: НМ з сигмоїдальною функцією, радіальна НМ, поліноміальна НМ. До того ж структурні параметри, наприклад, кількість шарів, кількість нейронів у прихованих шарах і вхідних змінних обираються автоматично за критерієм мінімізації помилки, прийнятого як критерій Акаіке:

$$AIC(s_f) = -2\ln\phi(X, \hat{\theta}_f) + 2s_f, \qquad (24)$$

де φ – значення функції правдоподібності.

Крім того, для кожної нейроархітектури використовується два види нейронів, які називаються перший і другий тип нейрона. Перший тип

нейрона має дві вхідні змінні, а другий тип має r вхідних змінних, де r – номер ряду.

Запропонована МГУА-подібна нейронна мережа зі зворотним зв'язком була успішно використана для розв'язання медичної задачі розпізнавання 3-вимірного зображення легенів.

Ідея, коли виходи подаються на вхід, була фактично частково реалізована в багаторядному алгоритмі з селекцією первинних аргументів. В цьому дослідженні така ідея теж представлена у програмній реалізації розробленого узагальненого алгоритму.

Еволюція груп адаптивних моделей. Еволюційний алгоритм під назвою GAME (group of adaptive model evolution – еволюція груп адаптивних моделей), розроблений в Чеському технічному університеті в Празі, базується на МГУА. В GAME коефіцієнти передавальних функцій вузлів нейромережі оцінюються за вибіркою, що описує систему, яка моделюється.

Головні модифікації цієї МГУА-подібної системи:

– передавальна функція вузла може обиратися з декількох типів: лінійна, поліноміальна, логістична, РБФ, перцептронна тощо. Кожен тип вузла має свій власний алгоритм оцінювання коефіцієнтів. Вибір типу вузлів, що утворюють мережу, визначається значенням критерію. Це так звана *гетерогенна* (неоднорідна) структура нейронної МГУА-подібної мережі;

– кількість входів вузла зростає з ростом глибини вузла в мережі, передавальні функції вузлів відображають зростаючу кількість входів;

– в мережі існують міжрядні зв'язки;

– у процесі побудови мережі не знаходяться усі можливі розміщення вузлів, а лише випадкові підмножини цих розміщень. Оригінальний алгоритм БІА МГУА формує одну оптимальну модель, а запропонований алгоритм – групу моделей, які є локально оптимальними для своєї підмножини розміщень.

Модифікований таким чином МГУА-подібний гібридного типу генерує групу моделей за єдиною навчальною вибіркою. Випадкові процеси впливають на процедуру конструювання. Початкові значення ваг та коефіцієнтів вузлів призначаються випадковим чином. Передавальні функції багатьох типів вузлів визначаються псевдовипадково при ініціалізації вузла. Входи вузлів також відбираються випадково. Внаслідок цього топологія моделей, побудованих за однією навчальною вибіркою, відрізняється (рис. 29).

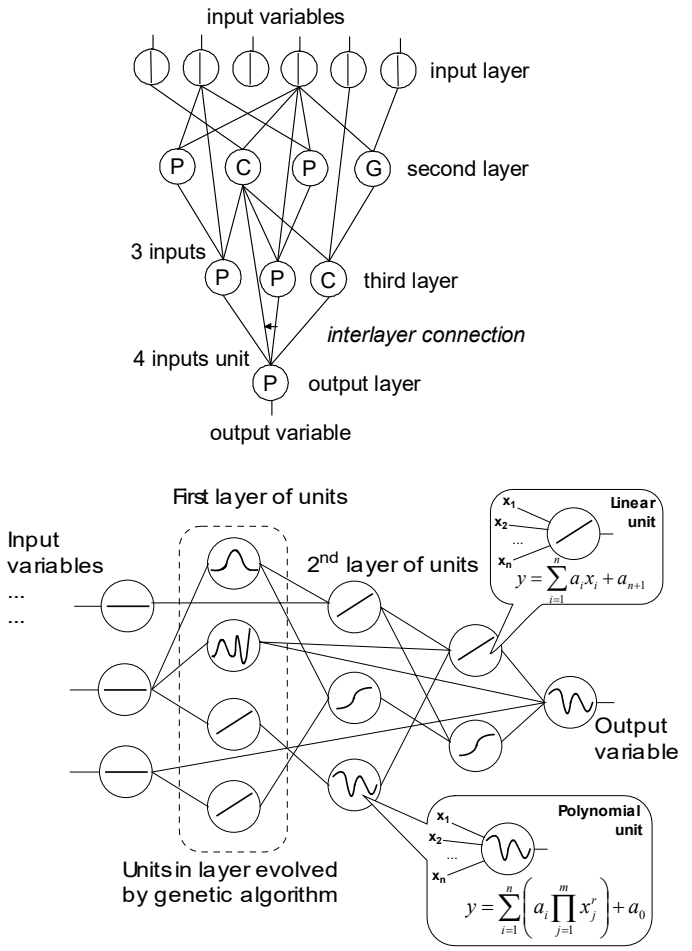


Рис. 29. Структура GAME-мережі

Інші шляхи розвитку ідей самоорганізації. Є багато нових тенденцій у розвитку методів самоорганізації моделей на основі МГУА: створення гібридних архітектур у поєднанні з алгоритмами обчислювального інтелекту, застосування розпаралелювання операцій, побудова нейромереж з нечіткими функціями активації та багато інших.

4.3. Завдання для самоконтролю

Оберіть правильні відповіді.

1. Хто розробив перший нейрокомп'ютер?
- А) У. Маккалок
- Б) М. Мінський
- В) Ф. Розенблатт
- Г) немає правильної відповіді

2. Які завдання не вирішують нейронні мережі?
- А) класифікації
- Б) апроксимації
- В) маршрутизації
- Г) управління
- Д) кодування

3. Яку функцію не може вирішити одношарова нейронна мережа?
- А) логічне «ні»
- Б) підсумовування
- В) логічне «виключає або»
- Г) логічне «або»

4. Що з переліченого нижче відноситься до персептрону?
- А) одношарова нейронна мережа
- Б) нейронна мережа прямого поширення
- В) багатошарова нейронна мережа
- Г) нейронна мережа із зворотними зв'язками
- Д) створений Ф. Розенблаттом
- Е) створений У. Маккалоком і В. Піттом

5. Хто написав книгу «Персептрони»?
- А) У. Маккалок та В. Пітт
- Б) М. Мінський та С. Папертi
- В) Ф. Розенблатт

6. Яку нейронну мережу вивчають за допомогою дельта-правила?
- А) одношарову нейронну мережу
- Б) нейронну мережу прямого поширення
- В) нейронну мережу з зворотними зв'язками
- Г) мережу Хопфілда
- Д) немає правильної відповіді

7. Яку нейронну мережу вивчають за допомогою алгоритму зворотного поширення помилки?
- А) одношарову нейронну мережу
- Б) багатошарову нейронну мережу прямого поширення
- В) багатошарову нейронну мережу з зворотними зв'язками
- Г) немає правильної відповіді

8. Які з перерахованих мереж є рекурентними?
- А) персептрон
- Б) мережу Хопфілда
- В) мережу радіальних базисних функцій
- Г) немає правильної відповіді

9. Хто запропонував МГУА?
- А) Івахненко О.Г.
- Б) Івахненко Г.О.
- В) Степашко В.С.
- Г) немає правильної відповіді

4.4. Завдання для самостійної роботи

Навчання НМ за Δ-правилом

1. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 2 нейронів і має порогову функцію активації (T = 0,7). В якості навчальної вибірки використовувати таблицю істинності для операцій диз'юнкції і імплікації (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

2. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 2 нейронів і має лінійну функцію активації (k = 0,6). В якості навчальної вибірки використовувати таблицю істинності для операцій кон'юнкції і диз'юнкції (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

3. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 2 нейронів і має сигмоїдальну функцію активації (k = 1). В якості навчальної вибірки використовувати таблицю істинності для операцій імплікації

і кон'юнкції (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

4. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 2 нейронів і має функцію активації гіперболічний тангенс (k = 1). В якості навчальної вибірки використовувати таблицю істинності для операцій еквівалентності і імплікації (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

5. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів має функції активації: гіперболічний тангенс (k = 2) і порогову функцію (T = 0,5). В якості навчальної вибірки використовувати таблицю істинності для операцій еквівалентності і кон'юнкції (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

6. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: сигмоїдальну (k = 1) і лінійну (k = 0,6). В якості навчальної вибірки використовувати таблицю істинності для операцій імплікації і кон'юнкції (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

7. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: лінійну (k = 0,7) і порогову (T = 0,75). В якості навчальної вибірки використовувати таблицю істинності для операцій кон'юнкції і еквівалентності (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

8. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: порогову (T = 0,8) і сигмоїдальну (k = 1). В якості навчальної вибірки використовувати таблицю істинності для операцій кон'юнкції і імплікації (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

9. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: гіперболічний тангенс (k = 2) і лінійну

(k = 0.8). В якості навчальної вибірки використовувати таблицю істинності для операцій диз'юнкції і еквівалентності (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

10. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: гіперболічний тангенс (k = 2) і сигмоїдальну (k = 0,9). В якості навчальної вибірки використовувати таблицю істинності для операцій імплікації і диз'юнкції (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

11. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 3 нейронів і має функцію активації гіперболічний тангенс (k = 3). В якості навчальної вибірки використовувати таблицю істинності для X1 & X2 → X3, X1 & X2 і X2 → X3 (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

12. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 3 нейронів і має сигмоїдальну функцію активації (k = 1). В якості навчальної вибірки використовувати таблицю істинності для X1 → X2 & X3, X1 & X2 і X1 & X3 (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

13. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 3 нейронів і має лінійну функцію активації (k = 0,9). В якості навчальної вибірки використовувати таблицю істинності для X3 → X1 & X2, X2 & X3, X2 → X3 (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

14. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової бінарної однорідної нейронної мережі, що складається з 3 нейронів і має порогову функцію активації (T = 0,4). В якості навчальної вибірки використовувати таблицю істинності для (X2 → X1) & X3 (не використовувати перший рядок таблиці). Синаптичні ваги задати випадковим чином.

15. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової аналогової однорідної нейронної мережі, що складається

з 3 нейронів і має лінійну функцію активації (k = 0,9). Синаптичні ваги і навчальну вибірку задати випадковим чином (НЕ нулі).

16. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової аналогової однорідної нейронної мережі, що складається з 3 нейронів і має сигмоїдальну функцію активації (k = 0,8). Синаптичні ваги і навчальну вибірку задати випадковим чином (НЕ нулі).

17. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової аналогової однорідної нейронної мережі, що складається з 3 нейронів і має порогову функцію активації (T = 0,8). Синаптичні ваги і навчальну вибірку задати випадковим чином (НЕ нулі).

18. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової аналогової однорідної нейронної мережі, що складається з 3 нейронів і має функцію активації – гіперболічний тангенс (k = 1). Синаптичні ваги і навчальну вибірку задати випадковим чином (НЕ нулі).

19. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової аналогової неоднорідної нейронної мережі, що складається з 3 нейронів і має функції активації: сигмоїдальну (k = 1), лінійну (k = 0,8) і порогову (T = 0,5). Синаптичні ваги і навчальну вибірку задати випадковим чином (НЕ нулі).

20. Прорахувати одну ітерацію циклу навчання по Δ-правилу одношарової аналогової неоднорідної нейронної мережі, що складається з 3 нейронів і має функції активації: гіперболічний тангенс (k = 1), сигмоїдальну (k = 0,8) і порогову (T = 0,6). Синаптичні ваги і навчальну вибірку задати випадковим чином (НЕ нулі).

4.5. Лабораторний практикум

4.5.1. Лабораторна робота 9. Побудова НМ засобами MATLAB

Мета – дослідження процесів побудови інтелектуальних систем на базі нейронних мереж за допомогою програмного засобу MATLAB

Теоретичні відомості

Нейронні мережі (NN – Neural Networks) широко використовуються для вирішення різноманітних завдань. Серед сфер застосування NN, що розвиваються, – обробка аналогових і цифрових сигналів, синтез і ідентифікація електронних ланцюгів і систем. Основи

теорії і технології застосування NN широко представлені в пакеті MATLAB.

Графічний інтерфейс користувача NNTool дозволяє вибирати структури NN з великого переліку і надає безліч алгоритмів навчання для кожного типу мережі.

Елементи керування NNTool

Щоб запустити NNTool, необхідно виконати однойменну команду в командному вікні MATLAB :

>> nntool

після цього з'явиться головне вікно NNTool, що називається «Вікном управління мережами і даними» (рис. 30).

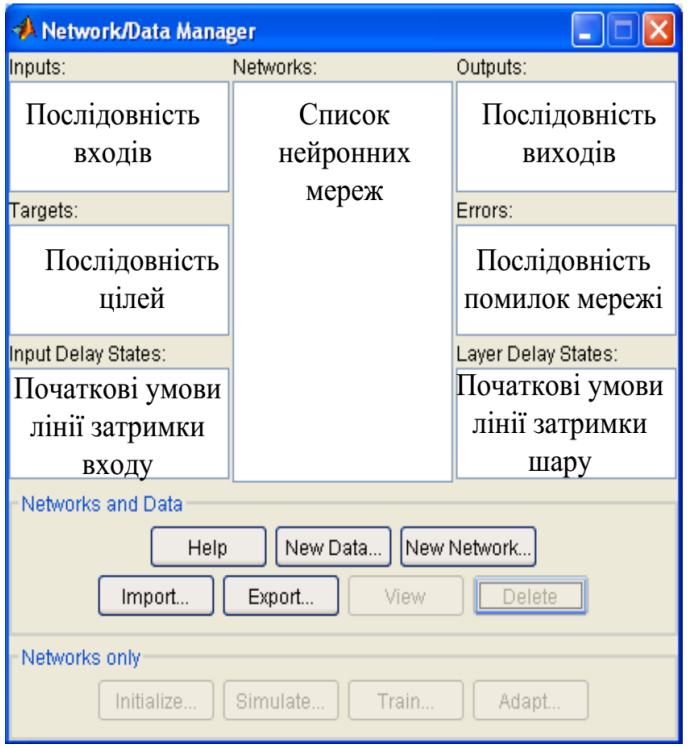


Рис. 30. Головне вікно NNTool

Панель «Мережі і дані» (Networks and Data) має функціональні клавіші з наступними призначеннями:

– Допомога (Help) – короткий опис елементів цього вікна, що управляють;

– Нові дані (New Data.) – виклик вікна, що дозволяє створювати нові набори даних;

– Нова мережа (New Network.) – виклик вікна створення нової мережі;

– Імпорт (Import.) – імпорт даних з робочого простору MATLAB в простір змінних NNTool;

– Експорт (Export.) – експорт даних з простору змінних NNTool в робочий простір MATLAB;

– Вид (View) – графічне відображення архітектури вибраної мережі;

– Видалити (Delete) – видалення вибраного об'єкту

Імпорт-експорт даних. На практиці часто доводиться переносити дані з одного комп'ютера на інший або користуватися зовнішніми, по відношенню до NNTool, засобами обробки даних. У зв'язку з цим виникає необхідність збереження результатів роботи і завантаження даних. Не менш важливий обмін даними між NNTool і MATLAB, оскільки простори їх змінних не перетинаються.

Ці завдання вирішують засоби імпорту-експорту і завантаження-збереження даних. Доступ до них здійснюється через головне вікно NNTool за допомогою кнопок Import і Export

Імпорт

Джерелом служить змінна в робочому просторі MATLAB, а пунктом призначення – змінна в робочому просторі NNTool. Натиснувши кнопку Import, потрапляємо у вікно «Імпорту-завантаження даних» (Import or Load to Network/Data Manager). По замовчуванню тут встановлена опція завантаження з робочого простору MATLAB, тому в центрі вікна з'являється список тих, що належать йому змінних. Вибравши мишкою потрібну змінну в полі «Вибір змінної» (Select a Variable), – їй можна задати довільне ім'я в полі «Ім'я» (Name), під яким вона буде скопійована в NNTool. Коли змінна виділена, NNTool аналізує її тип і робить доступними для вибору ті «Категорії даних» (Import As), які підтримуються. Вказавши одну з них, слід натиснути кнопку «Імпортувати» (Import), щоб завершити копіювання. Після того, як усі дії успішно проведені, ім'я змінної, що імпортується, з'явиться в одному із списків головного вікна NNTool.

Завантаження з файлу. Тут джерело – файл. При цьому важливо, щоб його формат підтримувався NNTool. У відповідному форматі зберігаються так звані MAT-файли. Вони містять бінарні дані і дозволяють MATLAB зберігати змінні будь-яких підтримуваних типів і розмірності. Такі файли можуть створюватися, наприклад, в процесі роботи з NNTool. Щоб завантажити змінні з MAT-файла в NNTool, необхідно відкрити вікно імпорту, натиснувши кнопку «Import» головного вікна NNTool. Потім слід зазначити опцію «Завантажити з файлу» (Load from disk file) і, натиснувши «Огляд» (Browse), відкрити файл з даними, що зберігаються у форматі MAT – файлов. Найчастіше, це файли з розширенням MAT. В результаті, список у вікні імпорту заповниться іменами змінних, збережених у вказаному MAT – файлі. Подальша послідовність дій повністю співпадає з описаною в пункті Імпорт.

Експорт. Ця функція копіює задані змінні з NNTool в робочий простір MATLAB. Вона доступна по натисненню кнопки Export головного вікна NNTool. У вікні експорту єдиним списком перераховані змінні усіх категорій, представлених в NNTool. Тут необхідно виділити ті з них, які підлягають експорту, і натиснути Export. Тепер виділені змінні скопійовані в робочий простір MATLAB.

Збереження у файл

В цілому, процедура схожа з експортом, за одним виключенням: відмітивши змінні, слід натиснути кнопку «Зберегти» (Save).

Тоді з'явиться вікно, в якому можна задати ім'я файлу. Його можна вказати без розширення

– по замовчуванню NNTool прикріпить розширення MAT. Це пов'язано з тим, що NNTool зберігає дані тільки у форматі MAT -файлів.

Розглянемо створення нейронної мережі за допомогою NNTool на прикладі.

Приклад 1. Створення НМ, що виконує логічну функцію «І».

1. Створення мережі. Виберемо мережу, що складається з одного персептрона з двома входами. В процесі навчання мережі на її входи подаються вхідні дані і проводиться зіставлення значення, отриманого на виході, з цільовим (бажаним). На підставі результату порівняння (відхилення набутого значення від бажаного) обчислюються величини зміни вагів і зміщення, що зменшують це відхилення.

Отже, перед створенням мережі необхідно заготовити набір повчальних і цільових даних. Складемо таблицю істинності для логічної функції «І», де P1 і P2 – входи, а A – бажаний вихід (таб. 10).

Таблиця 10

Таблиця істинності логічної функції «І»

P1	P2	A
0	0	0
0	1	0
1	0	0
1	1	1

Щоб задати матрицю, що складається з чотирьох векторів-рядків, як вхідну, скористаємося кнопкою New Data. У вікні, що з'явилося, слід провести зміни, показані на рис. 31, і натиснути клавішу «Створити» (Create).

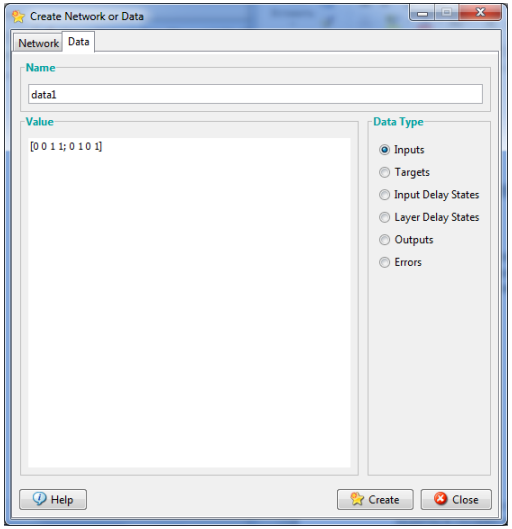


Рис. 31. Завдання вхідних векторів

Після цього у вікні управління з'явиться вектор data1 в розділі Inputs. Вектор цілей задається схожим чином (рис. 32).

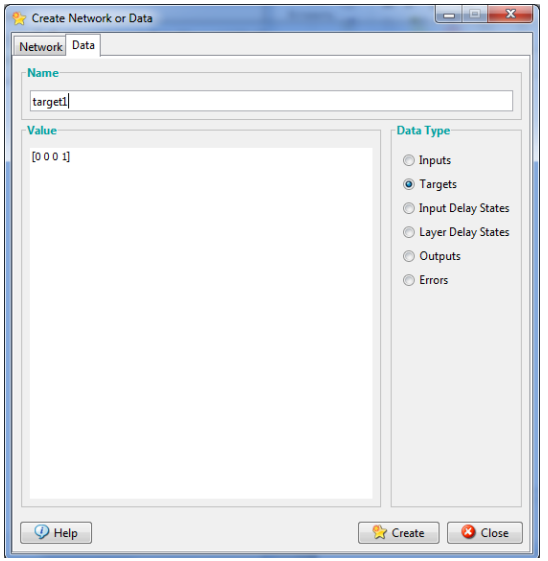


Рис. 32. Завдання цільового вектору

Після натиснення на Create в розділі Targets з'явиться вектор target1.

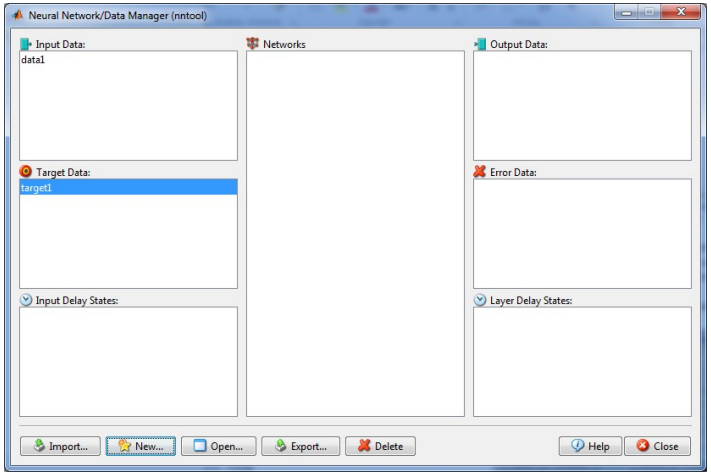


Рис. 33. Вектор target1

Дані в полі «Значення» (Value) можуть бути представлені будь-яким зрозумілим MATLAB виразом. Приміром, попереднє визначення вектору цілей можна еквівалентно замінити рядком виду bitand([0 0 1 1], [0 1 0 1]).

Тепер слід приступити до створення нейронної мережі. Вибираємо кнопку New Network і заповнюємо форму, як показано на рис. 34.

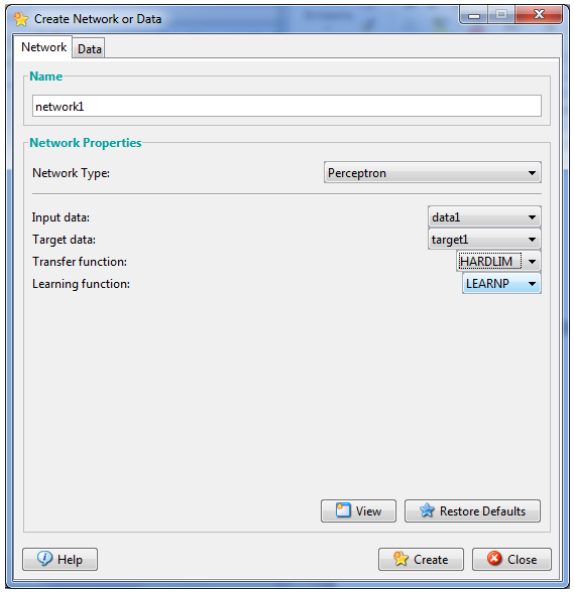


Рис. 34. Вікно «Створення мережі»

При цьому поля несуть наступні смислові навантаження:

– Ім'я мережі (Network Name) – це ім'я об'єкту створюваної мережі.

– Тип мережі (Network Type) – определяеттип мережі і в контексті вибраного типу представляє для введення різні параметри в частині вікна, розташованій нижче за цей пункт. Таким чином, для різних типів мереж вікно змінює свій зміст.

– Вхідні діапазони (Input ranges) – матриця з числом рядків, рівним числу входів мережі. Кожен рядок є вектором з двома елементами: перший – мінімальне значення сигналу, яке буде подано на відповідний вхід мережі при навчанні, другої, – максимальне. Для спрощення

введення цих значень передбачений випадний список «Отримати з входу» (Get from input), що дозволяє автоматично сформувати необхідні дані, вказавши ім'я вхідної змінної.

– Кількість нейронів (Number of neurons) – число нейронів в шарі.

– Передавальна функція (Transfer function) – в цьому пункті вибирається передавальна функція (функція активації) нейронів.

– Функція навчання (Learning function) – функція, що відповідає за оновлення вагів і зміщень мережі в процесі навчання.

За допомогою клавіші «Вигляд» (View) можна подивитися архітектуру створюваної мережі (рис.28). Так, ми маємо можливість упевнитися, чи усі дії були виконані вірно. На рис. 35 зображена персептронная мережа з вихідним блоком, що реалізовує передавальну функцію з жорстким обмеженням. Кількість нейронів в шарі дорівнює одному, що символічно відображується розмірністю вектору-стовпця на виході шару і вказується числом безпосередньо під блоком передавальної функції. Дана мережа має два входи, оскільки розмірність вхідного вектору-стовпця рівна двом.

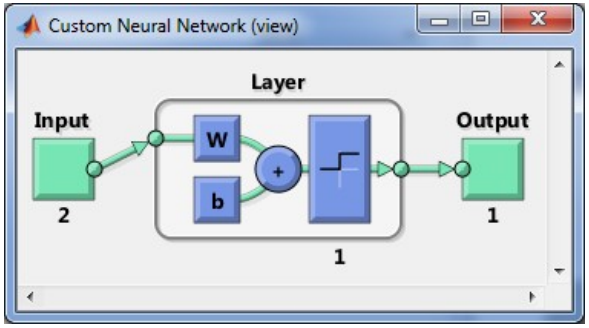


Рис. 35. Попередній перегляд створюваної мережі

Отже, структура мережі відповідає нашому завданню. Тепер можна закрити вікно попереднього перегляду, натиснувши клавішу «Закрити» (Close), і підтвердити намір створити мережу, натиснувши «Створити» (Create) у вікні створення мережі.

В результаті виконаних операцій в розділі «Мережі» (Networks) головного вікна NNTool з'явиться об'єкт з ім'ям network1.

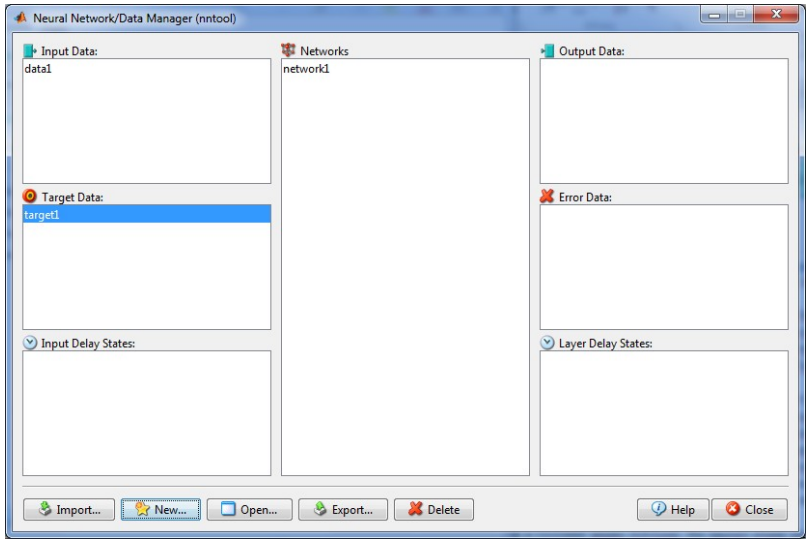


Рис. 36. Об'єкт з ім'ям network1

2. Навчання. Наша мета – побудувати нейронну мережу, яка виконує функцію логічного «І». Очевидно, не можна розраховувати на те, що відразу після етапу створення мережі остання забезпечуватиме правильний результат (правильне співвідношення «вхід/вихід»). Для досягнення мети мережунеобхідно належним чином навчити, тобто підібрати відповідні значення параметрів. У MATLAB реалізована більшість відомих алгоритмів навчання нейронних мереж, серед яких представлено два для персептронных мереж даного виду. Створюючи мережу, ми вказали LEARNP як функції, що реалізовує алгоритм навчання (рис. 34).

Повернемося в головне вікно NNTool. На цьому етапі інтерес представляє нижня панель «Тільки мережі» (Networks only). Натиснення будь-якої з клавіш на цій панелі викличе вікно, на безлічі вкладок якого представлені параметри мережі, необхідні для її навчання і прогону, а також мережі, що відбивають поточний стан.

Відмітивши покажчиком миші об'єкт мережі network1, викличемо вікно управління мережею натисненням кнопки Train. Перед нами виникне вкладка «Train» вікна властивостей мережі, що містить, у свою чергу, ще одну панель вкладок (рис. 37). Їх головне призначення –

управління процесом навчання. На вкладці «Інформація навчання» (Training info) вимагається вказати набір повчальних даних в полі «Входи» (Inputs) і набір цільових даних в поле «Цілі» (Targets). Поля «Виходи» (Outputs) і «Помилки» (Errors) NNTool заповнює автоматично. При цьому результати навчання, до яких відносяться виходи і помилки, зберігатимуться в змінних з вказаними іменами

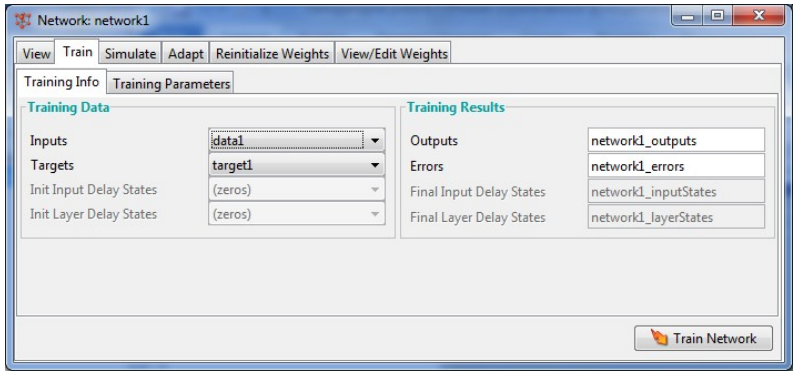


Рис. 37. Вікно параметрів мережі, відкрите на вкладці «навчання» (Train)

Завершити процес навчання можна, керуючись різними критеріями. Можливі ситуації, коли проводять зупинку навчання, вважаючи достатнім деякий інтервал часу. З іншого боку, об'єктивним критерієм є рівень помилки.

На вкладці «Параметри навчання» (Training parameters) для нашої мережі (рис. 38) можна встановити наступні поля:

– Кількість епох (epochs) – визначає число епох (інтервал часу), після яких навчання буде припинено. Эпохою називають одноразове представление усіх повчальних вхідних даних на входи мережі.

– Досягнення цілі або попадання (goal) – тут задається абсолютна величина функції помилки, при якій ціль вважатиметься досягнутою.

– Період оновлення (show) – період оновлення графіку кривою навчання, виражений числом епох.

– Час навчання (time) – після закінчення вказаного тут тимчасового інтервалу, вираженого в секундах, навчання припиняється.

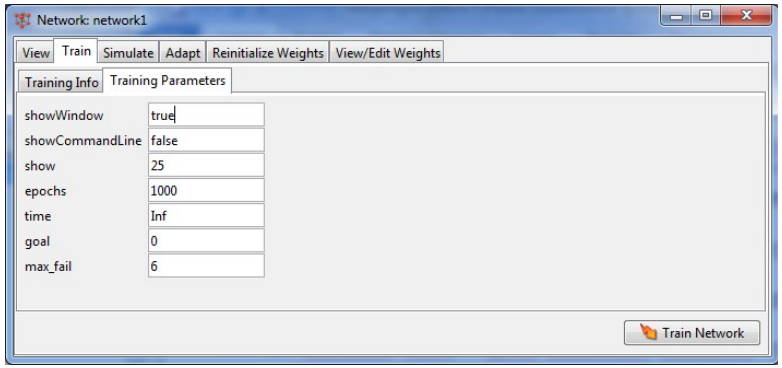


Рис. 38. Вкладка параметрів навчання

Зважаючи на той факт, що для завдань з лінійно віддільними множинами (а наше завдання відноситься до цього класу) завжди існує точне рішення, встановимо поріг досягнення мети, рівний нулю. Значення інших параметрів залишимо по замовчуванню. Помітимо тільки, що поле часу навчання містить запис Inf, який визначає нескінченний інтервал часу (від англійського Infinite – нескінченний).

Розглянемо вкладку навчання (Train). Щоб почати навчання, треба натиснути кнопку «Навчити мережу» (Train Network). Після цього, якщо у нинішній момент мережа не задовольняє жодній з умов, вказаних в розділі параметрів навчання (Training Parameters), з'явиться вікно, що ілюструє динаміку цільової функції – криву навчання. У нашому випадку графік може виглядати так, як показано на рис. 39. Кнопкою «Зупинити навчання» (Stop Training) можна припинити цей процес. З рис. 39 видно, що навчання було зупинене, коли функція мети досягла встановленої величини (goal = 0).

Слід зазначити, що для персептронів, що мають функцію активації з жорстким обмеженням, помилка розраховується як різниця між ціллю і отриманим виходом.

Отже, алгоритм навчання знайшов точне рішення задачі. У методичних цілях переконаємося в правильності рішення задачі шляхом прогону навченої мережі. Для цього необхідно відкрити вкладку «Прогін» (Simulate) і вибрати у випадному списку «Входи» (Inputs) заготовлені

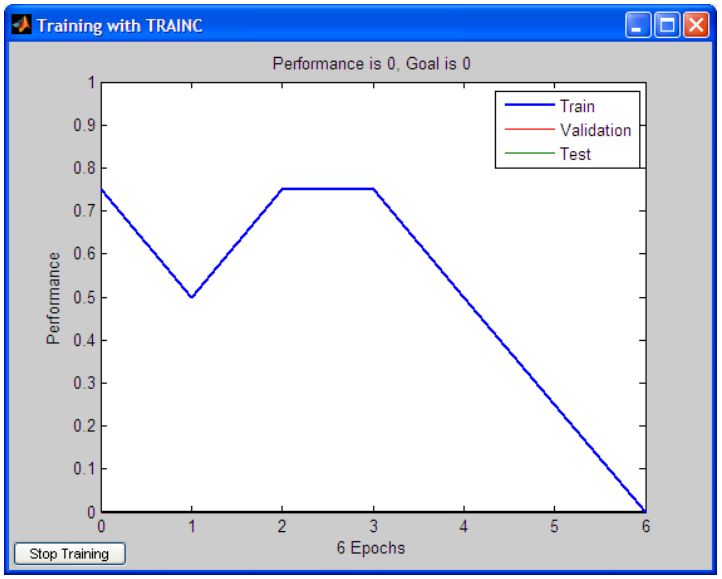


Рис. 39. Крива навчання

дані. У цьому завданні природно використовувати той же набір даних, що і при навчанні data1. За бажання можна встановити прапорець «Задати цілі» (Supply Targets). Тоді в результаті прогону додатково будуть розраховані значення помилки. Натиснення кнопки «Прогін мережі» (Simulate Network) запише результати прогону в змінну, ім'я якої вказане в полі «Виходи» (Outputs). Тепер можна повернутися в основне вікно NNTool і, виділивши вихідну змінну network1, натиснути кнопку «Перегляд» (View). Вміст вікна перегляду співпадає зі значенням вектору цілей – мережа працює правильно.

Слід зауважити, що мережа створюється такою, що ініціалізувала, тобто значення вагів і зміщень задаються певним чином. Перед кожним наступним досвідом навчання зазвичай початкові умови оновлюються, для чого на вкладці «Ініціалізація» (Initialize) передбачена функція ініціалізації. Так, якщо вимагається провести декілька незалежних дослідів навчання, ініціалізація вагів і зміщень перед кожним з них здійснюється натисненням кнопки «Ініціалізувати ваги» (Initialize Weights).

Повернемося до вкладки «додаткова інформація» (Optional info). Щоб зрозуміти, якій меті служать представлені тут параметри, необхідно обговорити два поняття: перенавчання та узагальнення.

При виборі нейронної мережі для вирішення конкретного завдання важко передбачити її порядок. Якщо вибрати невиправдано великий порядок, мережа може виявитися занадто гнучкою і може представити просту залежність складним чином. Це явище називається перенавчанням. У разі мережі з недостатньою кількістю нейронів необхідний рівень помилки ніколи не буде досягнутий. Тут в наявності черезмірне узагальнення.

Для попередження перенавчання застосовується наступна техніка. Дані діляться на дві множини: навчальну (Training Data) та контрольну (Validation Data). Контрольна множина в навчанні не використовується. На початку роботи помилки мережі на навчальній і контрольній множинах будуть однаковими. У міру того, як мережа навчається, помилка навчання зменшується, і, поки навчання зменшує дійсну функцію помилки, помилка на контрольній множині також зменшуватиметься. Якщо ж контрольна помилка перестала зменшуватися, або навіть стала рости, це вказує на те, що навчання слід закінчити. Зупинка на цьому етапі називається ранньою зупинкою (Early stopping).

Таким чином, необхідно провести серію експериментів з різними мережами, перш ніж буде отримана відповідна. При цьому щоб не бути введеним в оману локальними мінімумами функції помилки, слід кілька разів навчати кожну мережу.

Якщо в результаті послідовних кроків навчання і контролю помилка залишається неприпустимо великою, доцільно змінити модель нейронної мережі (наприклад, ускладнити мережу, збільшивши число нейронів, або використовувати мережу іншого виду). У такій ситуації рекомендується застосовувати ще одну множину – тестову безліч спостережень (Test Data), яка є незалежною вибіркою з вхідних даних. Підсумкова модель тестується на цій множині, що дає додаткову можливість переконатися в достовірності отриманих результатів. Очевидно, щоб зіграти свою роль, тестова множина має бути використана тільки один раз. Якщо його використовувати для коригування мережі, воно фактично перетвориться на контрольну множину.

Установка верхнього прапорця дозволить задати контрольну множину і відповідний вектор цілей (можливо, той же, що при навчанні).

Установка нижнього дозволяє задати тестову множину і вектор цілей для нього.

Навчання мережі можна проводити в різних режимах. У зв'язку з цим, в NNTool передбачено дві вкладки, що представляють повчальні функції, і розглянута раніше вкладка Train і «Адаптація» (Adapt). Adapt вміщує вкладку інформація адаптації (Adap – tion Info), на якій містяться поля, схожі по своєму призначенню з полями вкладки Training Info і що виконують ті ж функції і вкладку параметри адаптації (Adaption Parameters). Остання містить єдине поле «Проходи» (passes). Значення, вказане в цьому полі, визначає, скільки разів усі вхідні вектори будуть представлені мережі в процесі навчання.

Параметри вкладок «Train» та «Adapt» в MATLAB використовуються функціями train і adapt, відповідно.

Розподіл лінійно-невід'ємних великих кількостей. Розглянуте завдання синтезу логічного елементу «І» може трактуватися як завдання розпізнавання лінійно віддільних великих кількостей. На практиці ж частіше зустрічаються завдання розподілу лінійно невід'ємних великих кількостей, коли застосування персептронів з функцією активації з жорстким обмеженням не дасть рішення. У таких випадках слід використовувати інші функції активації.

Показовим прикладом лінійно невід'ємного завдання являється створення нейронної мережі, що виконує логічну функцію що «виключає АБО».

Приклад 2. Створення НМ, що виконує логічну функцію «виключає АБО».

Розглянемо таблицю істинності цієї функції (таб. 11).

Таблиця 11

Таблиця істинності логічної функції що «виключає АБО»

P1	P2	A
0	0	0
0	1	1
1	0	1
1	1	0

Що ж мається на увазі під «лінійною невід'ємністю» великих кількостей? Щоб відповісти на це питання, зображуватимемо безліч вихідних значень в просторі входів (рис. 40), дотримуючись наступного правила: поєднання входів P1 і P2, при яких вихід A перетворюється на нуль, позначаються колом, а ті, при яких A звертається в одиницю – хрестиком

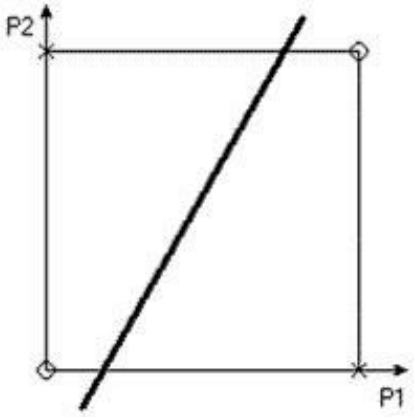


Рис. 40. Стани логічного елементу що «виключає АБО»

Наша мета – провести межу, що відділяє безліч нулів від безлічі хрестиків. З побудованої картини на рис. 40 видно, що неможливо провести пряму лінію, яка б відокремила нулі від одиниць. Саме у цьому сенсі безліч нулів лінійна невіддільно від безлічі одиниць, і персептрони, розглянуті раніше, в принципі, не можуть вирішити дане завдання

Якщо ж використовувати персептрони із спеціальними нелінійними функціями активації, наприклад, сигмоїдними, то рішення задачі можливе. Виберемо персептрон з двома нейронами прихованого шару, у яких функції активації сигмоїдні, і одним вихідним нейроном з лінійною функцією активації. Функцією помилки вкажемо MSE (Mean Square Error – середній квадрат помилки). Нагадаємо, що функція помилки встановлюється у вікні «Створення мережі» після вибору типу мережі

Ініціалізувати мережу, натиснувши кнопку Initialize Weights на вкладці Initialize, після чого навчимо, вказавши як вхідні значення сформовану раніше змінну data1, як цілі – новий вектор, відповідний бажаним виходам. В процесі навчання мережа не може забезпечити точного рішення, тобто звести помилку до нуля.

Слід зазначити, що ця крива може міняти свою форму від експерименту до експерименту, але, у разі успішного навчання, характер функції буде таким, що монотонно убуває. В результаті навчання, помилка була мінімізована до дуже малого значення, яке практично можна вважати рівним нулю. Завдання синтезу елементу що «виключає АБО» є також прикладом завдання класифікації.

Приклад 3. Апроксимація функції.

Використання GUI-інтерфейсу пакета нейронних мереж

Необхідно виконати апроксимацію функції наступного виду:

$$\sin\left(\left(\frac{5\pi x}{N}\right)+\sin\left(\frac{7\pi x}{N}\right)\right)$$

де x змінюється від 1 до N (N – число точок функції)

Заготовимо цільові дані, ввівши в поле «Значення» (Value) вікна створення нових даних вираз:

sin(5*pi*[1:100]/100+sin(7*pi*[1:100]/100)).

Ця крива є відрізком періодичного коливання з частотою $5\pi/N$, що модулюється по фазі гармонійним коливанням з частотою $7\pi/N$ (рис.33).

Тепер заготовимо набір навчальних даних (1, 2, 3, ., 100), задавши їх наступним виразом:

1:100.

Виберемо персептрон (Feed – Forward Back Propagation) с тринадцятьма сигмоїдними (TANSIG) нейронами прихованого шару і одним лінійним (PURELIN) нейроном вихідного шару. Навчання проводитимемо, використовуючи алгоритм Левенберга-Маркардта (Levenberg – Mar – quardt), який реалізує функція TRAINLM. Функція помилки – MSE.

Тепер можна приступити до навчання. Для цього необхідно вказати, які набори даних мають бути використані як повчальні і цільові, а потім провести навчання.

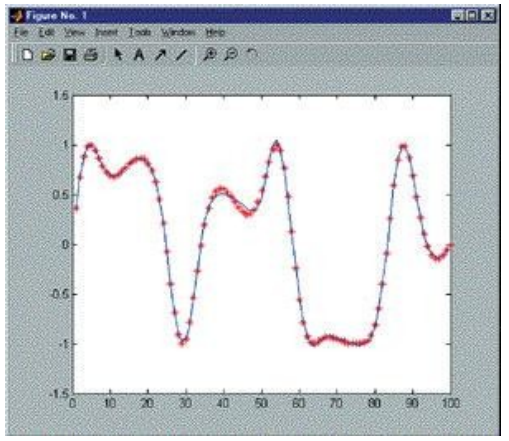


Рис. 41. Крива з зірочками– цільові дані, інша крива – апроксимуюча функція

Рис. 41 ілюструє різницю між цільовими даними і отриманою апроксимуючою кривою.

Робота з нейронною мережею в командному режимі

Створіть узагальненорегресійну нейронну мережу (НМ) з ім'ям *a* для апроксимації функції вигляду $y=x^2$ на відрізку [-1, 1], використовуючи наступні експериментальні дані:

x = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1],

y = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1].

Процедура створення і використання даної НМ описується таким чином:

```
% Задання вхідних значень
P = [-1 -0.8 -0.5 -0.2 0 0.1 0.3 0.6 0.9 1];% Задання вихідних значень
T = [1 0.64 0.25 0.04 0 0.01 0.09 0.36 0.81 1];
% Створення узагальнено-регресійної НМ з відхиленням 0.01
a = newgrnn (P, T, 0.01);
% Опитування НМY =
(a, [-0.9 -0.7 -0.3 0.4 0.8])
```

2. Визначте точність апроксимації з отриманих даних. Спробуйте поліпшити якість апроксимації, реалізувавши мережу з радіальними базисними елементами, використовуючи функцію newrb.

3. Порівняйте роботу різних мереж для завдань апроксимації.

4. Запустіть графічний редактор NNTool та створіть для тих самих вхідних даних та цільової функції нейронну радіально-базисну мережу з нульовою помилкою (Radial basis (exact fit)) та радіально-базисну мережу з мінімальним числом нейронів (Radial basis (fewer neurons)).

5. Виконавши експорт (Export…) розроблених мереж до робочої області, перевірте правильність їх роботи, задавши в командному рядку MATLAB дані для опитування.

6. Порівняйте результати роботи мереж, розроблених програмно та за допомогою графічного інтерфейсу NNTool.

7. Отримайте графіки цільових функцій та результати, що видала мережа. Зробіть порівняльний аналіз усіх отриманих графіків.

8. Побудуйте лінійну нейронну мережу, що дозволяє прогнозувати майбутнє значення сигналу (функції часу), що описується співвідношенням x(t)=cos(4πt), який піддається дискретизації з інтервалом 0,01с. Використовувати нижченаведений лістинг.

```
t=0:0.01:5
x=cos(t*4*pi)
plot(t,x)
q=length(x)
p=zeros(5,q)
p(1,2:q)=x(1,1:(q-1))
```



```
p(2,3:q)=x(1,1:(q-2))
p(3,4:q)=x(1,1:(q-3))
p(4,5:q)=x(1,1:(q-4))
p(5,6:q)=x(1,1:(q-5))
s=newlind(p,x)
y=sim(s,p)
plot(t,y,t,x,'*')
e=x-y
plot(t,e)
```

9. Наведіть отримані графіки для заданої функції.

10. Відобразіть графік отриманої помилки прогнозу.

11. Оцініть точність прогнозу і зробіть висновки про роботу НМ.

Завдання до лабораторної роботи 9

1. Побудувати НМ для двох обраних логічних операцій.
2. Провести апроксимацію функції $y = 2x^n$, де n-номер варіанту.

Контрольні запитання

1. Які основні задачі вирішуються за допомогою нейронни мереж?
2. Для чого використовується NNTool?
3. Назвіть послідовність операцій, які потрібно виконати прии навчанні нейронної мережі.
4. Дайте визначення поняттям «нейронна мережа», «персептрон».

4.5.2. Лабораторна робота 10. Кластеризація

Мета: освоїти основні принципи розв'язку задачі кластеризації з використанням нейронних мереж з шаром Кохонена і самоорганізованих карт.

Завдання: Використовуючи вбудовані функції пакета нейронних мереж математичної середовища MATLAB, побудувати нейронну мережу з шаром Кохонена, яка безліч множину даних розділить на кластери і виявить їх центри. На навчену мережу подати новий вхідний вектор і визначити, до якого кластеру він відноситься.

Теоретичні відомості

Самоорганізовані карти. Самоорганізовані карти (Self Organizing Maps – SOM) це одна з різновидів нейромережевих алгоритмів. Основною відмінністю даної технології від розглянутих нами раніше нейромереж,

яких навчають за алгоритмом зворотного поширення, є те, що при навчанні використовується метод навчання без вчителя, тобто результат навчання залежить тільки від структури вхідних даних. Нейронні мережі даного типу часто застосовуються для вирішення найрізноманітніших задач, від відновлення пропусків у даних до аналізу даних і пошуку закономірностей.

Основи самоорганізованих карт. Алгоритм функціонування самонавчаємих карт є один з варіантів кластеризації багатовимірних векторів. Прикладом таких алгоритмів може служити алгоритм найближчих середніх. Важливою відмінністю алгоритму SOM є те, що в ньому всі нейрони (вузли, центри класів) впорядковані в деяку структуру (зазвичай двовимірну сітку). При цьому в ході навчання модифікується не тільки нейрон-переможець, а й його сусіди, але в меншому ступені. За рахунок цього SOM можна вважати одним з методів проектування багатовимірного простору в простір з більш низькою розмірністю При використанні цього алгоритму вектора, схожі в вихідному просторі, виявляються поруч і на отриманій карті.

Структура самоорганізованих карт. SOM має на увазі використання впорядкованої структури нейронів. Зазвичай використовуються одно- і двовимірні сітки. При цьому кожен нейрон являє собою n-мірний вектор-стовпець $w = [w_1, w_2, ..., w_n]$, де n визначається розмірністю вихідного простору. Застосування одне і двовимірних сіток пов'язано з тим, що виникають проблеми при відображенні просторових структур більшої розмірності (при цьому знову виникають проблеми з пониженням розмірності до двовимірної).

Зазвичай нейрони розташовуються у вузлах двовимірної сітки з прямокутними або шестикутними осередками. При цьому, як було сказано вище, нейрони також взаємодіють один з одним. Величина цьої взаємодії визначається відстанню між нейронами на карті.

Початкова ініціалізація карти. При реалізації алгоритму SOM заздалегідь задається конфігурація сітки (прямокутна або шестикутна), а також кількість нейронів в мережі. Деякі джерела рекомендують використовувати максимально можливу кількість нейронів в карті. При цьому початковий радіус навчання (neighborhood) в значній мірі впливає на здатність узагальнення за допомогою, отриманою карти. У разі, коли кількість вузлів карти перевищує кількість прикладів в навчальній вибірці, то успіх використання алгоритму в великій мірі залежить від

відповідного вибору початкового радіуса навчання. Однак в разі, коли розмір карти становить десятки тисяч нейронів, час, необхідний на навчання карти, зазвичай буває занадто велике для вирішення практичних завдань, таким чином, необхідно досягати допустимого компромісу при виборі кількості вузлів.

Перед початком навчання карти необхідно ініціалізувати вагові коефіцієнти нейронів. Вдало обраний спосіб ініціалізації може істотно прискорити навчання і привести до отримання більш якісних результатів. Існують три способи ініціалізації початкових ваг:

1) ініціалізація випадковими значеннями, коли всім вагам даються малі випадкові величини;

2) ініціалізація прикладами, коли в якості початкових значень задаються значення випадково вибраних прикладів з навчальної вибірки;

3) лінійна ініціалізація. В цьому випадку ваги ініціюються значеннями векторів, лінійно впорядкованих вздовж лінійного підпростору, що проходить між двома головними власними векторами вихідного набору даних.

Навчання самоорганізованих карт. Навчання складається з послідовності корекцій векторів, що представляють собою нейрони. На кожному кроці навчання з вихідного набору даних випадково вибирається один з векторів, а потім проводиться пошук найбільш схожого на нього вектора коефіцієнтів нейронів. При цьому обирається нейрон-переможець, який найбільш схожий на вектор входів. Під схожістю в даній задачі розуміється відстань між векторами, зазвичай обчислюється в евклідовому просторі. Після того, як знайдений нейрон-переможець, здійснюється коректування ваг нейромережі. При цьому вектор, що описує нейрон-переможець, і вектори, що описують його сусідів у сітці, переміщаються в напрямку вхідного вектора.

Задача. Використовуючи вбудовані функції пакета нейронних мереж математичного середовища MATLAB, побудувати нейронну мережу з шаром Кохонена, яка множину вхідних даних розділить на кластери і виявить їх центри. На навчену мережу подати новий вхідний вектор і визначити, до якого кластеру він відноситься.

Розв'язок

Для створення нейронної мережі з шаром Кохонена скористаємося вбудованою в середу MATLAB функцією newc:

Newc – створення конкурентного шару.

Net = newc (PR, S, KLR, CLR) – функція створення шару Кохонена.

Аргументи функції:

PR – матриця мінімальних і максимальних значень для R вхідних елементів,

S – число нейронів,

KLR – коефіцієнт навчання Кохонена (за замовчуванням 0,01)

CLR – Коефіцієнт «справедливості» (за замовчуванням 0,001)

Задача 1.1.

```
X = [0 1; 0 1];
clusters = 5;
points = 5;% Задання кількості точок в кластері
std_dev = 0.01; % Задання стандартного відхилення
p = nngenc (X, clusters, points, std_dev);% Моделювання вхідних даних
h = newc ([0 1; 0 1], 5, .1);% створення шару Кохонена
h.trainParam.epochs = 50; % Задання кількості циклів навчання
h = init (h);
h = train (h, p);
w = h.IW {1}; % Висновок графіків вихідних даних і виявлених центрів кластерів
plot (p (1, :), p (2,:), '^ r'), grid;
hold on;
plot (w (:, 1), w (:, 2), 'ob');
xlabel ( 'p (1)');
ylabel ( 'p (2)');
A = 0.6
B = 0.5
p = [A; B];
plot (A, B, '^ k')
y = sim (h, p)% Опрос мережі
A = 0.6000
B = 0.5000
y = (3,1) 1
```

Пред'явлений вектор віднесений до 3-го кластеру.

Задача 1.2.

Тепер даний алгоритм застосуємо до реальної задачі кластеризації. На вхід нейронної мережі будемо подавати дані вагозрістових показників людей і спробуємо виявити три класи (кластера):

1) нормальний вагозрістовий показник;
2) надлишок ваги;
3) недолік ваги.

Вхідні дані (перший рядок матриці – зріст; друга – вага)

```
p = [175 180 182 175 183 176 183 176 183 176 175 180 178 180 178 182 178 182 179 174 172 179;
70 75 100 99 42 48 76 72 40 45 92 96 70 69 95 90 79 82 80 50 96 91]% створюємо НС Кохонена з 3 кластерами (нормальний вагозрістовий показник, надлишок ваги і недолік ваги)
h = newc ([0 200, 0 100], 3, .1)
h.trainParam.epochs = 500;% Задання кількості циклів навчання
h = train (h, p)
w = h.IW {1};
plot (p (1, :), p (2,:), '^ r');
hold on;
plot (w (:, 1), w (:, 2), 'ob');
xlabel ( 'Rost');
ylabel ( 'Ves');
% Задання нового вхідного вектора
% Опитування мережі
A = 181
B = 65
p = [A; B];
plot (A, B, '+ r')
y = sim (h, p)
A = 181
B = 65
y = (3,1) 1
```

Пред'явлений вектор віднесений до 3-го кластеру.

Задача 1.3.

Тепер розглянемо використання систем, що самоорганізує карти на прикладі двовимірних векторів. Використовуючи самоорганізовуючи карти, двовимірні вектори розбити на кластери і виявити їх центри, потім подати на вхід карти новий вектор і визначити кластер, до якого він належить.

```
P = rands (2,100)% Задання випадкових двомірних вхідних векторів
figure (1)
hold on
plot (P (1, :), P (2,:), '+ r')% візуальне зображення вхідних векторів
% Створення НС з 3 * 4 нейронами
% За замовчуванням функція TFCN = 'hextop', тобто нейрони розташовуються у вузлах двовимірної сітки з шестикутними осередками
net = newsom ([0 1; 0 1], [3 4]);
net.trainParam.epoch = 1 % Задання числа циклів налаштування
net = train (net, P) % налаштування мережі
A = 0.5
B = 0.3
p = [A; B]; % Задання нового вхідного вектора
plot (A, B, '^ k')% промальовування на рисунку вхідного вектора (чорний трикутник)
figure (2)
plotsom (net.iw {1,1}, net.layers {1} .distances)
a = sim (net, p)%       опитування мережі
A = 0.5000
B = 0.3000
a = (9,1) 1
```

Предявлений вектор віднесений до 9-го кластеру.

Контрольні запитання

1. Що таке SOM?
2. В чому різниця між класифікацією та кластеризацією?
3. Назвіть особливості мережі Кохонена.
4. Як може задаватися ініціалізація ваг.

4.5.3. Лабораторна робота 11. Принципи роботи нейронних мереж у складі систем керування

Мета: дослідити систему керування з нейромережевим регулятором на основі еталонної моделі.

Теоретичні відомості

Пакет *Neural Network Toolbox* (нейронні мережі) містить засоби для проектування, моделювання, навчання та використання відомих парадигм апарата штучних нейронних мереж. Пакет може використовуватися при розв'язанні різноманітних задач, таких як обробка сигналів, нелінійне керування, фінансове моделювання та ін.

У пакеті Neural Network Toolbox реалізовані 3 архітектури нейронних мереж у вигляді наступних контролерів:

– контролер із прогнозом (*NN Predictive Contrloller*);

– контролер на основі моделі авторегресії з ковзним середнім (*NARMAL2 Controller*);

– контролер на основі еталонної моделі (*Model Reference Controller*).

Застосування нейронних мереж для розв'язання завдань керування дозволяє виділити два етапи проектування:

– етап ідентифікації;

– етап синтезу закону керування.

На етапі ідентифікації розробляється модель керованого процесу у вигляді нейронної мережі, що на етапі синтезу використовується для синтезу регулятора. Для кожної з трьох архітектур використовується одна й та сама процедура ідентифікації, однак етапи синтезу істотно різняться. При керуванні з прогнозом модель керованого процесу використовується для того, щоб пророчити його майбутню поведінку, а алгоритм оптимізації застосовується для розрахунку такого керування, що мінімізує різницю між бажаними й дійсними змінами виходу моделі.

При керуванні на основі моделі авторегресії з ковзаним середнім регулятор являє собою досить просту реконструкцію моделі керованого процесу.

При керуванні на основі еталонної моделі регулятор – це нейронна мережа, що навчена керувати процесом так, щоб він відслідковував поведінку еталонного процесу. При цьому модель керованого процесу активно використовується при настроюванні параметрів самого регулятора.

Динамічні моделі систем керування із нейромережевими регуляторами розміщені в спеціальному розділі *Control System* набору блоків *NN Blocksets* і включають три згадані вище моделі регуляторів, а також блок побудови графіків.

Оскільки жоден конкретний регулятор не є універсальним, то описані можливості всіх трьох типів регуляторів, кожний з яких має свої переваги й недоліки.

Регулятор з прогнозом. Цей регулятор використовує модель керованого процесу у вигляді нейронної мережі, для того щоб прогнозувати майбутні реакції процесу на випадкові сигнали керування. Алгоритм оптимізації обчислює керуючі сигнали, які мінімізують різницю між бажаними й дійсними змінами сигналу на виході моделі й у такий спосіб оптимізують керований процес. Побудова моделі керованого процесу виконується автономно з використанням нейронної мережі, що навчається в груповому режимі з використанням одного з алгоритмів навчання. Контролер, що реалізує такий регулятор, вимагає значного обсягу обчислень, оскільки для розрахунку оптимального закону керування оптимізація виконується на кожному такті керування.

Регулятор NARMA-L2. Із всіх архітектур цей регулятор вимагає найменшого обсягу обчислень. Даний регулятор – це просто деяка реконструкція нейромережевої моделі керованого процесу, отриманої на етапі ідентифікації.

Обчислення в реальному часі пов'язані тільки з реалізацією нейронної мережі. Недолік методу полягає в тому, що модель процесу повинна бути задана в канонічній формі простору стану, якій відповідає супровідна матриця, що може приводити до обчислювальних похибок.

Регулятор на основі еталонної моделі. Необхідний обсяг обчислень для цього регулятора можна зрівняти з попереднім. Однак архітектура регулятора з еталонною моделлю вимагає навчання нейронної мережі керованого процесу й нейронної мережі регулятора. При цьому навчання регулятора виявляється досить складним, оскільки навчання основане на динамічному варіанті методу зворотного поширення помилки. Перевагою регуляторів на основі еталонної моделі те що вони можуть застосовуватися до різних класів керованих процесів.

Практична робота

1. Необхідно реалізувати систему керування, яка б керувала рухом ланки промислового робота, відслідковуючи вихід еталонної моделі:

$$\frac{d^2 y_r}{dt^2} = -9 y_r - 6 \frac{dy_r}{dt} + 9r \qquad (25)$$

де y_r – вихід еталонної моделі; r – задає сигнал на вході моделі.

Структурна схема, що пояснює принцип побудови системи керування з еталонною моделлю, показана на рисунку 42.

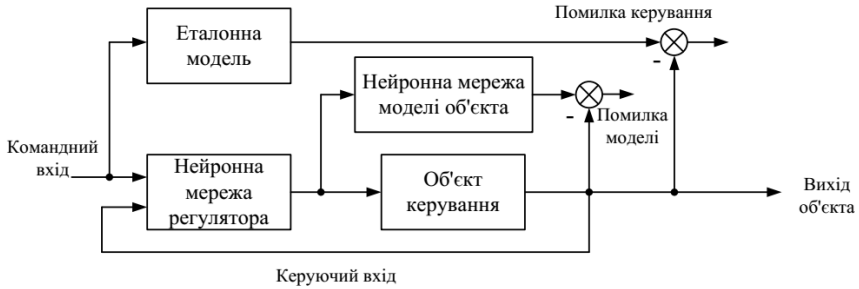


Рис. 42. Структурна схема

2. Побудуйте відповідну динамічну модель, реалізувавши її в *Simulink*.

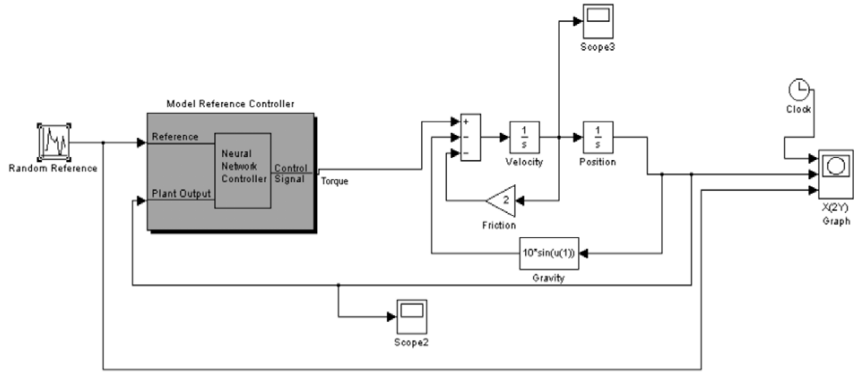


Рис. 43. Модель системи

3. Активуйте блок нейромережевого регулятора, двічі клацнувши на блок *Model Reference Controller*. Відкриється вікно графічного інтерфейсу контролера:

4. Перед тим як встановити параметри контролера, необхідно побудувати модель керованого процесу. Тобто необхідно виконати ідентифікацію керованого процесу – побудувати нейромережеву модель. Для цього відкрийте вікно спеціальної процедури *Plant Identification*, що зображене на рис. 44.

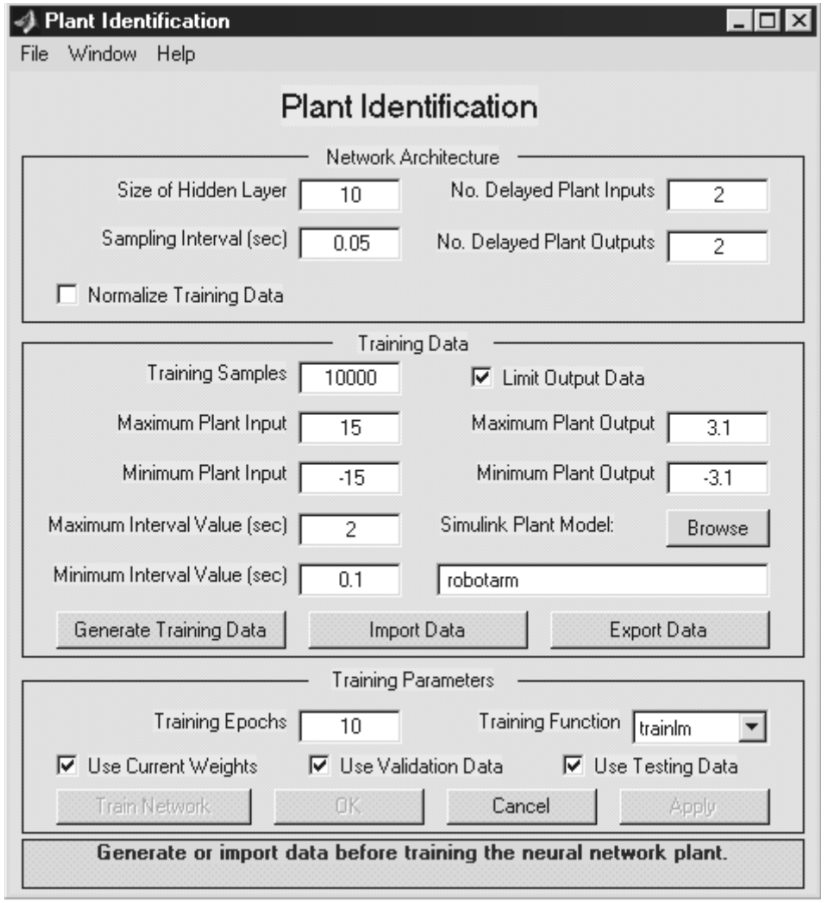


Рис. 44. Вид вікна Plant Identification

Це вікно може бути використане для побудови нейромережевих моделей для будь-якого динамічного об'єкта, що описаний моделлю Simulink.

5. Для системи керування необхідна динамічна модель, реалізована в Simulink, що відповідає рівнянню руху ланки, має наступний вигляд:

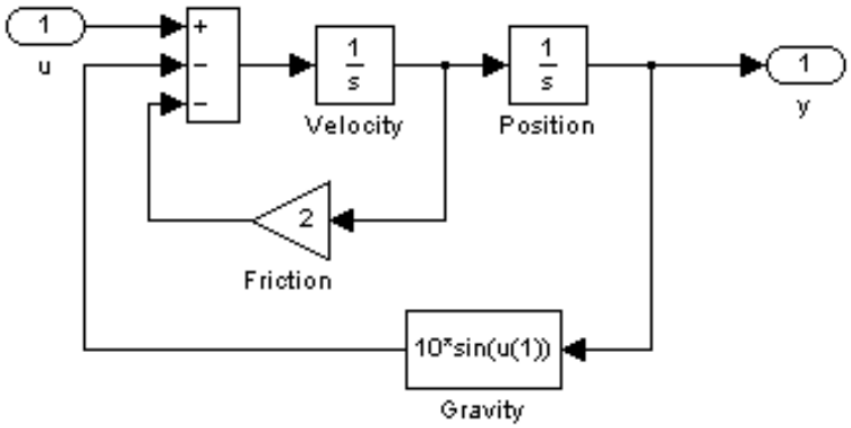


Рис. 45. динамічна модель, реалізована в Simulink

Реалізуйте наведену схему та збережіть її в робочому каталозі з ім'ям "robotarm".

6. Процедура ідентифікації потребує встановлення наступних параметрів:

– *Size of Hidden Layer* – розмір прихованого шару визначається кількістю використовуваних нейронів;

– *Sampling Interval* – такт дискретності в секундах визначає інтервал між двома послідовними моментами знімання даних;

– *No. Delayed Plant Inputs* – кількість елементів запізнення на вході моделі;

– *No. Delayed Plant Outputs* кількість елементів запізнення на виході моделі;

– *Normalize Trainig Data* – вікно контролю нормування навчальних даних до діапазону [0 1] ;

– *Trainig Samples* – довжина навчальної вибірки (кількість точок знімання інформації);

– *Maximum Plant Input* – максимальне значення вхідного сигналу;

– *Minimum Plant Input* – мінімальне значення вхідного сигналу;

– *Maximum Interval Value* – максимальний інтервал ідентифікації в секундах;

– *Minimum Interval Value* – мінімальний інтервал ідентифікації в секундах;

– *Limit Output Data* – вікно контролю, що дозволяє обмежити обсяг вихідних даних; тільки при включеному вікні контролю будуть доступні два наступних вікна редагування тексту;

– *Maximum Plant Output* – максимальне значення вихідного сигналу;

– *Minimum Plant Output* – мінімальне значення вихідного сигналу;

– *Simulink Plant Model* – завдання моделі Simulink із зазначенням вхідних і вихідних портів, що використовуються для побудови нейромережевої моделі керованого процесу;

– *Generate Trainig Data* – кнопка запуску процесу генерації навчальної послідовності;– *Import Data* – імпорт навчальної послідовності з робочої області або файла даних;

– *Export Data* – експорт згенерованих даних у робочу область або файл;

– *Trainig Epochs* – кількість циклів навчання;

– *Trainig Function* – завдання навчальної функції;

– *Use Current Weights* – вікно контролю, що дозволяє підтвердити використання поточних ваг нейронної мережі;

– *Use Validation/Testing For Training* – вибір цих вікон контролю буде означати, що 25 відсотків даних з навчальної послідовності буде використано для формування тестової і контрольної підмножин відповідно.

7. Установивши параметри, як зазначено на рис. 46, натисніть кнопку *Generate Trainig Data*, буде запущена програма генерації навчальної послідовності. Програма генерує навчальні дані шляхом впливу випадкових східчастих впливів на модель Simulink керованого процесу. Графіки вхідного і вихідного сигналів об’єкта мають бути такими, як зображено нижче:

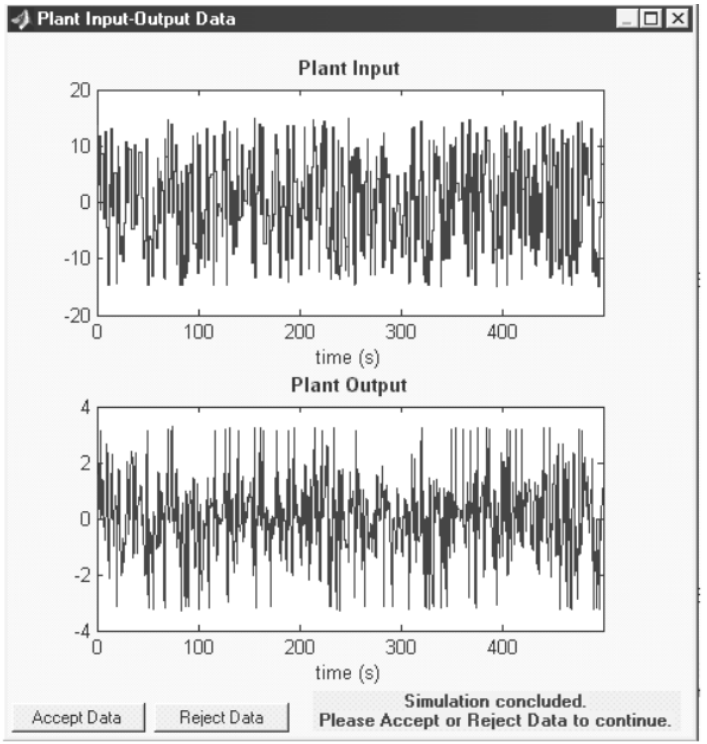


Рис. 46. Графіки вхідного і вихідного сигналів об’єкта

8. По завершенні генерації навчальної послідовності необхідно прийняти згенеровані дані (*Accept Data*) або відмовитися від них (*Reject Data*).

9. При поверненні до зміненого вікна *Plant Identification* починайте навчання нейронної мережі, натиснувши кнопку *Train Network*.

10. По завершенні навчання нейромережевої моделі на графіках відобразяться результати, а саме результати навчання та тестування на контрольній підмножині (*Training data, Testing data*). Для того щоб згенерувати або імпортувати нові дані, продовжити навчання або зберегти отримані результати, натисніть кнопку *Ok* або *Apply*. У результаті параметри нейромережевої моделі керованого процесу будуть занесені до блока *Model Reference Controller*.

11. Для навчання контролера в полі *Reference Model* необхідно вказати ім'я схеми, що описує еталонну модель. Для цього слід рівняння еталонної моделі (25) методом зниження похідної реалізувати в Simulink моделі та зберегти її в робочому каталозі з ім'ям "*robotref*".

Завдання до лабораторної роботи 11

Самостійно згенеруйте параметри контролера та навчіть мережу.

Навчіть регулятор на основі нейронної мережі. Установіть кількість циклів навчання за номером варіанту. Проведіть 5000 вимірів навчальної послідовності та виведіть результати на екран.

Порівняйте роботу моделі з різними параметрами, отримавши графіки їх роботи.

Контрольні запитання

1. Які архітектури нейронних мереж реалізовані в пакеті Neural Network Toolbox? Охарактеризуйте кожну з них.
2. Як побудувати нейромережеву модель?
3. Що таке ідентифікація керованого процесу?
4. Які обов'язкові параметри вказують при створенні нейромережевої моделі?

Література до розділу

1. Збірник завдань з курсу «Інтелектуальні інформаційні системи» навчальний посібник / С.В. Ліпатова. – Ульяновськ: УлГУ, 2010. – 64 с. (рос.)

2. Інтелектуальні інформаційні системи: навчальний посібник / А. А. Смагін, С. В. Ліпатова, О. С. Мельниченко. – Ульяновськ: УлГУ, 2010. – 136 с. (рос.)

3. Д'яконов В.П. MATLAB 6.5 SP1 7/7 SP1/7 SP1 + Simulink 5/6. Інструменти штучного інтелекту та біоінформатики / Д'яконов В. П., Круглов В. В. // Серия «Библиотека професионала». – М.: СОЛОН – ПРЕСС, 2006. – 456 с.: ил.

4. Stepashko V., Bulgakova O., Zosimov V. Construction and research of the generalized iterative GMDH algorithm with active neurons// Advances in Intelligent Systems and Computing Vol. 689, 2018, Pages 492-510 Springer Verlag.

5. Zosimov V. Khrystodorov O., Bulgakova O. Dynamically changing user interfaces: software solutions based on automatically collected user information// Programming and Computer Software, vol 44 (6), 2018, P. 492–498.

6. Зосімов В. В., Погромська Г. С., Махровська Н. А., Булгакова О. С. Інформаційний аналіз управління процесами в складних системах: теорія та практика : [монографія]. – Миколаїв, МНУ ім. В. О. Сухомлинського, 2018 – 217 с.

7. Степашко В. С., Зосімов В. В., Булгакова О. С. Ітераційні алгоритми індуктивного моделювання : [монографія]. – Київ, Наукова думка 2018 – 190 с.

РОЗДІЛ V. СИСТЕМИ FUZZY LOGIC

5.1. Нечітка інформація та нечітки висновки

Теорія нечітких множин, основні ідеї якої були запропоновані американським математиком Лотфі Заде (Lotfi Zadeh) в 1965 році, дозволяє описувати якісні, неточні поняття і наші знання про навколишній світ, а також оперувати цими знаннями з метою отримання нової інформації. Засновані на цій теорії методи побудови інформаційних моделей істотно розширюють традиційні сфери застосування комп'ютерів і утворюють самостійний напрям науково-прикладних досліджень, яке отримало спеціальну назву – нечітке моделювання.

Нечітка логіка як науковий напрям розвивалася непросто, не уникнула вона і звинувачень в лженауковості. Навіть у 1989 році, коли приклади успішного застосування нечіткої логіки в обороні, промисловості і бізнесі обчислювалися десятками, Національне наукове товариство США обговорювало питання про виключення матеріалів по нечітких множинах з інститутських підручників.

Перший період розвитку нечітких систем (кінець 60-х – початок 70-х рр.) характеризується розвитком теоретичного апарату нечітких множин. У 1970 році Беллман спільно із Заде розробили теорію прийняття рішень в нечітких умовах.

У 70–80 роки (другий період) з'являються перші практичні результати в області нечіткого управління складними технічними системами (парогенератор з нечітким управлінням). І. Мамдани в 1975 році спроектував перший контроллер, що функціонує на основі алгебри Заде та управляє паровою турбіною. Одночасно стала приділятися увага питанням створення експертних систем, побудованих на нечіткій логіці, розробці нечітких контроллерів. Нечіткі експертні системи для підтримки прийняття рішень знайшли широке застосування в медицині та економіці.

Нарешті, в третьому періоді, який триває з кінця 80-х років і триває нині, з'являються пакети програм для побудови нечітких експертних систем, а сфери застосування нечіткої логіки помітно розширюються.

Вона застосовується в автомобільній, аерокосмічній і транспортній промисловості, в області виробів побутової техніки, у сфері фінансів, аналізу та прийняття управлінських рішень і багатьох інших. Крім того, чималу роль в розвитку нечіткої логіки зіграло доведення знаменитої теореми FAT (Fuzzy Approximation Theorem) Б. Коско, в якій стверджувалося, що будь-яку математичну систему можна апроксимувати системою на основі нечіткої логіки.

Одним з самих вражаючих результатів стало створення мікропроцесора на основі нечіткої логіки. У 1990 році Комітет з контролю експорту США вніс нечітку логіку в список критично важливих оборонних технологій, що не підлягають експорту потенційному супротивникові.

У бізнесі і фінансах нечітка логіка отримала визнання після того, як в 1988 році експертна система на основі нечітких правил для прогнозування фінансових індикаторів єдина передбачила біржовий крах.

У Японії цей напрям переживає справжній бум. Тут функціонує спеціально створена організація Laboratory for International Fuzzy Engineering Research. Програмою цієї організації є створення більш близьких для людини обчислювальних пристроїв.

Інформаційні системи, що базуються на нечітких множинах і нечіткій логіці, називають нечіткими системами.

Переваги нечітких систем:

– функціонування в умовах невизначеності;
– оперування якісними і кількісними даними;
– використання експертних знань в управлінні;
– побудова моделей наближених до міркувань людини;
– стійкість при дії на систему всіляких обурень.

Недоліки нечітких систем:

– відсутність стандартної методики конструювання нечітких систем;
– неможливість математичного аналізу нечітких систем існуючими методами;
– застосування нечіткого підходу в порівнянні з імовірнісним не призводить до підвищення точності обчислень.

5.2. Визначення нечіткої множини

Нечітка множина (Fuzzy set) являє собою сукупність елементів довільної природи, стосовно яких не можна з повною впевненістю стверджувати – чи належить той чи інший елемент розглянутої сукупності даній множині чи ні. Іншими словами, нечітка множина відрізняється від звичайної множини тим, що для всіх або частини її елементів не існує однозначної відповіді на питання: «Чи належить або Чи не належить той чи інший елемент розглянуій нечіткій множині?»

Формально нечітка множина А визначається як множина впорядкованих пар або кортежів виду: $\langle x,\mu_A(x)\rangle$, де x є елементом деякого універсальної множини або універсуму Е, а $\mu_A(x)=\begin{cases}1, & x\in A;\\ 0, & x\notin A.\end{cases}$ – функція приналежності (або ступінь приналежності), яка ставить у відповідність кожному з елементів $x\in E$ деяке дійсне число з інтервалу [0, 1], тобто дана функція визначається в формі відображення:

$$\mu_A(x): E\to[0,1]$$

При цьому $\mu_A(x)=1$ для деякого $x\in E$ означає, що елемент x повністю належить нечіткій множині А, а значення $\mu_A(x)=0$ означає, що елемент x безумовно не належить нечіткій множині А.

Функція приналежності може бути визначена явно у вигляді функціональної залежності (наприклад, $\mu_A(x)=(1+|x-10|^m)^{-1}$, де $m\in N.$) Або дискретно – шляхом завдання кінцевої послідовності значень $x\in\{x_i\}$ у вигляді

$$A=\{\mu_A(x_1)/x_1+\mu_A(x_2)/x_2+\ldots+\mu_A(x_n)/x_n\},$$

або у вигляді

$$A=\left\{\frac{\mu_A(x_1)}{x_1}+\frac{\mu_A(x_2)}{x_2}+\ldots+\frac{\mu_A(x_n)}{x_n}\right\}$$

при цьому коса і горизонтальна риска служить просто роздільником, а знак «+» позначає не арифметичну суму, а теоретико-множинне об'єднання окремих елементів. Нескінченні нечіткі множини іноді записують зі знаком інтеграла у вигляді $\int\mu_A(x)/x$. Все це скоріше данина

традиції, ніж щось, що має змістовний сенс. Бажаючи підкреслити або явно вказати, що множина А є нечіткою, іноді часто записують нечітку множину зі знаком тильда «~» вгорі.

З усіх нечітких множин виділимо два окремих випадки, які, по суті, збігаються зі своїми класичними аналогами і використовуються в подальшому при визначенні інших нечітких понять.

Пуста нечітка множина. В теорії нечітких множин зберігають свій сенс деякі спеціальні класичні множини. Так, наприклад, порожню нечітку множину або множина, яка не містить жодного елемента, як і раніше позначається через $\varnothing$ і формально визначається як така нечітка множина, функція приналежності якої тотожно дорівнює нулю для всіх без винятку елементів: $\mu_\varnothing=0$.

Універсум. Що стосується іншої спеціальної множини, то так званий універсум, що позначається через E, вже був використаний вище в якості звичайної множини, що містить в рамках деякого контексту всі можливі елементи. Формально зручно вважати, що функція приналежності універсуму як нечіткої множини тотожно дорівнює одиниці для всіх без винятку елементів: $\mu_E=1$.

Розглянемо представлені нижче приклади.

Задання нечіткої множини. Нехай $E=\{x_1, x_2, x_3, x_4, x_5\}$; A – нечітка множина, для якої

$$\mu_A(x_1)=0.3;\ \mu_A(x_2)=0;\ \mu_A(x_3)=1;\ \mu_A(x_4)=0.5;\ \mu_A(x_5)=0.9;$$

Тоді A можна уявити, наприклад, у вигляді:

$$A=0.3/x_1+0/x_2+1/x_3+0.5/x_4+0{,}9/x_5$$

або у вигляді

$A=$	x_1	x_2	x_3	x_4	x_5
	0.3	0	1	0.5	0.9

Приклади нечітких множин

1. Нехай E = {0, 1, 2, .., 10}. Нечітку множину «декілька» можна визначити наступним чином:

«декілька» = 0.5/3 + 0.8/4 + 1/5 + 1/6 + 0.8/7 + 0.5/8.

2. Нехай E= {0, 1, 2, 3, ..., *n,* ...}. Нечітку множину «малий» можна визначити за допомогою функції приналежності

$$\mu_{малий}(n) = \frac{1}{1+\left(\frac{n}{10}\right)^2}$$

Введемо далі наступні поняття:

Нечіткі числа – нечіткі змінні, визначені на числової осі, тобто нечітке число визначається як нечітка множина *A* на множині дійсних чисел з функцією приналежності $\mu_A(x) \in [0,1]$, де *x* – дійсне число.

В теорії нечітких множин, крім змінних числового типу, існують так звані лінгвістичні змінні. Нехай змінна *x* позначає вік (*x* = «вік»). Тоді, за аналогією з вищенаведеним прикладом, можна визначити нечіткі поняття «молодий», «літній», «старий», що характеризуються деякими функціями належності $\mu_{молодий}(x)$, $\mu_{літній}(x)$, $\mu_{старий}(x)$. Так само, як звичайна змінна може набувати різних значень, лінгвістична змінна «вік» може приймати різні лінгвістичні значення. У нашому прикладі це: «молодий», «літній» і «старий».

Лінгвістичною змінною називається змінна, значеннями якої можуть бути слова або словосполучення деякої природньої або штучної мови.

Терм-множиною називається множина всіх можливих значень лінгвістичної змінної.

Термом називається будь-який елемент терм-множини. В теорії нечітких множин терм формалізується нечіткою множиною за допомогою функції приналежності.

Кожна нечітка множина має певний носій (англ. support). *Носієм* множини Supp (*A*) є підмножина тих елементів *A*, для яких ступінь приналежності до *A* не дорівнює нулю, тобто $Supp(A) = \{x, \mu_A(x) > 0\}$.

Ядром нечіткої множини А називається чітка множина таких точок *A* в *E*, для яких величина $\mu_A(x) = 1$.

Множиною рівня α (α-розрізом) нечіткої множини А називається чітка підмножина універсальної множини, визначена по формулі $A_\alpha = \{x | \mu_A(x) \geq \alpha\}$, де $\alpha \in [0,1]$.

Функцію приналежності називають *нормальною*, якщо ядро нечіткої множини містить хоч би один елемент.

5.3. Нечіткість та ймовірність

Добре відомий той факт, що стохастична (ймовірнісна) та лінгвістична невизначеності мають різний характер. Стохастична невизначеність має справу з невизначеністю того, чи відбудеться деяка описана подія в майбутньому, а теорія ймовірностей дозволяє дати на це питання ту чи іншу відповідь.

Лінгвістична невизначеність пов'язана з неточністю опису самої ситуації або події незалежно від часу їх розгляду. Теорія ймовірностей не може використовуватися для вирішення подібних проблем, оскільки уявлення про суб'єктивні категорії, присутніх в процесах мислення людини, в повній мірі не узгоджуються з її аксіомами.

Тим не менш, деякі з фахівців, які інтенсивно працюють з теорією ймовірності та математичної статистики, довгий час заперечували саму можливість застосування нечіткої логіки. Ці фахівці часто стверджували, що всі види невизначеності можуть бути виражені в поняттях теорії ймовірності. У той же час, очевидно, що як теорію ймовірностей, так і теорію нечітких множин доцільно використовувати для моделювання різних аспектів невизначеності, що відрізняються за своєю природою.

Наведемо в цьому зв'язку наочний приклад, який добре ілюструє семантичне розходження між стохастичною та лінгвістичної невизначеністю. Уявімо собі ситуацію, коли подорожуючий після тривалої подорожі, відчуваючи почуття спраги, знаходить дві пляшки з невідомою рідиною всередині кожної з них. Природним бажанням подорожуючого є вгамувати свою спрагу. Однак ніяких етикеток із зазначенням напоїв знайдені пляшки не містять крім, можливо, позначок А і Б.

Припустимо, що додатково відома ступінь приналежності містимого пляшки А до рідин, придатним для пиття, і ця ступінь приналежності дорівнює 0.91. Відома також ймовірність того, що вміст пляшки Б придатне для пиття, і ця ймовірність також дорівнює 0.91. Якщо подорожуючий обмежений у виборі напоїв цими двома пляшками, яку з них йому слід вибрати для втамування спраги?

Якщо подорожуючий знайомий з теорією нечітких множин та теорією ймовірності, то його вибір може грунтуватися на наступному міркуванні. Аналізуючи інформацію про вміст пляшки А, він може припустити, що в ній знаходиться не зовсім придатна для пиття рідина, наприклад, болотна вода. При цьому природно вважати, що чиста вода мала б ступінь приналежності, рівну 1. У той же час в цій пляшці не може перебувати отруйна рідина, скажімо, сірчана кислота, оскільки в цьому випадку ступінь приналежності вмісту пляшки А до рідин, придатним для пиття, була б дорівнювати 0.

Аналізуючи інформацію про вміст пляшки Б, резонно припустити, що якби подорожуючий мав можливість багаторазового вибору пляшки Б, то приблизно в 9 випадках з 10 він зміг би благополучно вгамувати свою спрагу. При цьому вміст пляшки Б мав би бути за якістю близьким до чистої води. Що ж має статися в тому єдиному випадку з розглянутих 10 – залишається незрозумілим. Можливо, результатом може виявитися найсумніший результат для людини або її серйозне нездужання. Очевидно, що це не може статися, якщо в пляшці Б знаходиться пиво або квас. Значить, в одному випадку з 10 в цій пляшці може перебувати щось зовсім неприйнятне для пиття, наприклад, соляна кислота.

Висновок напрошується очевидний – при наявності тільки зазначеної інформації необхідно вибрати вміст пляшки А. У всякому разі це дозволить уникнути серйозного отруєння або летального результату. Але цей вибір може змінитися, якщо зміниться інформація про вміст пляшок. Так, зовсім не очевидно, яку з пляшок віддати перевагу, якщо ступінь приналежності для пляшки А і ймовірність для пляшки Б стануть рівними 0.5.

Таким чином, поняття нечіткої множини здатне забезпечити нас адекватною інформацією щодо неточного опису тих чи інших ситуацій. По суті, цей підхід найбільш прийнятний для вирішення таких проблем, в яких невизначеність характеризується відсутністю добре визначених критеріїв, що дозволяють однозначно судити про приналежність елементів того чи іншого класу. Саме в цьому проявляється відмінність між нечіткістю та ймовірністю. У той же час нечіткі моделі не є заміною моделей, розроблених в теорії ймовірностей.

5.4. Операції над нечіткими множинами і відносинами

Операції над нечіткими множинами

Логічні операції

Розглянемо основні логічні операції, які можливі з нечіткими множинами.

1. *Включення*. Нехай A і B – нечіткі множини на універсальній множині E.

Кажуть, що A міститься в B, якщо $\forall x \in E \ \ \mu_A(x) \le \mu_B(x)$.

Позначення: $A \subset B$.

Іноді використовують термін «домінування», тобто в разі, коли $A \subset B$, кажуть, що B домінує над A.

2. *Рівність*. A і B рівні, якщо $\forall x \in E \ \ \mu_A(x) = \mu_B(x)$. Позначення: A = B.

3. *Доповнення*. Нехай A і B – нечіткі множини, задані на E. A і B доповнюють один одного, якщо $\forall x \in E \ \ \mu_A(x) = 1 - \mu_B(x)$.

Позначення: $B = \overline{A}$ или $A = \overline{B}$. Очевидно, що $\overline{\overline{A}} = A$.

4. *Перетин*. $A \cap B$ – найбільша нечітка підмножина, що міститься одночасно в A та B:

$$\mu_{A \cap B}(x) = \min(\mu_A(x), \mu_B(x)) = \mu_A(x) \wedge \mu_B(x).$$

5. *Об'єднання*. A і B – найменша нечітка підмножина, що включає як A, так і B, з функцією приналежності

$$\mu_{A \cup B}(x) = \max(\mu_A(x), \mu_B(x)) = \mu_A(x) \vee \mu_B(x).$$

6. *Різниця*. $A - B = A - \overline{B}$ з функцією приналежності

$$\mu_{A-B}(x) = \min(\mu_A(x), \ 1 - \mu_B(x)).$$

Приклад. Нехай

$$A = 0.4 / x_1 + 0.2 / x_2 + 0 / x_3 + 1 / x_4$$

$$B = 0.7 / x_1 + 0.9 / x_2 + 0.1 / x_3 + 1 / x_4$$

$$C = 0.1 / x_1 + 1 / x_2 + 0.2 / x_3 + 0.9 / x_4$$

тут:

1) $A \subset B$, тобто A міститься в B або B домінує над A, C незрівнянно ні з A, ні з B, тобто пари $\{A, C\}$ і $\{B, C\}$ – пари недоминирующих нечітких множин;

2) $A \neq B \neq C$

3) $\overline{A} = 0.6 / x_1 + 0.8 / x_2 + 1 / x_3 + 0 / x_4$

$\overline{B} = 0.3 / x_1 + 0.1 / x_2 + 0.9 / x_3 + 0 / x_4$

4) $A \cap B = 0.4 / x_1 + 0.2 / x_2 + 0 / x_3 + 1 / x_4$

5) $A \cup B = 0.7 / x_1 + 0.9 / x_2 + 0.1 / x_3 + 1 / x_4$

6) $A - B = A \cap \overline{B} = 0.3 / x_1 + 0.1 / x_2 + 0 / x_3 + 0 / x_4$

$B - A = \overline{A} \cap B = 0.6 / x_1 + 0.8 / x_2 + 0.1 / x_3 + 0 / x_4$

Для нечітких множин можна будувати візуальне уявлення. Розглянемо прямокутну систему координат, на осі ординат якої відкладаються значення $\mu_A(x)$, а на осі абсцис в довільному порядку розташовані елементи E. Якщо E за своєю природою впорядковано, то цей порядок бажано зберегти в розташуванні елементів на осі абсцис. Таке уявлення робить наочними прості логічні операції над нечіткими множинами (рис. 47).

У верхній частині рис. 47 заштрихована область відповідає нечіткій множині A і, якщо говорити точно, зображує область значень A і всіх нечітких множин, що містяться в A та $\overline{A}$ У нижній частині рис. показані $A \cap \overline{A},\ A \cup \overline{A}$.

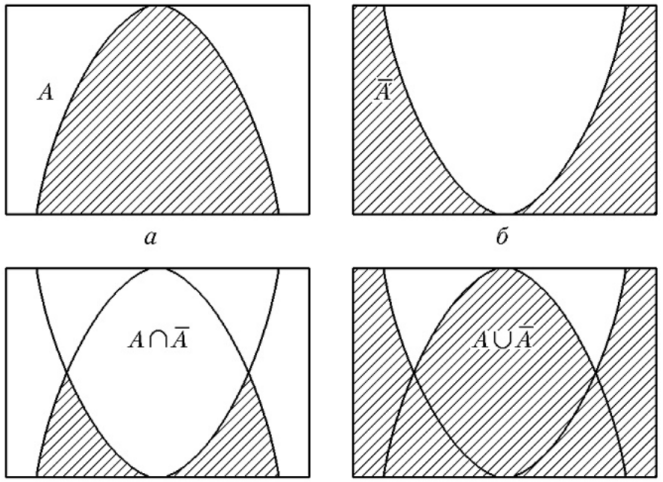


Рис. 47. Графічна інтерпретація логічних операцій

Зауваження. Введені вище операції над нечіткими множинами засновані на використанні операцій max та min. В теорії нечітких множин розробляються питання побудови узагальнених, параметрезованих операторів перетину, об'єднання та доповнення, що дозволяють врахувати різноманітні смислові відтінки відповідних їм зв'язок «І», «АБО», «НІ».

Один з підходів до операторів перетину та об'єднання полягає в їх визначенні в класі трикутних норм та конорм.

Трикутної нормою (Т-нормою) називається двомісна дійсна функція *T:* $[0,1] \times [0,1] \to [0,1]$, яка задовольняє таким умовам:

1) $T(0,0) = 0;\ T(\mu_A, 1) = \mu_A;\ T(1, \mu_A) = \mu_A$ – обмеженість;

2) $T(\mu_A, \mu_B) \le T(\mu_C, \mu_D)$, якщо $\mu_A \le \mu_C,\ \ \mu_B \le \mu_D)$ – монотонність;

3) $T(\mu_A, \mu_B) = T(\mu_B, \mu_A)$ – комутативность;

4) $T\left(\mu_A, T\left(\mu_B, \mu_C\right)\right) = T\left(T\left(\mu_A, \mu_B\right), \mu_C\right)$ – асоціативність.

Простим випадком трикутних норм є:

- $\min(\mu_A, \mu_B)$;
- добуток $\mu_A \cdot \mu_B$;
- $\max(0,\ \mu_A + \mu_B - 1)$.

Трикутною конормою (Т-конормою або S-нормою) називається двомісна дійсна функція *S:* $[0,1] \times [0,1] \to [0,1]$, з властивостями:

1) $S(1,1) = 1;\ S(\mu_A, 0) = \mu_A;\ S(0, \mu_A) = \mu_A$ – обмеженість;

2) $S(\mu_A, \mu_B) \ge S(\mu_C, \mu_D)$, якщо $\mu_A \ge \mu_C,\ \ \mu_B \ge \mu_D)$ – монотонність;

3) $S(\mu_A, \mu_B) = S(\mu_B, \mu_A)$ – комутативність;

4) $S(\mu_A, S(\mu_B, \mu_C)) = S(S(\mu_A, \mu_B), \mu_C)$ – асоціативність.

Приклади Т-конорм:

- $\max(\mu_A, \mu_B)$;
- $\mu_A + \mu_B - \mu_A \cdot \mu_B$;
- $\min(1,\ \mu_A + \mu_B)$.

Алгебраїчні операції

1. *Алгебраїчний добуток A* і *B* позначається A • B і визначається:

$$\forall x \in E\ \ \mu_{A \cdot B}(x) = \mu_A(x) \cdot \mu_B(x)$$

2. *Алгебраїчна сума* цих множин позначається $A \hat{+} B$ і визначається:

$$\forall x \in E \;\; \mu_{A \hat{+} B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$$

На основі операції алгебраїчного добутку визначається операція піднесення до ступеню α нечіткої множини А, де α – додатне число. Нечітка множина A^a визначається функцією приналежності $\mu_{A^\alpha}(x) = \mu_A^\alpha(x)$. ОЧастинними випадками піднесення до ступеню в ступінь є:

CON (A) = A^2 – операція концентрування (ущільнення),

DIL (A) = $A^{0.5}$ – операція розтягування, яка використовуються при роботі з лінгвістичними змінними.

Перша з цих операцій ототожнюється при цьому з поняттям «дуже», лінгвістичне значення другої операції – «приблизно».

3. *Декартовий (прямий) добуток* нечітких множин. Нехай A_1, A_2, ..., A_n – нечіткі підмножини універсальних множин E_1, E_2, ..., E_n відповідно. Декартовий або прямий добуток $A = A_1 \times A_2 \times ... \times A_n$ є нечіткою підмножиною множини $E = E_1 \times E_2 \times ... \times E_n$ з функцією приналежності

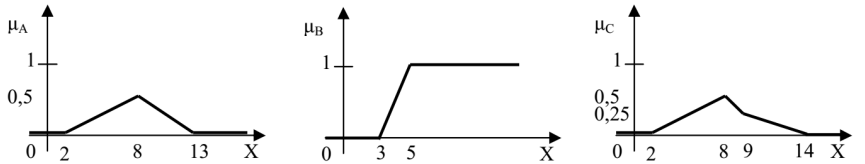
$$\mu_A(x_1, x_2, ..., x_n) = \min\{\mu_A(x_1), \mu_A(x_2), ..., \mu_A(x_n)\}.$$

Для визначення перетину та об'єднання нечітких множин найбільшою популярністю користуються наступні три групи операцій, що вміщують в собі попередні:

Максимінні	$\mu_{A\cup B}(x) = \max\{\mu_A(x), \mu_B(x)\}$, $\mu_{A\cap B}(x) = \min\{\mu_A(x), \mu_B(x)\}$
Алгебраїчні	$\mu_{A\cup B}(x) = \mu_A(x) + \mu_B(x) - \mu_A(x) \cdot \mu_B(x)$, $\mu_{A\cap B}(x) = \mu_A(x) \cdot \mu_B(x)$
Обмежені	$\mu_{A\cup B}(x) = \min\{1, \mu_A(x) + \mu_B(x)\}$, $\mu_{A\cap B}(x) = \max\{0, \mu_A(x) + \mu_B(x) - 1\}$

Доповнення нечіткої множини в усіх трьох випадках визначається однаково: $\mu_{\bar{A}}(x) = 1 - \mu_A(x)$.

Задача. Дано 3 нечітких множини *A, B, C* (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = \overline{A} \cap (A \cup C \cup B)$ і визначити ступінь приналежності одного елемента множини *D*, використовуючи метод обмежень.



Опис процесу розв'язку

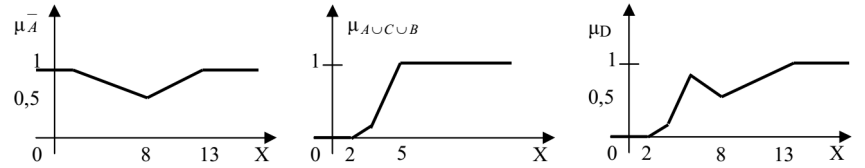
Для побудови функції приналежності нової множини необхідно:

1. Визначити послідовність виконання операцій у формулі.
2. Побудувати на окремих графіках проміжні множини, відповідно до визначеної послідовності дій. Звести проміжні множини на одному графіку і визначити підсумкову функцію приналежності.
3. Використовуючи певний в завданні метод, визначити аналітично ступінь приналежності елемента, що входить в ядро (від $\mu_D(x) = 0$ до $\mu_D(x) = 1$) підсумкової множини.
4. Перевірити аналітичні обчислення по побудованому графіку функції приналежності.

Розв'язок

1. Множина $D = \overline{A} \cap (A \cup C \cup B)$, тобто, послідовність операцій буде наступною: $\overline{A}$, $A \cup C \cup B$, $\overline{A} \cap (A \cup C \cup B)$

2. Побудуємо відповідно до цієї послідовності операцій графіки функцій приналежності:



3. Ядро множини *D* складається з елементів з інтервалу (2,13). Виберемо елемент 8.

$\mu_A(8) = 0.5$;

$\mu_B(8) = 1$;

$\mu_C(8) = 0.5$;

$\mu_{\overline{A}}(8) = 1 - \mu_A(8) = 1 - 0.5 = 0.5$;

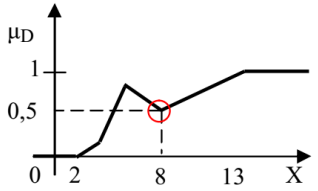
$\mu_{\overline{C}}(8) = 1 - \mu_C(8) = 1 - 0.5 = 0.5$;

$\mu_{A\cup C}(8) = \min\{1, \mu_C(8) + \mu_A(8)\} = \min\{1, 0.5 + 0.5\} = 1$;

$\mu_{A\cup C\cup B}(8) = \min\{1, \mu_{A\cup C}(8) + \mu_B(8)\} = \min\{1, 1+1\} = 1$;

$\mu_{\overline{A}\cap(A\cup C\cup B)}(8) = \max\{0, \mu_{\overline{A}}(8) + \mu_{A\cup C\cup B}(8) - 1\} = \max\{0, 0.5 + 1 - 1\} = 0.5$

4. Ядро множини $\mu_D(8) = 0.5$



Нечіткі відношення

Нехай $E = E_1 \ E_2 \ \ldots \ E_n$ – прямий добуток універсальних множин. Нечітке *n*-арне відношення визначається як нечітка підмножина *R* на *E*, що приймає свої значення на відрізку [0, 1]. У разі $n = 2$ нечітким відношенням *R* між множинами $X = E_1$ та $Y = E_2$ буде називатися функція R: $(X, Y) \to [0,1]$, яка ставить у відповідність кожній парі елементів (x, y) $X \times Y$ величину [0,1]. Позначення: нечітке відношення на $X \times Y$ запишеться в наступному вигляді: . У разі, коли $X = Y$, тобто X та Y збігаються, нечітке відношення R: $X \times X \to [0,1]$ називається *нечітким відношенням* на множині X.

Приклади.

1. Нехай $X = \{x_1, x_2, x_3\}$, $Y = \{y_1, y_2, y_3\}$. Нечітке відношення $R = XRY$ може бути задано, наприклад, у вигляді таблиці.

R	y_1	y_2	y_3	y_4
x_1	0	1	0.2	1
x_2	0	0.7	0.4	1
x_3	1	0.5	0.6	0.1

2. Нехай $X = Y = (-\infty, \infty)$, тобто множина всіх дійсних чисел. Відношення $x >> y$ (x забагато більше y) можна задати наступною функцією приналежності:

$$\mu_R = \begin{cases} \dfrac{0.1}{1 + \dfrac{1}{(x-y)^2}}, & \text{якщо } x \le y, \\ \text{якщо } y < x \end{cases}$$

3. Відношення R, для якого $\mu_R(x,y) = e^{-k(x-y)^2}$, при досить великих k можна інтерпретувати так: x і y – близькі один до одного числа.

Операції над нечіткими відношеннями

1. *Об'єднання двох відношень.*

Об`єднання двох відношень R_1 і R_2 позначається $R_1 \cup R_2$ і визначається виразом:

$$\mu_{R_1\cup R_2}(x,y) = \mu_{R_1}(x,y) \vee \mu_{R_2}(x,y) = \max\left[\mu_{R_1}(x,y), \mu_{R_2}(x,y)\right]$$

2. *Перетин двох відношень.*

Перетин двох відношень R_1 і R_2 позначається $R_1 \cap R_2$ і визначається виразом:

$$\mu_{R_1\cup R_2}(x,y) = \mu_{R_1}(x,y) \wedge \mu_{R_2}(x,y) = \min\left[\mu_{R_1}(x,y), \mu_{R_2}(x,y)\right]$$

3. *Алгебраїчний добуток двох відношень.*

Алгебраїчний добуток двох відношень R_1 і R_2 позначається $R_1 \cdot R_2$ і визначається виразом

$$\mu_{R_1R_2}(x,y) = \mu_{R_1}(x,y) \cdot \mu_{R_2}(x,y)$$

4. *Алгебраїчна сума двох відношень.*

Алгебраїчна сума двох відношень R_1 і R_2 позначається $R_1 \hat{+} R_2$ і визначається виразом

$$\mu_{R_1\hat{+}R_2}(x,y) = \mu_{R_1}(x,y) + \mu_{R_2}(x,y) - \mu_{R_1}(x,y) \cdot \mu_{R_2}(x,y)$$

5. *Доповнення відносини.*

Доповнення *відношення* R позначається $\overline{R}$ і визначається функцією приналежності

$$\mu_{\overline{R}}(x,y) = 1 - \mu_R(x,y).$$

6. *Звичайне відношення, найближче до нечіткого.*

Нехай R – нечітке відношення з функцією приналежності $\mu_R(x,y)$. Звичайне відношення, найближче до нечіткого, позначається $\underline{R}$ і визначається виразом:

$$\mu_{\underline{R}}(x,y)=\begin{cases}0, & \text{якщо } \mu_R(x,y)<0.5\\ 1, & \text{якщо } \mu_R(x,y)>0.5\\ 0 \text{ або } 1, & \text{якщо } \mu_R(x,y)=0.5\end{cases}$$

За домовленістю приймають $\mu_R(x,y)=0$ при $\mu_R(x,y)=0.5$.

7. *Композиція (згортка)* двох нечітких відношень.

Нехай R_1 – нечітке відношення $R_1:(X\times Y)\to[0,1]$ між X та Y; R_2 – нечітке відношення $R_2:(Y\times Z)\to[0,1]$ між Y та Z. Нечітке відношення між X та Z, позначається $R_1\bullet R_2$, та визначається як:

$$\mu_{R_1\bullet R_2}(x,y)=\bigvee_y[\mu_{R_1}(x,y)\wedge\mu_{R_2}(x,y)]=\max[\min[\mu_{R_1}(x,y),\mu_{R_2}(x,y)]]$$

де символом «$\bigvee_y$» позначена операція вибору найбільшого по y значення, називається (*max-min*) – композицією ((*max-min*) – згортка) відношень R_1 і R_2.

Приклад. Нехай відносини R_1 і R_2 визначаються наступним чином:

R_1	y_1	y_2	y_3
x_1	0.1	0.7	0.4
x_2	1	0.5	0

R_2	z_1	z_2	z_3	z_4
y_1	0.9	0	1	0.2
y_2	0.3	0.6	0	0.9
y_3	0.1	1	0	0.5

тоді

$R_1\bullet R_2$	z_1	z_2	z_3	z_4
x_1	0.3	0.6	0.1	0.7
x_2	0.9	0.5	1	0.5

При цьому

$$\mu_{R_1R_2}(x_1,z_1)=\left[\mu_{R_1}(x_1,y_1)\wedge\mu_{R_2}(y_1,z_1)\right]\vee\left[\mu_{R_1}(x_1,y_2)\wedge\mu_{R_2}(y_2,z_1)\right]\vee$$
$$\vee\left[\mu_{R_1}(x_1,y_3)\wedge\mu_{R_2}(y_3,z_1)\right]=(0.1\wedge0.9)\vee(0.7\wedge0.3)\vee(0.4\wedge0.1)=$$
$$=0.1\vee0.3\vee0.1=0.3$$
$$\mu_{R_1R_2}(x_1,z_2)=\left[\mu_{R_1}(x_1,y_1)\wedge\mu_{R_2}(y_1,z_2)\right]\vee\left[\mu_{R_1}(x_1,y_2)\wedge\mu_{R_2}(y_2,z_2)\right]\vee$$
$$\vee\left[\mu_{R_1}(x_1,y_3)\wedge\mu_{R_2}(y_3,z_2)\right]=(0.1\wedge0)\vee(0.7\wedge0.6)\vee(0.4\wedge1)=$$
$$=0\vee0.6\vee0.4=0.3$$

Зауваження. В даному прикладі спочатку використаний «аналітичний» спосіб композиції відношень R_1 і R_2, тобто i-й рядок R_1, «множиться» j-й стовпець R_2 з використанням операції, Отриманий результат «згортається» з використанням операції v в $\mu(x_i,z_j)$.

5.5. Нечітка імплікація

Нехай маємо звичайні («чіткі») висловлювання виду: p = «$x\in A$» та q = «$y\in B$». Тоді імплікацією (у звичайній, «чіткій» логіці) «якщо p, то q» називається речення, яке хибне тоді і тільки тоді, коли p істинне, а q хибне.

Такий вид відносин між висловлюваннями зазвичай позначається як $p\to q$ і розуміється як твердження «p тягне за собою q».

Якщо перейти до бінарної (булевої) алгебри логіки, де з поняттям «істина» зіставляється 1, а з поняттям «брехня» – 0, то імплікацію можна уявити логічною формулою:

$$p\to q=\overline{p}\vee q=\overline{p\wedge\overline{q}}$$

або таблицею істинності:

p	q	$p\to q$
1	1	1
0	1	1
0	0	1
1	0	0

Більш повна трактування поняття імплікації означає, що істинність $p\to q$ – це те ж, що істинність твердження «ступінь істинності q не менш, ніж ступінь істинності p».

*Приклад***.** Нехай p = «x більше 5» і q = «x більше 4». Легко бачити, що в даному випадку імплікація $p \rightarrow q$ є істинною, оскільки з нерівності x> 5 слід нерівність x> 4.

Нечітка імплікація, в принципі, зберігає той же сенс, що і імплікація чіткої логіки. Відмінність полягає лише в тому, що в цьому випадку «ступеня істинності »можуть мати будь-яке значення між 0 і 1.

Нечітка імплікація визначається наступним чином.

Будемо вважати, що задані універсальні множини X і Y, які містять кінцеве число елементів. Під способом визначення нечіткої імплікації «якщо A, то B», де A і B – нечіткі множини на X і Y відповідно, будемо розуміти спосіб завдання нечіткого відношення R на $X \times Y$, яке відповідає заданому висловлюванню.

Таке відношення можна задати по-різному, тому для математичного представлення нечіткої імплікації запропоновано велику кількість різних формул, одна з яких наведена нижче (формула Kleene-Dienes):

$$A \rightarrow B = \overline{A \cup B} \; i \; \mu_{A \rightarrow B}(x) = \max\left\{1 - \mu_A(x), \mu_B(x)\right\}.$$

5.6. Нечіткі висновки

Як робляться висновки у звичайній (чіткої) логіці?

Проілюструємо механізм виведення на прикладі обчислення значення деякої функції $y = f(x)$. Тут можна виділити наступну послідовність дій:

передумова: $y = F(x)$
факт: $x = x_0$

наслідок: $y_0 = f(x_0)$

Щось подібне відбувається і при операціях з нечіткими поняттями і числами.

Відзначимо спочатку, що механізм нечітких висновків який використовується в різного роду експертних і керуючих системах у своїй основі має базу знань, що формується фахівцями предметної області у вигляді сукупності нечітких предикатних правил виду:

$П_1$: якщо $x \in A_1$, тоді $y \in B_1$
$П_2$: якщо $x \in A_2$, тоді $y \in B_2$,
.....
$П_m$: якщо $x \in A_m$, тоді $y \in B_m$,

де x – вхідна змінна (відомі значення), y – змінна виводу (значення які будуть обчислені); A_m та B_m – нечітки множини, визначені відповідно на X і Y.

Приклад подібного правила: Якщо x – «низьке», то y – «високе».

Механізм нечіткого виводу можна представити у вигляді, аналогічному раніше наведеному:

передумова:
$П_1$: якщо $x \in A_1$, тоді $y \in B_1$
$П_2$: якщо $x \in A_2$, тоді $y \in B_2$,
.....
$П_m$: якщо $x \in A_m$, тоді $y \in B_m$,
факт: $x \in A$

наслідок: $y \in B$

Операція композиції

Операцію композиції (як і операцію імплікації) в алгебрі нечітких множин можна реалізовувати по-різному (при цьому, природно, буде відрізнятися і підсумковий одержуваний результат), але в будь-якому випадку загальний логічний висновок здійснюється за наступні чотири етапи.

1. *Введення нечіткості*, *фазифікація* (перехід від заданих чітких значень до мір упевненості). Функції приналежності, визначені на вхідних змінних, застосовуються до їх фактичних значеннь для визначення ступеня істинності кожної передумови кожного правила.

2. *Логічний висновок*. Обчислення значення істинності для передумов кожного правила застосовується до висновків кожного правила. Це призводить до одної нечіткої підмножині, яке буде призначено кожній змінній виводу для кожного правила. В якості правил логічного висновку зазвичай використовуються тільки операції min (МІНІМУМ) або prod (МНОЖЕННЯ). У логічному висновку МІНІМУМУ функція приналежності виведення «відсікається» по висоті, відповідного

обчисленого ступеня істинності передумови правила (нечітка логіка «І»). У логічному висновку МНОЖЕННЯ функція приналежності виведення масштабується за допомогою обчисленого ступеня істинності передумови правила.

3. *Агрегування (акумуляція)* (об'єднання результатів застосування усіх правил). Всі нечіткі підмножини, призначені до кожної змінної виводу (у всіх правилах), об'єднуються разом, щоб сформувати одну нечітку підмножину для кожної змінної виводу. При подібному об'єднанні зазвичай використовуються операції max (МАКСИМУМ) або sum (СУМА). При композиції МАКСИМУМУ комбінований висновок нечіткої підмножини конструюється як поточечной максимум за всіма нечітким підмножини (нечітка логіка «АБО»). При композиції СУМИ комбінований висновок нечіткої підмножини конструюється як поточечной сума за всіма нечітким підмножини, призначеним змінної виводу правилами логічного висновку.

4. На закінчення (додатково) – *приведення до чіткості* (*дефазифікація*), яке використовується, коли необхідно перетворити нечіткий набір висновків в чітке число. Є більша кількість методів приведення до чіткості, наприклад за допомогою центроїдного методу.

Задача. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для задачі визначення тимчасових витрат для вирішення студентом завдань даного посібника (враховувати успішність студента і кількість вирішуваних варіантів), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

Опис процесу розв'язку

Для реалізації логічного висновку необхідно виконати наступне:

1) сформулювати на природній мові у вигляді пропозицій «Якщо .., то» закономірності предметної о бласті;

2) виділити з цих пропозицій лінгвістичні змінні, їх значення (побудувати їх функції приналежності), висловлювання різних видів, формалізувати нечіткі правила;

3) перевірити отриману базу знань на повноту;

4) провести фазифікації (вхідні дані вибираємо випадковим чином) та логічний висновок;

5) провести агрегування;

6) провести дефазифікацію.

Розв'язок

1) Пропозиції, що описують задачу наступні:

• *Якщо успішність студента висока або добра і він розв'язує малу кількість варіантів, то йому потрібно трохи часу.*

• *Якщо успішність студента висока або добра і він розв'язує багато варіантів, то йому потрібно досить великий проміжок часу.*

• *Якщо успішність студента низька і він розв'язує багато варіантів, то йому потрібно багато часу.*

• *Якщо успішність студента середня і він розв'язує досить велику кількість варіантів, то йому потрібно досить великий проміжок часу.*

Виділити з цих пропозицій лінгвістичні змінні (визначимо їх через формальний запис <β, T, X, G, M>, де β – найменування лінгвістичної змінної, T – множина її значень (терм-множина), що представляє собою найменування нечітких змінних, областю визначення кожної з яких є множина X (множина T називається базовою терм-множиною лінгвістичної змінної), G – синтаксична процедура, що дозволяє оперувати елементами терм-множини T, зокрема, генерувати нові терми (значення), M – семантична процедура, що дозволяє перетворити кожне нове значення лінгвістичної змінної, утворене процедурою G, в нечітку змінну, тобто сформувати відповідну нечітку множину).

Нечітка змінна характеризується трійкою <α, X, A>, де α – найменування змінної, X – універсальна множина (область визначення α, A – нечітка множина на X, що описує обмеження (тобто $\mu_A(x)$) на значення нечіткої змінної α.):

β = *успішність студента, T = («висока», «середня», «низька»), X = [2,5] (використовується п'ятибальна система), G = («дуже низька», «висока або середня »), M – зменшення на одиницю ступеня приналежності нечіткої змінної «висока», операція об'єднання нечітких множин;*

β – *кількість варіантів, T = («мало», «досить», «багато»), X = [1,20] (кількість варіантів 20 в кожній темі), G = («дуже багато», «досить або мало»), M – збільшення на одиницю ступеня приналежності нечіткої змінної «багато», операція об'єднання нечітких множин;*

β – *кількість часу, T = («мало», «досить», «багато»), X = [1,7] (кількість годин на тиждень, приділено предмету вивчення), G = («дуже багато», «досить або мало»), M – збільшення на одиницю ступеня приналежності нечіткої змінної «багато», операція об'єднання нечітких множин;*

Для повного завдання лінгвістичної змінної необхідно визначити нечіткі змінні, що входять в Т:

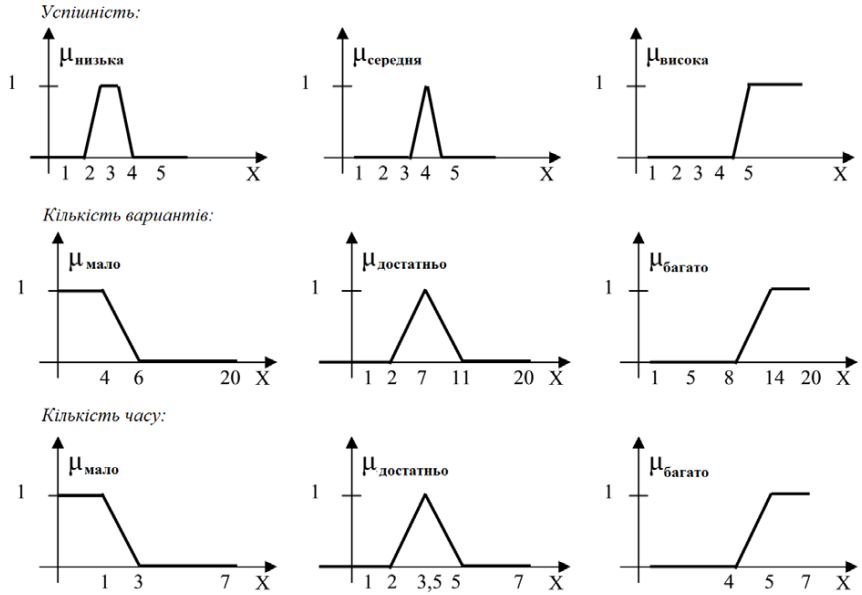


Рис. 48

З урахуванням виділених лінгвістичних змінних, нечіткі правила такі:

1. Якщо Успішність = «висока» або Успішність = «середня» і Кількість варіантів = «мало», то Кількість часу = «мало».

2. Якщо Успішність = «висока» або Успішність = «середня» і Кількість варіантів = «багато», то Кількість часу = «достатньо».

3. Якщо Успішність = «низька» і Кількість варіантів = «багато», то Кількість часу = «багато».

4. Якщо Успішність = «середня» і Кількість варіантів = «достатньо», то Кількість часу = «достатньо».

2) Перевіримо отриману базу на повноту:

• існує хоча б одне правило для кожного лінгвістичного терма вихідної змінної – вихідна змінна «Кількість часу» має 3 терма: «мало» використовується в 1 правилі, «достатньо» – в 2 і 4, «багато» – в третьому;

• для будь-якого терма вхідної змінної є хоча б одне правило, в якому цей терм використовується в якості передумови – є дві вхідних змінних «Успішність» і «Кількість варіантів» у кожній з них 3 терма: «висока» використовується в 1 і 2-му правилі, «середня» – 1, 2 і 4, «низька» – в 3, «мало» – в 1, «достатньо» – 4, «багато» – 3 і 2. Значить, отримана база нечітких правил повна.

3) Нехай є студент Приходько А.А., який має середню оцінку 3.5 і вирішив розв'язувати 9 варіантів. Потрібно визначити скільки йому знадобиться часу.

Визначимо ступінь впевненості найпростіших тверджень:

Успішність = «висока» – 0;

Успішність = «середня» – 0.5;

Успішність = «низька» – 1;

Кількість варіантів = «мало» – 0;

Кількість варіантів = «достаньо» – 0.5;

Кількість варіантів = «багато» – 0.125.

Визначимо ступінь впевненості посилок правил:

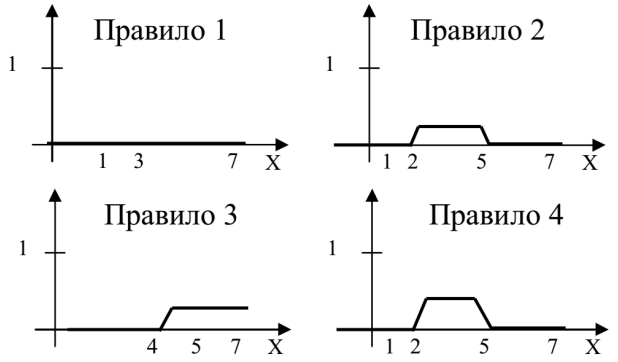
Правило 1: min (max (0,0.5), 0) = 0;

Правило 2: min (max (0,0.5), 0.125) = 0.125;

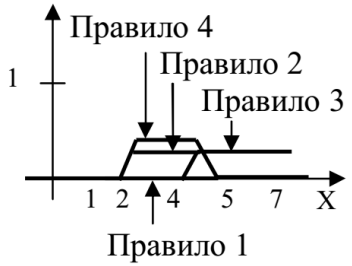
Правило 3: min (1,0.125) = 0.125;

Правило 4: min (0.5,0.5) = 0.5.

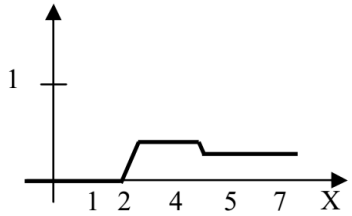
4) Побудуємо нову вихідну нечітку змінну, використовуючи отримані ступені впевненості:



5) Агрегування:



Новий терм вихідної змінної Кількість годин:



6) Виходячи з отриманого графіка ступеня приналежності вихідного терма, можна сказати, що Приходько А.А., має середню оцінку 3.5, на рішення 9 варіантів завдань знадобиться не менше 2.75 години (ступінь впевненості цього твердження 0.5).

5.7. Модифікація алгоритму нечіткого висновку

Алгоритми нечіткого виведення розрізняються в основному видом використовуваного правила нечіткої імплікації. Розглянемо наступні найбільш вживані модифікації алгоритму нечіткого виведення, вважаючи, для простоти, що базу знань організують два нечітких правила виду:

- $П_1$: якщо $x \in A_1$ *і* $y \in B_1$, тоді $z \in C_1$;
- $П_2$: якщо $x \in A_2$ *і* $y \in B_2$, тоді $z \in C_2$;

де x і y – імена вхідних змінних, z – ім'я змінної виводу, A_1, A_2, B_1, B_2, C_1, C_2 – деякі нечіткі множини, задані функціями приналежності $\mu_{A_1}(x)$, $\mu_{A_2}(x)$, $\mu_{B_1}(y)$, $\mu_{B_2}(y)$, $\mu_{C_1}(z)$, $\mu_{C_2}(z)$, При цьому чітке значення z_0 необхідно визначити на основі наведеної інформації та чітких значень x_0 та y_0.

Алгоритм Мамдані

Одним з поширених алгоритмів нечіткого висновку є алгоритм Мамдані (Mamdani). У ситуації, що розглядається він математично може бутиописаний таким чином, рис. 49:

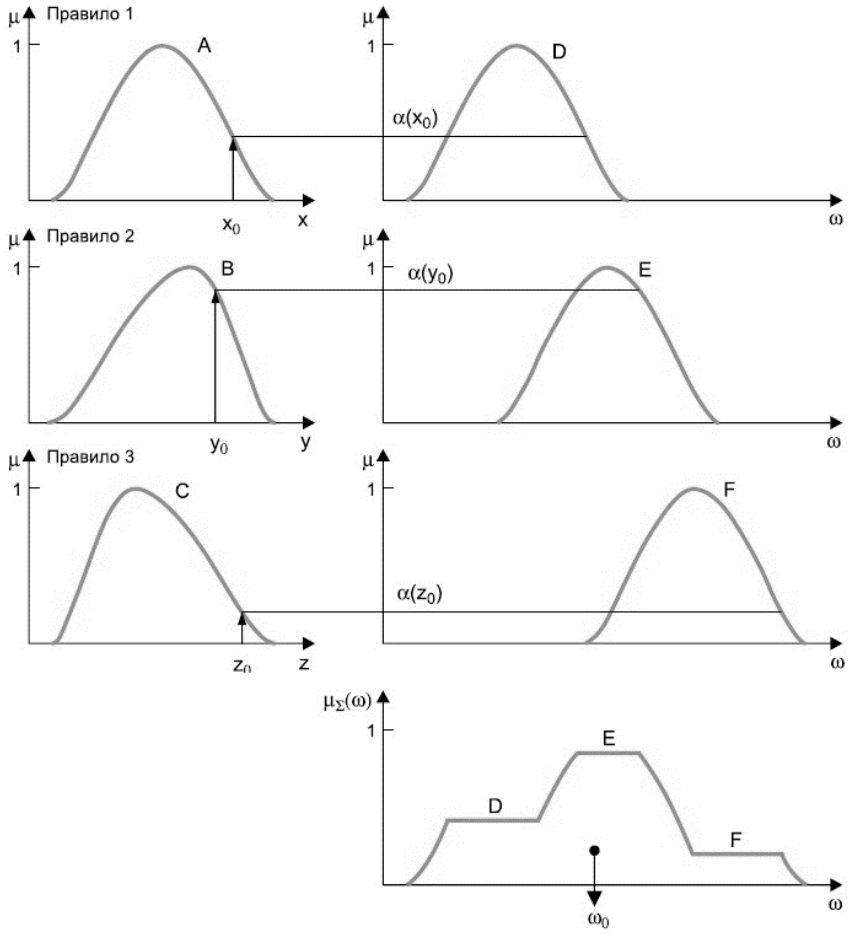


Рис. 49. Ілюстрація алгоритму Мамдані (для трьох правил)

1. Нечіткість: знаходяться ступеня істинності для передумов кожного правила:

$$\mu_{A_1}(x_0),\ \mu_{A_2}(x_0),\ \mu_{B_1}(y_0), \qquad (\ )$$

2. Нечіткий висновок: знаходяться рівні «відсікання» для передумов кожного з правил (з використанням операції МІНІМУМ):

$$\alpha_1 = \mu_{A_1}(x_0) \wedge \mu_{B_1}(y_0)$$

$$\alpha_2 = \mu_{A_2}(x_0) \wedge \mu_{B_2}(y_0)$$

де через «∧» позначена операція логічного мінімуму (min), потім знаходяться «усічені»функції приналежності:

$$\mu_{C_1'}(z) = \left(\alpha_1 \wedge \mu_{C_1}(z)\right)$$

$$\mu_{C_2'}(z) = \left(\alpha_2 \wedge \mu_{C_2}(z)\right)$$

3. Композиція: з використання операції МАКСИМУМ (max, далі позначається як «∨») проводиться об'єднання знайдених усічених функцій, що призводить до отримання підсумкової нечіткої підмножини для змінної виходу з функцією приналежності:

$$\mu_\Sigma(z) = \mu_C(z) = \mu_{C_1'}(z) \vee \mu_{C_2'}(z) = \left(\alpha_1 \wedge \mu_{C_1}(z)\right) \vee \left(\alpha_2 \wedge \mu_{C_2}(z)\right)$$

4. Нарешті, приведення до чіткості (для знаходження z_0) проводиться, наприклад, центроїдним методом.

Алгоритм Сугено

Сугено (Sugeno) і Такагі (Takagi) використовували набір правил в наступній формі (як і раніше, наводимо приклад двох правил):

- $П_1$: якщо x є A_1 *і* y є B_1, тоді $z_1 = a_1 x + b_1 y$;
- $П_2$: якщо x є A_2 *і* y є B_2, тоді $z_2 = a_2 x + b_2 y$;

1. Перший етап – як в алгоритмі Мамдані.

2. На другому етапі знаходяться $\alpha_1 = \mu_{A_1}(x0) \wedge \mu_{B_1}(y0)$, $\alpha_2 = \mu_{A_2}(x0) \wedge \mu_{B_2}(y0)$ та індивідуальні виходи правил:

$$z_1^* = a_1 x_0 + b_1 y_0$$

$$z_2^* = a_2 x_0 + b_2 y_0$$

3. На третьому етапі визначається чітке значення змінної виводу:

$$z_0 = \frac{\alpha_1 z_1^* + \alpha_2 z_2^*}{\alpha_1 + \alpha_2}$$

Ілюструє алгоритм рис. 50.

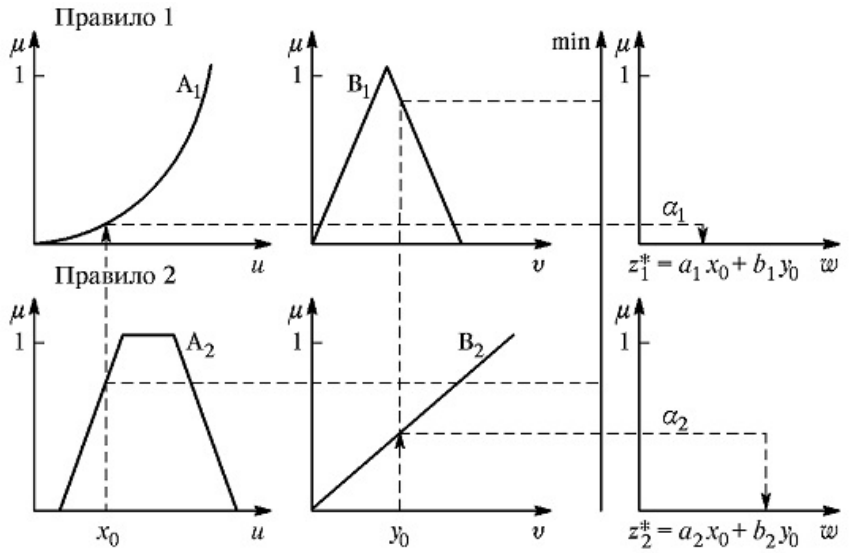


Рис. 50. Ілюстрація алгоритму Сугено

Наведене уявлення відноситься до алгоритму Сугено 1-го порядку. Якщо правила записані в формі

- $П_1$: якщо x є A_1 *і* y є B_1, тоді z_1 є c_1;
- $П_2$: якщо x є A_2 *і* y є B_2, тоді z_2 є c_2;

де c_1, c_2 – деякі дійсні (чіткі) числа, то говорять, що задано алгоритм Сугено 0-го порядку. В цьому випадку

$$z_0 = \frac{\alpha_1 c_1 + \alpha_2 c_2}{\alpha_1 + \alpha_2}$$

5.8. Методи приведення до чіткості

Розглянемо методи приведення до чіткості.

1. Вище вже було розглянуто один з даних методів – *центроїдний* (centroid of area). Центроид площі або центр ваги розраховується за формулами: для безперервного варіанту

$$z_0 = \frac{\int_{z_{min}}^{z_{max}} z\mu_c(z)dz}{\int_{z_{min}}^{z_{max}} \mu_c(z)dz}$$

де $z_{\min}$, $zmax$ – границі області визначення $\mu_{C(z)}$;

для дискретного варіанту:

$$z_0 = \frac{\sum_{i=1}^{n} \alpha_i z_i}{\sum_{i=1}^{n} \alpha_i}$$

2. *Метод центру площі* (bisector of area). Центр площі z_0 визначається з рівняння:

$$\int_{z_{min}}^{z_0} \mu_C(z)dz = \int_{z_0}^{z_{max}} \mu_C(z)dz$$

Іншими словами, центр площі дорівнює абсцисі, яка ділить площу, обмежену графіком кривої функції приналежності відповідної вихідної змінної, на дві рівні частини. Іноді центр площі називають бісектрісоюй площі.

3. *Метод лівого модального значення* (smallest of maximum, SOM) розраховується за формулою:

$$z_0 = min\{z_m\}$$

де z_m – модальне значення (мода – будь-яка точка максимуму функції, в даному випадку, функції $\mu_{C(z)}$ нечіткої множини, відповідного вихідній змінній після проведення етапу агрегування.

Іншими словами, значення вихідної змінної визначається як мода нечіткої множини для відповідної вихідної змінної або найменша з мод (найлівіша), якщо нечітка множина має кілька модальних значень.

4. *Метод правого модального значення* (largest of maximum, LOM) розраховується за формулою:

$$z_0 = max\{z_m\}$$

У цьому випадку значення вихідної змінної також визначається як мода нечіткої множини для відповідної вихідної змінної або найбільша з мод (сама права), якщо нечітка множина має кілька модальних значень.

5. *Метод середнього модального значення* (mean of maximum, MOM). В даному методі чітке значення z_0 визначається як середнє модальних значень відповідної вихідної змінної. Ілюстрація до методів приведення до чіткості приведена на рис. 51.

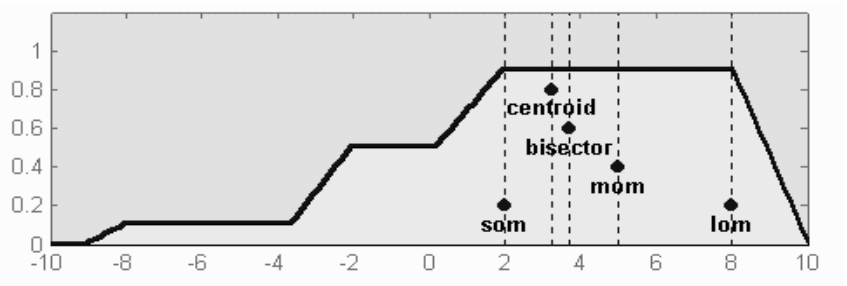


Рис. 51. Методи приведення до чіткості

5.9. Завдання для самоконтролю

Оберіть правильні відповіді.

1. Хто заклав основи теорії нечітких множин?

А) І. Мамдані
Б) М. Блек
В) Л. Заде
Г) Б. Коско
Д) немає правильної відповіді

2. Яких значень може набувати функція приналежності?

А) [0, ∞]
Б) [-∞, +∞]
В) [0, 1]
Г) немає правильної відповіді

3. Множина точок, для яких значення функція приналежності дорівнює 1 називається:

А) носієм
Б) ядром
В) α -зрізом
Г) немає правильної відповіді

4. Яка формула визначає об'єднання нечітких множин A і B?

А) $\min\{1,\mu_A(x)+\mu_B(x)\}$

Б) $\mu_A(x)+\mu_B(x)-\mu_A(x)\cdot\mu_B(x)$

В) $\max\{0,\mu_A(x)+\mu_B(x)-1\}$

Г) $\max\{\mu_A(x),\mu_B(x)\}$

Д) немає правильної відповіді

5. У разі обмежених операцій не виконуватимуться:

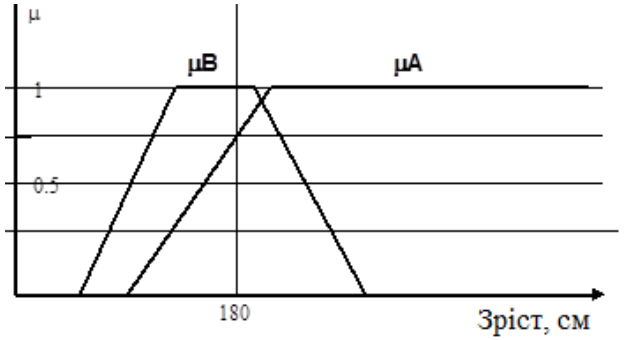
А) $A\cap\overline{A}\neq 0, A\cup\overline{A}\neq U$

Б) $A\cup A\neq A, A\cap A\neq A$

В) $A\cup(B\cap C)\neq(A\cap B)\cup(A\cap C), A\cap(B\cup C)\neq$
$\neq(A\cup B)\cap(A\cup C)$

Г) немає правильної відповіді

6. На рис. показані графіки функцій приналежності нечітких множин – «Високий зріст» та «Середній зріст».



Визначити функцію приналежності людини зі зростом 180 см до першої (/180) та другої (/180) множинам:

А) $\mu_A/180=\mu_B/180=\min\{0.75; 1\}$

Б) $\mu_A/180=\mu_B/180=\max\{0.75; 1\}$

В) $\mu_A/180=\mu_B/180=0.5*(\mu_A/180+\mu_B/180)=0.875$

Г) $\mu_A/180=0.75$, $\mu_B/180=1$

Д) немає правильної відповіді

7. Нехай $\mu_A(x)$, $\mu_B(x)$ – функції приналежності нечітких множин A і B на універсальній множині E. Нехай також C – нечітка множина з функцією приналежності $\mu_C(x)$, яка є об'єднанням A і B. Визначити значення приналежності $x\in$E нечіткій множині C, якщо $\mu_A(x)=0.5$ і $\mu_B(x)=0$:

А) $\mu_C(x)=\max\{\mu_B(x), \mu_A(x)\}=0.5$

Б) $\mu_C(x)=\min\{\mu_B(x), \mu_A(x)\}=0$

В) $\mu_C(x)=1-\min\{\mu_B(x), \mu_A(x)\}=1$

Г) немає правильної відповіді.

8. Нехай $\mu_A(x)$, $\mu_B(x)$ – функції приналежності нечітких множин A і B на універсальній множині E. Нехай також C – нечітка множина з функцією приналежності $\mu_C(x)$, яка являється перетином A і B. Визначити значення приналежності $x\in$E нечіткій множині C, якщо $\mu_A(x)=0.5$ и $\mu_B(x)=0$:

А) $\mu_C(x)=\max\{\mu_B(x), \mu_A(x)\}=0.5$

Б) $\mu_C(x)=\min\{\mu_B(x), \mu_A(x)\}=0$

В) $\mu_C(x)=1-\max\{\mu_B(x), \mu_A(x)\}=0.5$

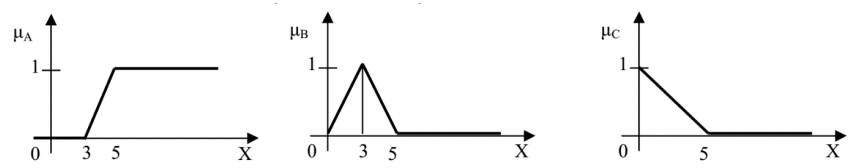
Г) $\mu_C(x)=1-\min\{\mu_B(x), \mu_A(x)\}=1$

Д) немає правильної відповіді.

5.10. Завдання для самостійної роботи

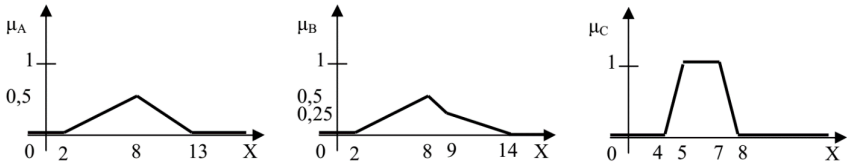
Операції над нечіткими множинами

1) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D=A\cup B\cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи максимінний спосіб.

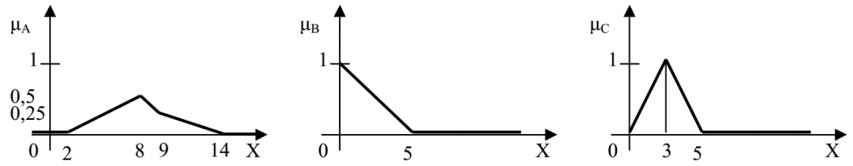


2) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

$D = A \cup B \cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи алгебраїчний спосіб.



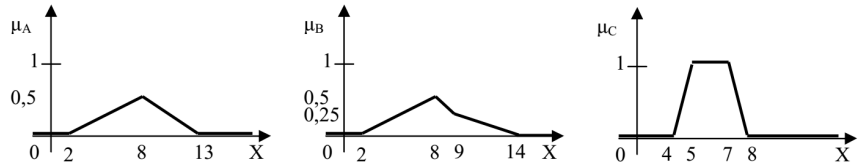
3) Дано 3 нечітких безлічі A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = A \cup B \cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.



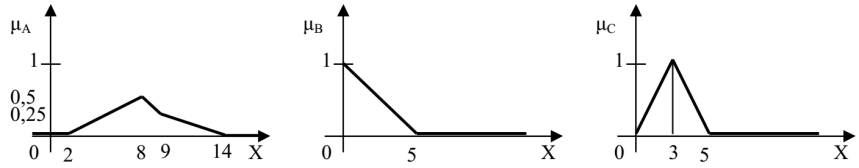
4) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = A \cup \overline{B} \cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи максиміний спосіб.

5) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

$D = A \cup \overline{B} \cap C$ і визначити ступінь приналежності одного елементу множині D, використовуючи алгебраїчний спосіб.



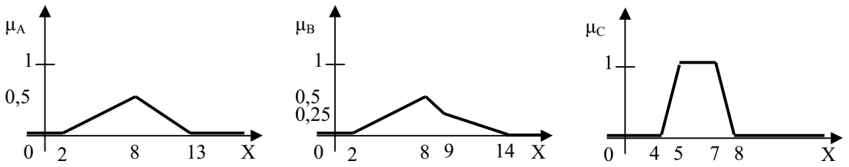
6) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = A \cap B \cup \overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.



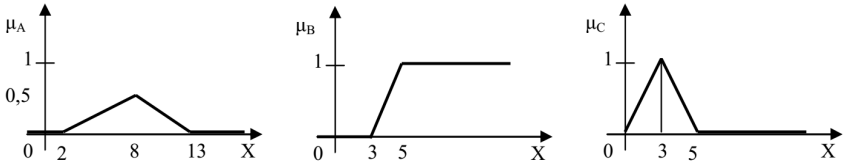
7) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = A \cap B \cup \overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи максимінний спосіб.

8) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

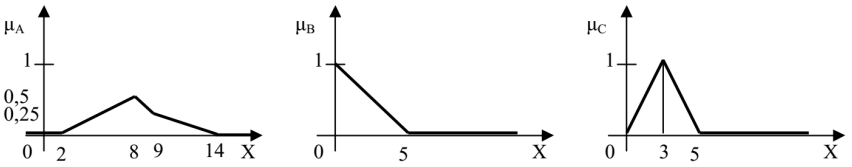
$D = A \cap B \cup \overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи алгебраїчний спосіб.



9) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = A \cap \overline{B} \cup \overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.

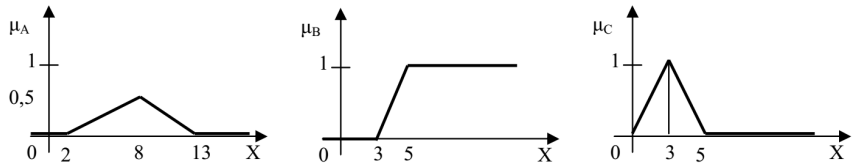


10) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = \overline{A} \cup B \cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи максимінний спосіб.

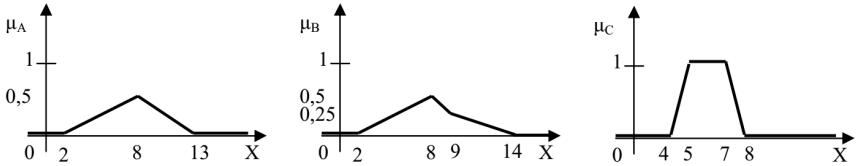


11) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

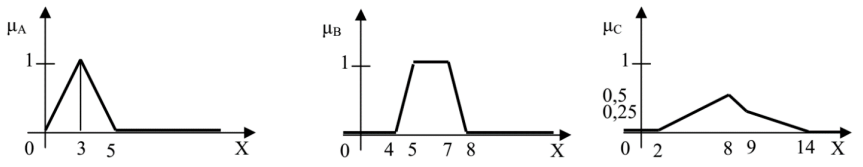
$D = \overline{A} \cup B \cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи алгебраїчний спосіб.



12) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = \overline{A} \cup B \cap C$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.

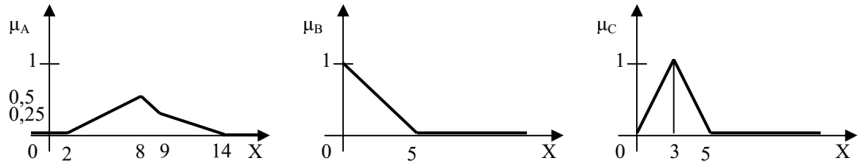


13) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D = \left(\overline{A} \cup B\right) \cap \overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи максимінний спосіб.

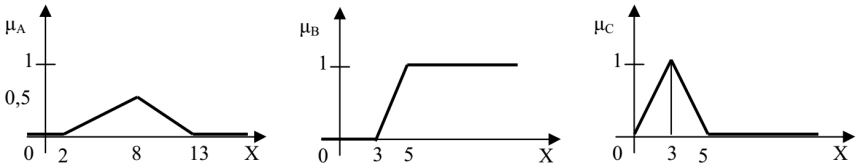


14) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

$D=\left(\overline{A}\cup B\right)\cap\overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи алгебраїчний спосіб.



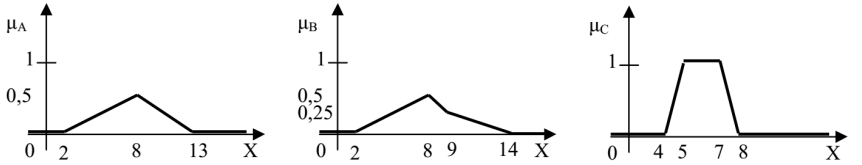
15) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D=\left(\overline{A}\cup B\right)\cap\overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.



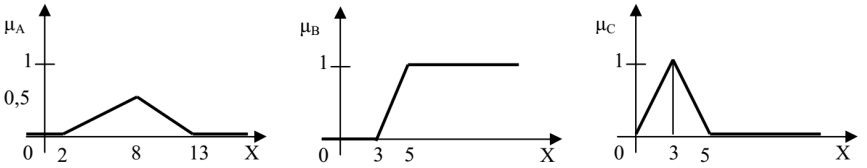
16) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D=\overline{A}\cap\left(C\cup B\right)\cap\overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи максимінний спосіб.

17) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

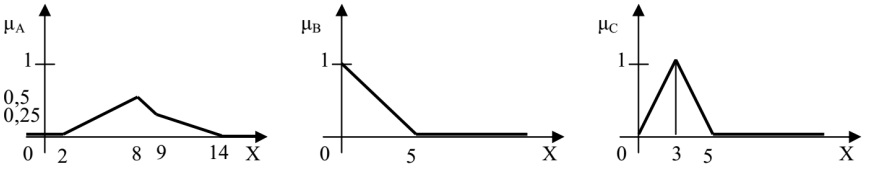
$D=\overline{A}\cap\left(C\cup B\right)\cap\overline{C}$ і визначити ступінь приналежності одного елемента множини D, використовуючи алгебраїчний спосіб.



18) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D=\overline{A}\cap\left(C\cup B\right)\cap\overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.

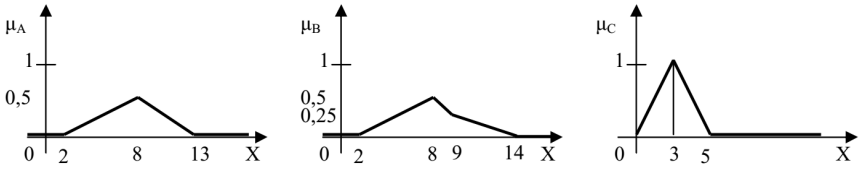


19) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини $D=\overline{A}\cup\overline{B}\cap\overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи алгебраїчний спосіб.



20) Дано 3 нечітких множини A, B, C (задані їх функції приналежності). Побудувати функцію приналежності нечіткої множини

$D = \overline{A} \cup \overline{B} \cap \overline{C}$ і визначити ступінь приналежності одного елемента множині D, використовуючи метод обмежень.



Нечіткий висновок

1. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання закупівель (співвідношення ціни, якості, обсягу закупівель і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

2. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання розподілу навантажень спортсмена (співвідношення навантажень, фізичного стану, споживаних калорій і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

3. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання керування транспортним засобом (регулювання швидкості з урахуванням передачі, погодних умов, інтенсивності потоку і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

4. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання керування транспортним засобом (управління кермом, газом, гальмами при в'їзді на парковку), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

5. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання регулювання теплопостачанням (співвідношення середньодобової температури, вітру, розміру будівлі і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

6. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання регулювання реверсного руху на мості (враховувати час, інтенсивність потоку, день тижня і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

7. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання підбору спецій для страви (співвідношення кількості і гостроти спецій, рецептури, переваг їдця, обсягу їжі і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

8. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання підбору кількості страв (враховувати калорійність, смакові переваги, кількість їдців і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

9. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання подачі електроенергії в умовах економії (облік часу доби, типу приміщень, кількості людей, типу обладнання і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

10. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання підбору інтенсивності занять (враховувати початковий рівень підготовки, обсяг навчального матеріалу, кількість осіб в групі, необхідний рівень засвоєння і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

11. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання розрахунку споживання бензину (враховувати тип здійснюваних маневрів, рівень підготовки водія, стан автомобіля, тип автомобіля і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

12. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання регулювання системи зрошення (враховувати пору року, кількість випадаючих осадків, вид зрошуваної культури і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

13. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання налаштування аудіосистеми (потужність колонок, їх кількість, розмір приміщення, призначення установки і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

14. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання вибору дози снодійного (кількість препарату, дія препарату, сприйнятливість до вибраного препарату, мета і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

15. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання планування обсягу виробництва продукції (з урахуванням можливого прибутку, необхідних ресурсів, платоспроможності населення, ринку збуту і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

16. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання регулювання кондиціонера (враховувати його потужність, обсяг приміщення, температуру навколишнього середовища, необхідну температуру в приміщенні і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

17. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання розподілу навантаження між комп'ютерами при використанні їх в кластерах (враховувати характеристики комп'ютерів, їх кількість, кількість паралельного коду, характеристики мережі і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

18. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання вибору складського приміщення (враховувати площу складу, кількість і розміри продукції, віддаленість від місця виробництва і точок реалізації, якості продукції і характеристики приміщень і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

19. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для завдання вибору комплектуючих для

комп'ютера (враховувати ціну, потреби користувача, сумісність, терміни використання і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

20. Побудувати нечітку базу знань (використовувати не менше 3 лінгвістичних змінних) для задачі визначення кількості ліній в службі підтримки (враховувати кількість обслуговуваних клієнтів, середню частоту звернення до служби одного клієнта, середній час обслуговування однієї заявки, кваліфікацію персоналу і т. п.), перевірити її на повноту і зробити нечіткий висновок для конкретних значень (вибрати випадковим чином).

5.11. Лабораторна робота 12. Побудова нечіткої ЕС

Мета: розглянути методику побудови нечіткої експертної системи.

Завдання: Використовуючи вбудовані функції пакета Fuzzy Logic середовища MATLAB, побудувати нечітку експертну систему. ЕС повинна допомогти користувачеві з відповіддю на питання: скільки дати «на чай» офіціанту за обслуговування в ресторані?

Теоретичні відомості

Визначення нечіткої множини не накладає ніяких обмежень на вибір конкретної функції приналежності для її представлення. Однак на практиці зручно використовувати ті з них, які допускають аналітичне представлення у вигляді деякої простої математичної функції. Це спрощує не тільки відповідні чисельні розрахунки; але і скорочує обчислювальні ресурси, необхідні для зберігання окремих значень цих функцій приналежності. Необхідність типізації окремих функцій приналежності також обумовлена наявністю реалізацій відповідних функцій в відомих інструментальних засобах (наприклад, в системі MATLAB).

Найчастіше в нечіткій логікі для завдання функцій приналежності використовують такі типові форми наведених нижче функцій (рис. 52)

• трикутна (trimf) $trimf\left(x,a,b,c\right)=max\left(\ min\left(\frac{x-a}{b-a},\frac{c-x}{c-b}\right),0\right);$

• трапецеїдальна (trapmf)

$$trapmf\left(x,a,b,c,d\right)=max\left(\ min\left(\frac{x-a}{b-a},\frac{d-x}{c-b}\right),0\right);$$

• гаусова (gaussmf) $gaussmf\left(x,\sigma,c\right)=e^{-\left(\frac{x-c}{\sigma}\right)^2}$;
• подвійна гаусова (gauss2mf);
• узагальнена колоколообразна (gbellmf)

$$gbellmf\left(x,a,b,c\right)=\frac{1}{1+\left|\frac{x-c}{a}\right|^{2b}};$$

• сигмоїдальна (sigmf) $sigmf\left(x,a,c\right)=\frac{1}{1+\exp\left(-a\left(x-c\right)\right)}$;

• подвійна сигмоїдальна (dsigmf);
• Z-функція;
• S-функція;
• P_i-функція.

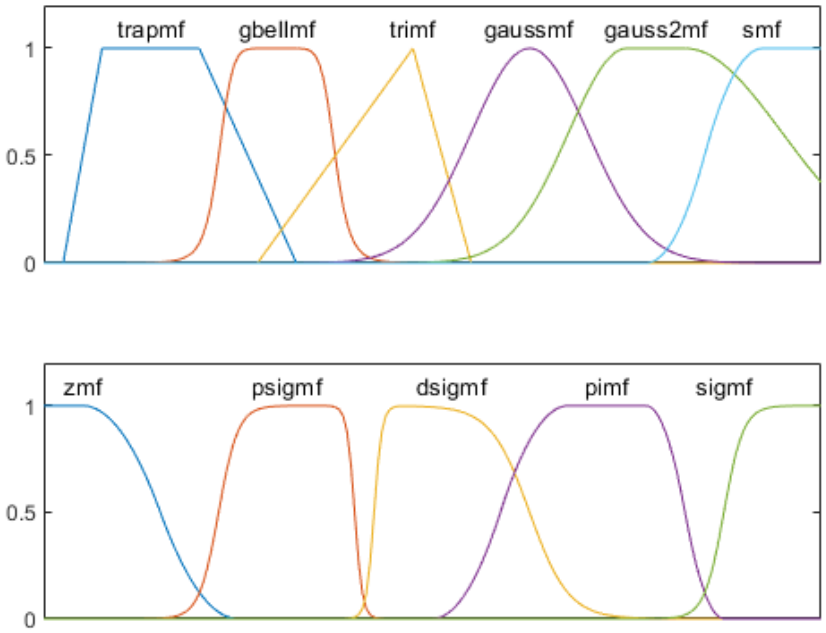


Рис. 52. Типові функції приналежності в MATLAB

Практична частина

Ґрунтуючись на якихось усталених звичаях і інтуїтивних уявленнях, приймемо, що задача про чайові може бути описана наступними пропозиціями:

- якщо обслуговування погане або їжа підгоріла, то чайові – малі;
- якщо обслуговування хороше, то чайові – середні;
- якщо обслуговування відмінне або їжа – чудова, то чайові – щедрі.

Якість обслуговування та їжі будемо оцінювати по 10-бальній системі (0 – найгірша оцінка, 10 – найкраща).

Припустимо, що малі чайові складають близько 5% від вартості обіду, середні – близько 15% і щедрі – приблизно 25%.

Зауважимо, що представленої інформації, в принципі, досить для проектування нечіткої експертної системи. Така система буде мати 2 входи (які умовно можна назвати «service» («сервіс») та «food» («їжа»)), один вихід «tip» («чайові»), три правила типу «якщо ... то» (відповідно до трьох наведених пропозицій) і по три значення (відповідно 0 балів, 5 балів, 10 балів та 5%, 15%, 25%) для центрів функцій приналежності входів і виходу. Побудуємо цю систему, використовуючи алгоритм виводу Mamdani, описуючи необхідні дії по пунктам.

1. Командою fuzzy запускаємо FIS-редактор (в командному рядку MATLAB). За замовчуванням пропонується алгоритм виведення типу Мамдані (про що говорить напис в центральному білому блоці) і тут ніяких змін не потрібно, але в системі повинно бути два входи, тому через пункт меню Edit → Add Variable→Input додаємо в систему цей другий вхід (в вікні редактора з'являється другий жовтий блок з ім'ям input2). Роблячи, далі, одноразове натискання по блоку inputl, міняємо його ім'я на «service» («сервіс», «якість обслуговування»), завершуючи введення нового імені натисканням клавіші Enter. Аналогічним чином встановлюємо ім'я «food» («їжа») блоку input2 і «tip» («чайові») – вихідного блоку (справа вгорі) output1. Дамо відразу ж і ім'я всій системі, наприклад, «tip» («чайові»), виконавши це через пункт меню File → Export → To Workspace (Зберегти в робочому просторі). Вид вікна редактора після зазначених дій наведено на рис. 53.

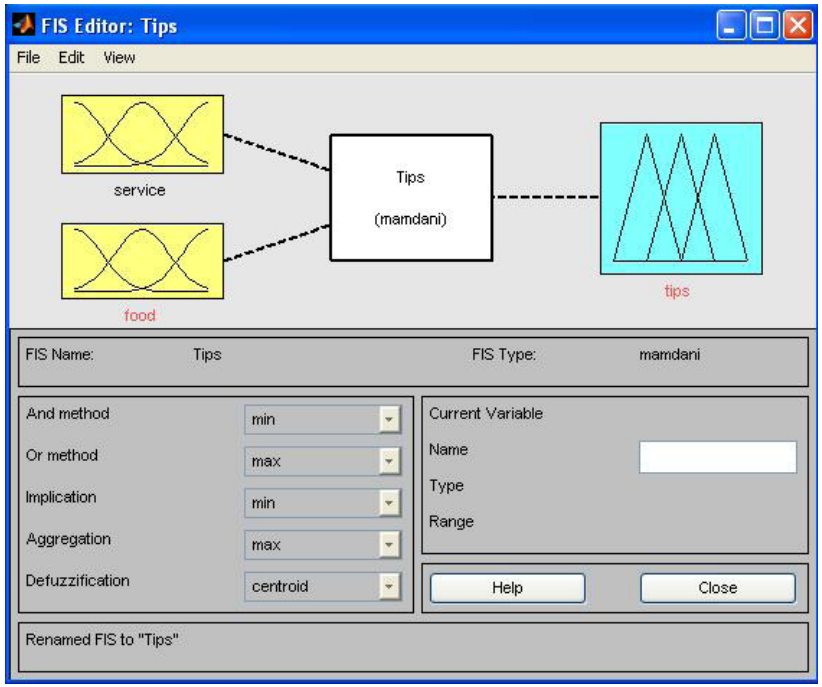


Рис. 53. Вид вікна FIS-редактора

2. Задамо функції приналежності змінних. Зазначимо, що програму-редактор функцій приналежності можна відкрити трьома способами:

1) через пункт меню Edit → Membership functions;

2) подвійним натисканням на значок, що відображає відповідну змінну;

3) натисканням клавіш Ctrl + 2.

Будь-яким з наведених способів перейдемо до даної програми.

Завдання і редагування функцій приналежності почнемо зі змінної «service». Спочатку в полях Range і Display Range встановимо діапазон зміни і відображення цієї змінної – від 0 до 10 (балів), підтверджуючи введення натисканням клавіші Enter. Потім через пункти меню Edit → Remove All MFs, Edit → Add MFs перейдемо до діалогового вікна і поставимо в ньому три функції приналежності гауссова типу (gaussmf). Натиснемо кнопку OK і повернемося у вікно редактора

функцій належності. Не зважаючи на положення заданих функцій, замінимо тільки їх імена на «poor» («поганий»), «good» («добрий») та «excellent» («відмінний»).

3. Натисканням на значок «food» увійдемо в вікно редагування функцій приналежності для цієї змінної. Задамо спочатку діапазон її зміни від 0 до 10, а потім, як раніше, задамо дві функції приналежності трапецеподібні (trapmf) з параметрами відповідно [0 0 1 3] і [7 9 10 10] та іменами «rancid» («підгоріла») і« delicious »(«чудова»).

4. Для вихідної змінної «tip» («чайові») вкажемо спочатку діапазон зміни (від 0 до 30), потім поставимо три функції приналежності трикутної форми з іменами «small» («малі»), «middle» («середні») та «large» («великі») так, як це представлено на рис. 38. Зауважимо, що можна, зрозуміло, поставити і будь-які інші функції або вибрати їх інші параметри.

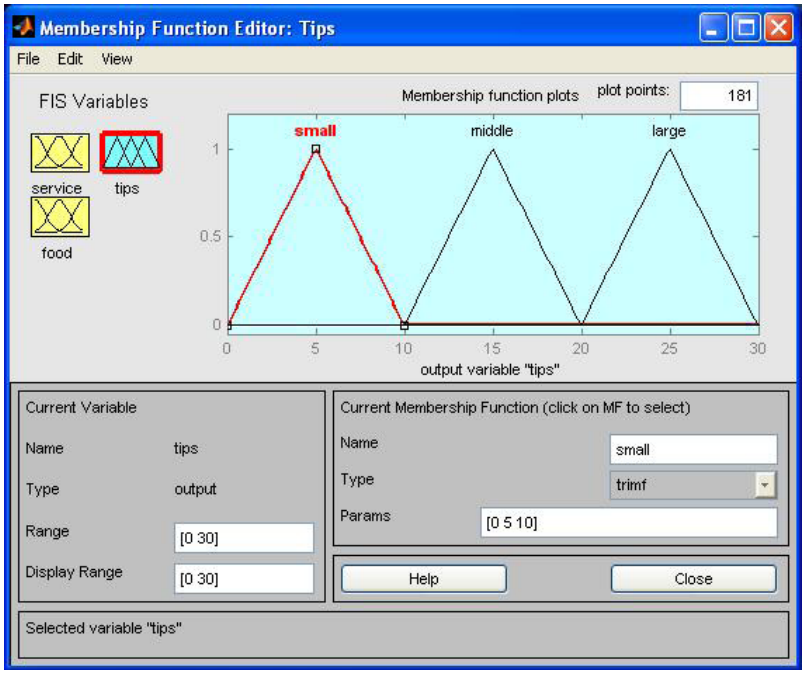


Рис. 54. Функції приналежності після завдання структури системи змінної «tip» («чайові»)

5. Перейдемо до конструювання правил. Для цього виберемо пункт меню Edit → Rules. Далі введення правил проводиться таким чином:

При введенні кожного правила необхідно позначити відповідність між кожною функцією приналежності аргументу *x* і числовим значенням *y*. Тому, вибираємо в лівому полі (з заголовком *x* is) варіант вашої кривої, а в правому чому вона дорівнює і натискаємо кнопку Add rule. Аналогічно поступимо так з усіма іншими значеннями. Зауважимо, що в першому і третьому правилах як «зв'язки» в передумовах правила необхідно використовувати не «І» (and), а «АБО» (or); при введенні другого правила, де відсутня змінна «їжа» («food»), для неї вибирається опція none. Підсумковий набір правил відображений на рис. 55.

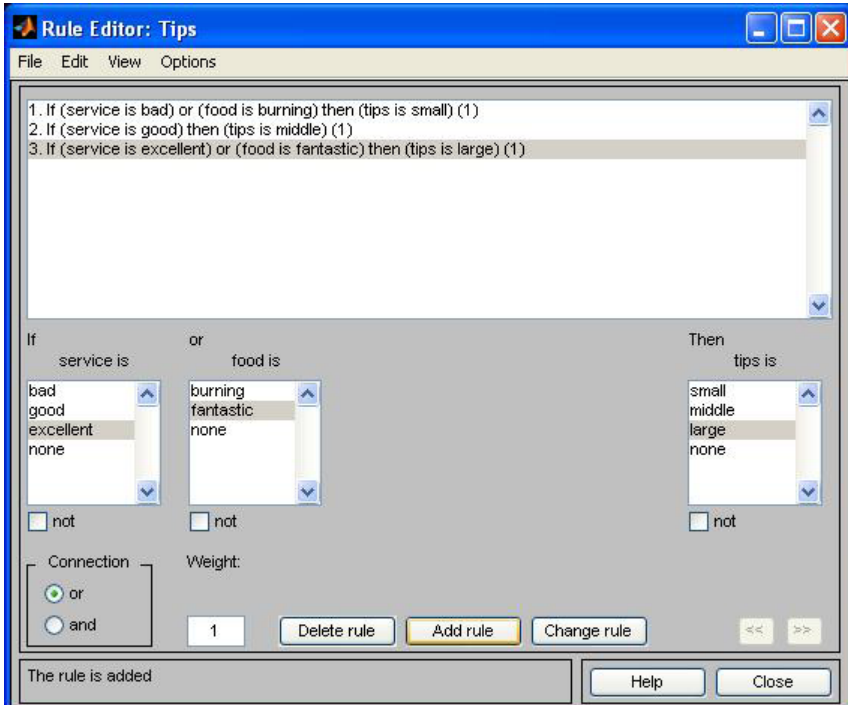


Рис. 55. Підсумковий набір правил

Такий детальний (verbose) запис представляється досить зрозумілою; одиниця в дужках після кожного правила вказує на йогу «вагу» (Weight), тобто значимість правила. Дану вагу можна міняти, використовуючи відповідне поле в лівій нижній частині вікна редактора правил. Правила, що представлені і в інших формах: символічній (symbolic) та індексній (indexed), при цьому перехід від однієї форми в іншу відбувається за допомогою меню Options → Format редактора правил.

Ось як виглядають розглянуті правила в символічній формі:

1. (Service == poor) | (Food == rancid) => (tip = small) (l).
2. (Service == good) => (tip = middle) (1).
3. (Service == excellent) | (Food == delicious) => (tip = large) (1).

Правила в індексній формі (самий стислий формат представлення правил) є тим форматом, який в дійсності використовується програмою. У цьому форматі наведені правила виглядають наступним чином:

1 1, 1 (1): 2
2 0, 2 (1): 2
3 2, 3 (1): 2

Тут перша колонка відноситься до першої вхідної змінної (відповідно перше, друге або третє можливе значення), друга – до другої, третя (після коми) – до вихідної змінної, цифра в дужках показує вагу правила і остання цифра (після двокрапки) вказує тип «зв'язки»(1 для «І», 2 для «АБО»).

6. На цьому, власне, конструювання експертної системи закінчено. Збережемо її на диску під обраним ім'ям (tip).

7. Тепер саме час перевірити систему в дії. Відкриємо (через пункт меню View → Rules) вікно перегляду правил і встановимо значення змінних: «service» (сервіс) = 0 (тобто поганий), «food» (їжа) = 10 (тобто чудова). Побачимо відповідь: «tip» (чайові) = 15 (тобто середні). Ну що ж, з системою не посперечаєшся, треба платити (рис. 56).

Можна перевірити й інші варіанти. Зокрема, з'ясується, що нашою системою «обслуговування» цінується більше, ніж «якість» їжі: при наборі «service = 10, food = 3» система радить визначити розмір чайових в 23.9%, в той час як набору «service = 3, food = 10» розмір чайових за рекомендацією системи – 16.6% (від вартості обіду). Втім, нічого дивного тут немає: це ми самі (не дуже підозрюючи про це) заклали в систему відповідні знання у вигляді сукупності наведених правил.

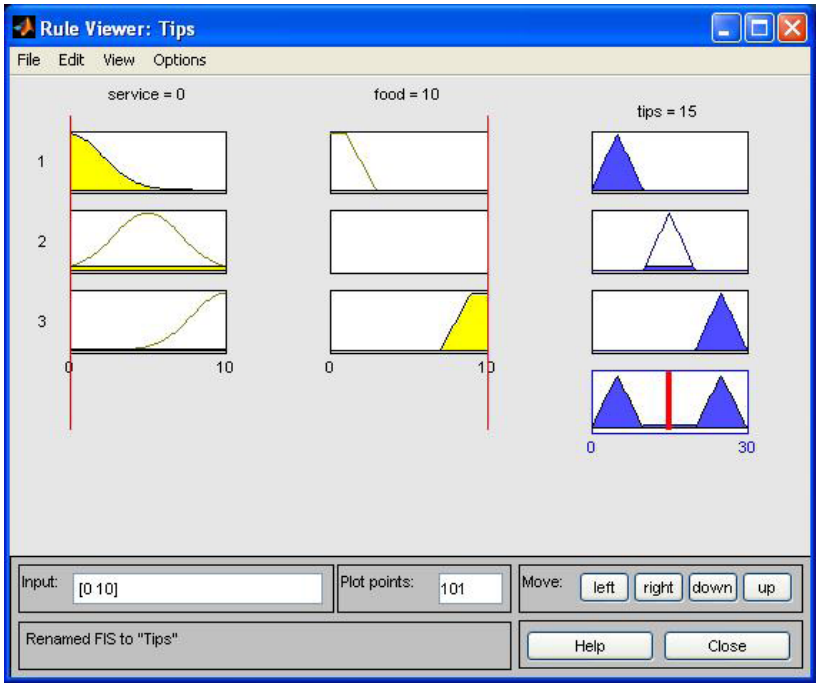


Рис. 56. Вікно перегляду правил

Підтвердженням зазначеної залежності вихідної змінної від вхідних може служити вигляд поверхні відгуку, який видається при виборі пункту меню View → Surface (рис. 57); зверніть увагу, що за допомогою мишки графік можна повертати в різні боки.

У вікні, змінюючи імена змінних в полях введення (X (input) і Y (input)) можна задати і перегляд одновимірних залежностей, наприклад «чайових» («tip») від «їжі» («food») (рис. 58).

Експорт та імпорт результатів

Коли ви зберігаєте створену вами нечітку систему, використовуючи пункт меню File → Export → To disk, на диску створюється текстовий (ASCII) файл досить простого формату з розширенням .fis. Його можна переглядати, при необхідності редагувати поза системою MATLAB, а також використовувати повторно при наступних сеансах роботи

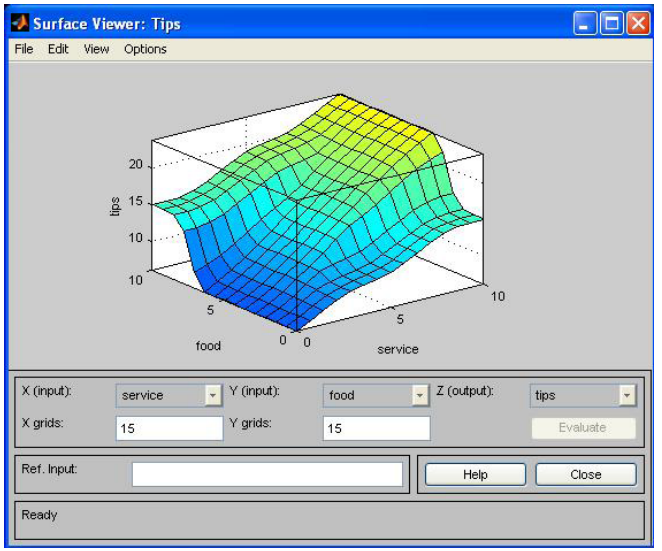


Рис. 57. Графічний вид залежності вихідної змінної від вхідних

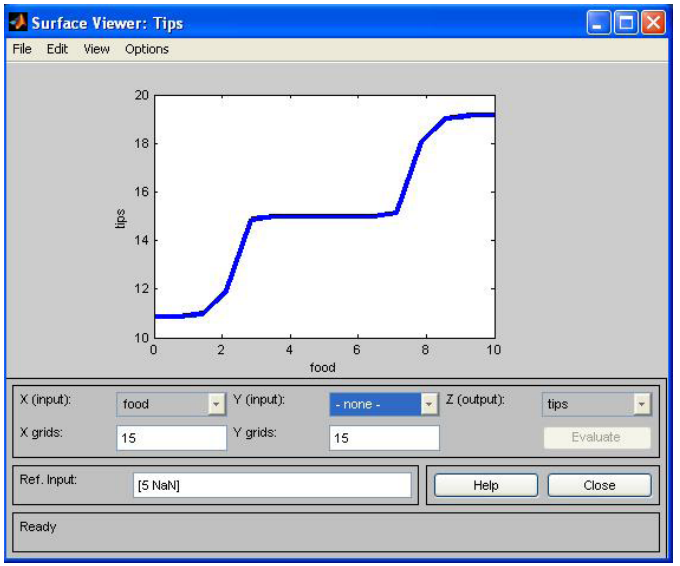


Рис. 58. Одновимірна залежність розміру чайових (tip) від якості їжі (food)

з системою. Однак збереження з використанням пункту File → Export → To Workspace насправді тільки «легалізує» створену вами систему (під будь-яким ім'ям) в середовищі MATLAB протягом поточного сеансу роботи і не допускає її повторного використання в інших сеансах.

Створення своїх функцій приналежності

Якщо з яких-небудь причин вас не влаштовує жодна з вбудованих функцій приналежності, ви можете створити і використовувати власну відповідну функцію. Така функція повинна бути створена як m-файл, повертати значення в діапазоні від 0 до 1 і мати число аргументів не більше 16. Наведемо етапи створення даної функції під деяким ім'ям custmf.

– Створити відповідний m-файл з ім'ям custmf.m.

– Обирається пункт меню Edit → Add Custom MF (додати призначену для користувача функцію приналежності) в меню редактора функцій приналежності (при необхідності – спочатку за допомогою команди Edit → Remove All MFs видалити функції приналежності, що задаються за умовчанням). Встановити (в поле Range) діапазон зміни аргументу функції приналежності.

– В поле M-File function name діалогового вікна Add customized membership function ввести ім'я створеного m-файлу (custmf).

– В поле Parameter list даного вікна ввести необхідні числові параметри.

– Нарешті, в поле MF name (ім'я функції приналежності) ввести унікальне ім'я функції (наприклад, custmf).

Зазначені дії підтверджуються натисканням кнопки ОК.

Нижче наведено приклад m-файлу деякої користувальницької функції приналежності дискретного типу, що має ім'я testmfl і залежить від 8 числових параметрів (кожен – з діапазону [0, 10]).

```
function out = testmfl (x, params)
for i = l: length (x)
if x (i) <params (1)
    y (i) = params (1);
elseif x (i) <params (2)
    y (i) = params (2);
elseif x (i) <params (3)
    y (i) = params (3);
elseif x (i) <params (4)
    y (i) = params (4);
elseif x (i) <params (5)
    y (i) = params (5);
elseif x (i) <params (6)
    y (i) = params (6);
elseif x (i) <params (7)
    y (i) = params (7);
elseif x (i) <params (8)
    y (i) = params (8);
else
    y (i) = 0;
 end
end
out = .l * y ';
```


Завдання до лабораторної роботи 12

1. Розробіть нечітку експертну систему за обраною темою. В системі нечіткого висновку повинно бути не менше трьох вхідних змінних та п'яти правил.
2. Створіть свою функцію приналежності.

Література до розділу

1. Збірник завдань з курсу «Інтелектуальні інформаційні системи» навчальний посібник / С.В. Ліпатова. – Ульяновськ: УлГУ, 2010. – 64 с. (рос.)

2. Інтелектуальні інформаційні системи: навчальний посібник / А. А. Смагін, С. В. Ліпатова, О. С. Мельниченко. – Ульяновськ: УлГУ, 2010. – 136 с. (рос.)

3. Д'яконов В.П. MATLAB 6.5 SP1 7/7 SP1/7 SP1 + Simulink 5/6. Інструменти штучного інтелекту та біоінформатики / Д'яконов В.П., Круглов В.В. // Серия «Библиотека професионала». – М.: СОЛОН – ПРЕСС, 2006. – 456 с.: ил.

4. Зосімов В.В., Погромська Г.С., Махровська Н.А., Булгакова О.С. Інформаційний аналіз управління процесами складних системах:

теорія та практика: [монографія] – Миколаїв, МНУ ім. В. О. Сухомлинського, 2018 – 217 с.

5. Паклін Н. Нечітка логіка – математичні основи [Електронний ресурс] / Н. Паклін. – Режим доступу: http://www.basegroup.ru/library/analysis/fuzzylogic/math/

6. Півкін, В. Я. Нечіткі множини в системах управління: навч. посібник [Електронний ресурс] / В. Я. Півкін, Е. П. Бакулін, Д. І. Кореньков; під ред. проф. Ю. М. Золотухіна. – Режим доступу: http://www.vevivi.ru/best/Nechetkie-mnozhestva-v-sistemakh-upravleniya-ref41397.html

7. Штовба, С. Д. Введення в теорію нечітких множин і нечітку логіку [Електронний ресурс] / С. Д. Штовба. – Режим доступу: http://MATLAB.exponenta.ru/fuzzylogic/book1/index.php

РОЗДІЛ VI. ЕВОЛЮЦІЙНЕ МОДЕЛЮВАННЯ

6.1. Еволюційне моделювання

Однією з головних характеристик штучного інтелекту як науки є його міждисциплінарність, що дозволяє залучати цікаві ідеї, теорії з інших галузей знань, адаптуючи і використовуючи готові розробки для своїх завдань. Так було з нейронними мережами, з комп'ютерною лінгвістикою, тощо. Багато значимих теорій науки було так чи інакше розглянуто через призму штучного інтелекту. Теорія Ч. Дарвіна (1859 р.) стала відправною точкою для ще одного напряму досліджень – еволюційного моделювання.

Основна теза еволюційного моделювання – замінити процес моделювання складного об'єкту моделюванням його еволюції. Він спрямований на застосування механізмів природної еволюції при синтезі складних систем обробки інформації. Дарвін сформулював основний закон розвитку органічного світу, охарактеризувавши його взаємодією трьома наступними чинниками:

– спадковість (нащадки зберігають властивості батьків);
– мінливість (нащадки майже завжди не ідентичні);
– природний відбір (виживають найбільш пристосовані).

Теорія Дарвіна, доповнена генетичними знаннями, називається синтетичною теорією еволюції. Випадкову появу нових ознак вона пояснила мутаціями – змінами, що виникають в ДНК організмів.

Поняття «Еволюційне моделювання» сформувалося в роботах Л. Фогеля, А. Оуена, М. Уолша. У 1966 році вийшла їх спільна книга «Штучний інтелект та еволюційне моделювання». Історія еволюційних обчислень розпочалася з розробки ряду різних незалежних моделей. Основними з них були генетичні алгоритми (ГА) і класифікаційні системи Д. Голланда (Holland), що опубліковані на початку 60-х років і отримали загальне визнання після виходу у світ книги, що стала класикою в цій області, «Адаптація в природних і штучних системах» (Adaptation in Natural and Artifical Systems, 1975). У 70-х роках у рамках теорії випадкового пошуку Л. Растригіним був запропонований ряд алгоритмів, що використовують ідеї біонічної поведінки індивідуумів. Розвиток цих ідей знайшов відображення в циклі робіт І. Л. Букатової

по еволюційному моделюванню. Розвиваючи ідеї М. Л. Цетліна про доцільну і оптимальну поведінку стохастичних автоматів, Ю. І. Неймарк запропонував здійснювати пошук глобального екстремуму на основі колективу незалежних автоматів, що моделюють процеси розвитку і елімінації індивідуумів. Незважаючи на різницю в підходах, усі вони базувалися на принципах еволюції.

У рамках еволюційного моделювання створювалися і досліджувалися моделі походження молекулярно-генетичних систем обробки інформації, моделі, що характеризують загальні закономірності еволюційних процесів, і робився аналіз моделей штучної «еволюції» з метою застосування методу еволюційного пошуку до практичних завдань оптимізації.

На початку 70-х років лауреат Нобелівської премії М. Ейген зробив вражаючу спробу побудови моделей виникнення в ранній біосфері Землі молекулярно-генетичних систем обробки інформації (у моделі квазівидів розглядається поетапна еволюція популяції інформаційних послідовностей (векторів), компоненти яких приймають невелике число дискретних значень).

Д. Коза розробив метод, що дозволяє удосконалювати реальні технічні системи методами генетичного програмування. При цьому еволюція зачіпає не окремі чисельні параметри, а цілі системи (за допомогою цього методу вдалося наново відкрити 15 запатентованих рішень схемотехнік: підсилювачі, фільтри, контролери і т.п.).

Переваги еволюційних обчислень:

1) велика сфера застосування;

2) можливість проблемно-орієнтованого кодування рішень, підбору початкової популяції, комбінування еволюційних обчислень з нееволюційними алгоритмами, продовження процесу еволюції до тих пір, поки є необхідні ресурси;

3) придатність для пошуку в складному просторі рішень великої розмірності;

4) відсутність обмежень на вид цільової функції;

5) ясність схеми і базових принципів еволюційних обчислень;

6) інтегрованість еволюційних обчислень з іншими некласичними парадигмами штучного інтелекту, такими як штучні нейромережі і нечітка логіка.

Недоліки еволюційних обчислень:

1) евристичний характер еволюційних обчислень не гарантує оптимальності отриманого рішення;

2) відносно висока обчислювальна трудомісткість, яка долається за рахунок розпаралелювання на рівні організації еволюційних обчислень і на рівні їх безпосередньої реалізації в обчислювальній системі;

3) відносно невисока ефективність на завершальних фазах моделювання еволюції (оператори пошуку в еволюційних алгоритмах не орієнтовані на швидке попадання в локальний оптимум);

4) невирішеність питань самоадаптації.

До *методів еволюційного моделювання* відносяться:

Метод групового урахування аргументів (був розглянутий в розділі «Нейронні мережі» як один з підвидів поліноміальної НМ).

Еволюційне (генетичне) програмування (дані, які закодовані в генотипі, можуть бути командами якої-небудь віртуальної машини, в простому випадку нічого не міняється в генетичному алгоритмі, але тоді довжина отримуваної послідовності дій (програми) виходить такою, що не відрізняється від початкових; сучасні алгоритми генетичного програмування поширюють генетичні алгоритми для систем зі змінною довжиною генотипу).

6.2. Генетичні алгоритми

«Батьком» генетичних алгоритмів по праву вважається Д. Голланд, метод спочатку називався репродуктивним планом Голланда. У подальшому генетичні алгоритми розвивалися в роботах учнів Голланда: Д. Голдберга і К. Де Йонга – саме в них і закріпилася назва методу.

Генетичний алгоритм (англ. Genetic algorithm) – це евристичний алгоритм пошуку, який використовується для вирішення завдань оптимізації та моделювання, заснований на концепціях природного відбору і генетики.

Генетичні алгоритми оперують сукупністю індивідуумів (популяцією), які є рядками, що кодують одне з рішень задачі. Цим метод відрізняється від більшості інших алгоритмів оптимізації, які оперують лише з одним рішенням, покращуючи його.

Генетичні алгоритми застосовуються для вирішення наступних задач:

– оптимізація функцій;
– різноманітні задачі на графах (задачі комівояжера, розфарбовка, тощо);
– налаштування і навчання штучної нейронної мережі;
– завдання компонування;
– складання розкладів;
– ігрові стратегії;
– апроксимація функцій;
– штучне життя;
– біоінформатика.

Переваги генетичних алгоритмів:

1) універсальність;
2) висока видимість пошуку;
3) немає обмежень на цільову функцію;
4) будь-який спосіб завдання функції.

Недоліки генетичних алгоритмів:

1) відносно висока обчислювальна вартість;
2) квазіоптимальність.

Коли потрібно використати генетичний алгоритм: багато параметрів, погана цільова функція, комбінаторні завдання.

Коли не потрібно використати генетичний алгоритм: завдання добре вирішується традиційними методами, потрібно високу точність рішення.

6.3. Схема функціонування генетичних алгоритмів

З біології відомо, що будь-який організм може бути представлений своїм фенотипом, який фактично визначає, чим є об'єкт в реальному світі, і генотипом, який містить всю інформацію про об'єкт на рівні хромосомного набору. При цьому кожен ген, тобто елемент інформації генотипу, має своє відображення в фенотипі. Таким чином, для вирішення завдань необхідно представити кожну ознаку об'єкта в формі, придатній для використання в генетичному алгоритмі. Все подальше функціонування механізмів генетичного алгоритму проводиться на

рівні генотипу, дозволяючи обійтися без інформації про внутрішню структуру об'єкта, що й обумовлює його широке застосування в самих різних завданнях.

Найбільш часто у різновидах генетичного алгоритму для подання генотипу об'єкту застосовуються бітові рядки. При цьому кожному атрибуту об'єкта в фенотипі відповідає один ген в генотипі об'єкта. Набір генів або один ген представляє собою закодовану ознаку. Ген – це бітовий рядок, найчастіше фіксованої довжини. Для кодування гена в бінарній реалізації генетичного алгоритму часто використовують код Грея (див. приклад в таб. 12).

Таблиця 12

Приклад фенотипу

Ознака	Двійкове значення ознаки	Десяткове значення ознаки
Ознака1	0011	3
Ознака 2	1100	12
Ознака 3	1110	14
Ознака 4	0111	7
Ознака 5	1001	9

Після визначення фенотипу генетичний алгоритм функціонує за наступною схемою дій (рис. 59):

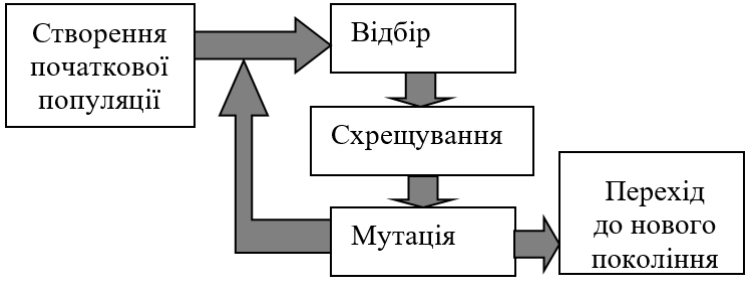


Рис. 59. Функціонування генетичного алгоритму

1 крок. Формування початкової популяції.

2 крок. Оцінка індивідуумів популяції (використовується фітнес-функція).

3 крок. Відбір (використовується один з методів відбору).

4 крок. Схрещування (використовується оператор кросовера).

5 крок. Мутація (використовується один або кілька операторів мутації).

6 крок. Формування нової популяції.

7 крок. Якщо популяція не зійшлися, то 2, інакше – зупинка. (припинення функціонування генетичного алгоритму).

Формування початкової популяції

Стандартний генетичний алгоритм починає свою роботу з формування початкової популяції – кінцевого набору допустимих рішень задачі. Ці рішення можуть бути обрані випадковим чином або отримані за допомогою простих наближених алгоритмів. Вибір початкової популяції не має значення для збіжності процесу, проте формування «хорошої» початкової популяції (наприклад, з множини локальних оптимумів) може помітно скоротити час досягнення глобального оптимуму. Якщо відсутні припущення про місцезнаходження глобального оптимуму, то індивідів з початкової популяції бажано розподілити рівномірно по всьому простору пошуку рішення.

Популяція ініціюється в початковий момент часу $t = 0$ і складається з k індивідуумів, кожна з яких представляє можливе рішення даної проблеми. Індивідуум – це одна або кілька хромосом (зазвичай одна). Хромосома складається з генів, тобто це бітовий рядок (хромосоми не обмежені бінарним представленням, є реалізації генетичного алгоритму, побудовані на векторах дійсних чисел). Гени розташовуються в різних позиціях хромосоми і приймають значення, звані алелями.

Оцінка індивідуумів *популяції*

Для вирішення задачі за допомогою генетичного алгоритму необхідно задати міру якості для кожного індивіда в просторі пошуку. Для цієї мети використовується функція пристосованості (fitness function). Функція пристосованості повинна приймати невід'ємні значення на обмеженій області визначення, при цьому абсолютно не потрібні

безперервність і диференційованість. Значення цієї функції визначає, наскільки добре підходить індивідуум для вирішення задачі.

У задачах максимізації цільова функція часто сама виступає в якості функції пристосованості, для задач мінімізації цільову функцію слід інвертувати. Якщо задача має обмеження, виконання яких неможливо контролювати алгоритмічно, то функція пристосованості, як правило, включає також штрафи за невиконання обмежень (вони зменшують її значення).

Відбір (селекція)

На кожному кроці еволюції за допомогою імовірнісного оператора селекції (відбору) вибираються два рішення-батька для їх подальшого схрещування. Серед операторів селекції найбільш поширеними є два імовірнісних оператора пропорційної і турнірній селекції. У деяких випадках також застосовується відбір урізанням, див. таб. 13.

Таблиця 13

Види відбору індивідуумів в генетичних алгоритмах

Вид відбору	Опис
Пропорційний	кожному індивідууму призначається ймовірність $P_s(i)$, що дорівнює відношенню його пристосованості до сумарної пристосованості популяції, здійснюється відбір (із заміщенням) усіх n (встановлюється заздалегідь) індивідуумів для подальшої генетичної обробки, відповідно до величини $P_s(i)$
Рулетка	вид пропорційного відбору, коли індивідууми відбираються за допомогою n «запусків» рулетки (колесо рулетки містить по одному сектору для кожного члена популяції, розмір i-ого сектора пропорційний відповідній величині $P_s(i)$)
Турнірний	з популяції, що містить m індивідуумів, вибирається випадковим чином t індивідуумів і вибирається найбільш пристосований (між обраними індивідуумами проводиться турнір), ця операція повторюється m раз
Відбір урізанням	з відсортованої в порядку убування ступеня пристосованості популяції з урахуванням порога пристосованості $T \in [0; 1]$ (нижче порога індивідууми в відборі не беруть участь) випадковим чином $m/2$ раз вибираються батьківські пари

Закінчення таблиці 13

Вид відбору	Опис
Рангові	для кожного індивідуума його ймовірність потрапити в проміжну популяцію пропорційна його порядковому номеру в відсортованої по зростанню пристосованості популяції
Елітні	додає до будь-якого іншого виду відбору принцип елітизму – збереження в новій популяції одного або декількох найбільш пристосованих індивідуумів

Схрещування

Як відомо, в теорії еволюції важливу роль відіграє те, яким чином ознаки батьків передаються нащадкам. У генетичних алгоритмах за передачу ознак батьків нащадкам відповідає оператор, який називається оператором схрещування (його також називають **кросовер або кросинговер**). Цей оператор визначає передачу ознак батьків нащадкам, до них застосовується імовірнісний оператор схрещування, який будує на їх основі нові (1 або 2) рішення-нащадки. Відібрані індивідууми піддаються кросоверу (іноді званого рекомбінацією) із заданою ймовірністю P_C. Якщо кожна пара батьків породжує двох нащадків, для відтворення популяції необхідно схрестити m/2 пари. Для кожної пари з імовірністю P_C застосовується кросовер. Отже, з імовірністю $1 - P_C$ кросовер не відбувається, і тоді незмінені індивідууми переходять на наступну стадію (мутації).

Існує велика кількість різновидів оператора схрещування, таб. 14. Найпростіший одноточковий кросовер працює наступним чином.

Спочатку випадковим чином вибирається одна з можливих точок розриву. (Точка розриву – ділянка між сусідніми бітами в рядку.) Обидві батьківські структури розриваються на два сегменти по цій точці. Потім відповідні сегменти різних батьків склеюються і виходять два генотипи нащадків:

Батько 1: 1 0 0 1 0 1 1 | 0 1 0 0 1
Батько 1: 0 1 0 0 0 1 1 | 0 0 1 1 1

Нащадок 1: 1 0 0 1 0 1 1 | 0 0 1 1 1
Нащадок 1: 0 1 0 0 0 1 1 | 0 1 0 0 1

В даний час дослідники ГА пропонують багато інших операторів схрещування. Двоточковий кросовер і рівномірний кросовер – цілком

гідні альтернативи одноточкового оператору. У двоточковому кросовері обираються дві точки розриву, і батьківські хромосоми обмінюються сегментом, який знаходиться між двома цими точками. У рівномірному кросовері кожен біт першого нащадку випадковим чином успадковується від одного з батьків; другому нащадку дістається біт другого з батьків.

Таблиця 14

Основні види операторів генетичних алгоритмів.

Оператор	Опис	Приклад
Оператори схрещування		
Одноточковий кросовер	Обирається одна точка розриву і батьківські хромосоми обмінюються однією з одержаних частин	**Батько 1:** 1001011|01001 **Батько 2:** 0100011|00111 **Нащадок 1:** 1001011|00111 **Нащадок 2:** 0100011|01001
Двоточковий кросовер	Обираються дві точки розриву і батьківські хромосоми обмінюються сегментом, який знаходиться між двома цими точками	**Батько 1:** 100|101101|001 **Батько 2:** 010|001100|111 **Нащадок 1:** 100|001100|001 **Нащадок 2:** 010|101101|111
Рівномірний кросовер	Кожен біт першого нащадка випадковим чином успадковується від одного з батьків, другому нащадку дістається біт другого з батьків	**Батько 1**: 100101101001 **Батько 2**: 010001100111 Вірогідність: 90% Випадкові числа (100): 2, 24, 8, 93, 55, 13, 67, 43, 99, 61, 5, 89 **Нащадок 1**: 100001100001 **Нащадок 2**: 010101101111
Оператори мутації		
Одноточкова мутація	Довільний біт хромосоми з певною ймовірністю змінюється на протилежний	**до:** 100101100111 **після:** 100101000111
Транслокація	Перенесення якої-небудь ділянки хромосоми в інший сегмент цієї ж хромосоми	**до:** 10011100111 **після:** 11100110111
Інверсія	Перестановка генів в зворотному порядку всередині довільно обраного ділянки хромосоми	**до:** 1001111100111 **після:** 1001001111111

Мутація

Наступний генетичний оператор призначений для того, щоб підтримувати різноманітність індивідуумів в популяції, – це оператор мутації. Після того як закінчиться стадія кросовера, нащадки можуть піддаватися випадковим модифікаціям. У найпростішому випадку в кожній хромосомі, яка піддається мутації, кожен біт з імовірністю P_m змінюється на протилежний (це так звана одноточкова мутація), таб. 14:

Індивідуум до мутації: 1 0 0 1 0 1 1 0 0 1 1 1

Індивідуум після мутації:: 1 0 0 1 0 1 0 0 0 1 1 1

Складнішим різновидом мутації є оператори інверсії і транслокації. Інверсія – це перестановка генів в зворотному порядку всередині довільно обраної ділянки хромосоми:

Індивідуум до інверсії: 1 0 0 1 1 1 1 0 0 1 1 1

Індивідуум після інверсії: 1 0 0 1 0 0 1 1 1 1 1 1

Транслокація – це перенесення якої-небудь ділянки хромосоми в інший сегмент цієї ж хромосоми:

Індивідуум до транслокації: 1 0 0 1 1 1 1 0 0 1 1 1

Індивідуум після транслокації: 1 1 1 0 0 0 1 1 0 1 1 1

Всі перераховані генетичні оператори (одноточковий і багатоточковий кросовер, одноточкова мутація, інверсія, транслокація) мають схожі біологічні аналоги.

Формування нового покоління

Після схрещування і мутації індивідуумів необхідно вирішити проблему: які з нових індивідуумів увійдуть в наступне покоління, а які – ні, і що робити з їхніми предками.

Є два найпоширеніші способи.

1. Нові індивідууми (нащадки) займають місця своїх батьків. Після цього настає наступний етап, в якому нащадки оцінюються, відбираються, дають потомство і поступаються місцем своїм «дітям».
2. Наступна популяція включає в себе як батьків, так і їх нащадків.

У другому випадку необхідно додатково визначити, які з індивідуумів батьків і нащадків потраплять в нове покоління. У найпростішому випадку в нього після кожного схрещування включаються два кращі індивідуума з четвірки батьків та їхніх нащадків. Більш ефективним є механізм витіснення, який реалізується таким чином, що прагне видаляти «схожі» хромосоми з популяції і залишати що відрізняються.

Критерії зупинки

Інший важливий момент функціонування алгоритму – визначення критеріїв зупинки. Взагалі кажучи, такий процес еволюції може тривати до нескінченності. Зазвичай в якості критеріїв зупинки застосовуються або обмеження на максимальне число епох функціонування алгоритму, або визначення його збіжності, зазвичай шляхом порівнювання пристосованості популяції на кількох епохах і зупинки при стабілізації цього параметра.

Сходженням називається такий стан популяції, коли всі рядки популяції майже однакові і знаходяться в області деякого екстремуму. У такій ситуації кросовер практично ніяк не змінює популяції. А ті що вийшли з цієї області за рахунок мутації індивідуума схильні вимирати, так як частіше мають меншу пристосованість, особливо якщо даний екстремум є глобальним максимумом. Таким чином, сходження популяції зазвичай означає, що знайдене краще або близьке до нього рішення.

Перед запуском генетичного алгоритму на виконання необхідно закодувати ознаки (параметри, за якими ведеться відбір), сформувати з них фенотип, визначити фітнес-функцію (критерій пристосованості). Існують різні види генетичного алгоритму, вони відрізняються використовуваними операторами, видами відбору, а також розрізняють послідовні і паралельні алгоритми.

6.4. Види генетичних алгоритмів

Існують різні моделі генетичного алгоритму (класичний, простий генетичний алгоритм, гібридний, CHC генетичний алгоритм та ін.). Вони розрізняються по стратегіям відбору і формування нового покоління індивідуумів, операторами генетичного алгоритму, кодуванням генів і т.п.

CHC-алгоритм

CHC (Cross generational elitist selection, Heterogenous recombination, Cataclysmic mutation) був запропонований Есхелманом і характеризується наступними параметрами:

1. Для нового покоління вибираються N кращих різних індивідуумів серед батьків і дітей. Дублювання рядків не допускається.

2. Для схрещування вибирається випадкова пара, але не допускається, щоб між батьками було мала хеммінгова відстань або мала відстань між крайніми розрізняючими бітами.

3. Для схрещування використовується різновид однорідного кросовера HUX (Half Uniform Crossover): дитині переходить рівно половина бітів кожного з батьків.

4. Розмір популяції невеликий, близько 50 індивідуумів. Цим виправдано використання однорідного кросовера.

CHC протиставляє агресивний відбір агресивному кросоверу, проте все одно малий розмір популяції швидко викликають стан, коли створюються тільки більш-менш однакові рядки. В такому випадку CHC застосовує cataclysmic mutation: всі рядки, крім самого пристосованого, піддаються сильній мутації (змінюється близько третини бітів). Таким чином, алгоритм перезапускається і далі продовжує роботу, застосовуючи тільки кросовер.

Genitor

Цей алгоритм був створений Д. Уітлі. Genitor-подібні алгоритми відрізняються від класичного ГА наступними трьома властивостями:

1. На кожному кроці тільки одна пара випадкових батьків створює тільки одну дитину.

2. Ця дитина замінює не батьків, а одну з найгірших індивідуумів популяції (в первісному Genitor – найгіршу).

3. Відбір індивідуума для заміни проводиться по його рейтингу, а не по пристосованості.

У Genitor пошук гіперплоскостей відбувається краще, а збіжність швидше, ніж у класичного генетичного алгоритму, запропонованого Холландом.

Гибридні алгоритми

Ідея гібридних алгоритмів (hybrid algorithms) полягає в поєднанні генетичного алгоритму з деяким іншим методом пошуку. У кожному поколінні кожен отриманий нащадок оптимізується цим методом, після чого виробляються звичайні для генетичного алгоритму дії.

Такий вид розвитку називається ламарковою еволюцією, при якій індивідуум здатен навчатися, а потім отримані навички записувати в свій генотип, щоб потім передати їх нащадкам. І хоча такий метод погіршує здатність алгоритму шукати розв'язок за допомогою відбору

гіперплоскостей, однак на практиці гібридні алгоритми виявляються дуже вдалими. Це пов'язано з тим, що зазвичай велика ймовірність того, що один з індивідуумів потрапить в область глобального максимуму і після оптимізації виявиться розв'язком задачі.

Паралельні генетичні алгоритми

Генетичні алгоритми можна організувати як кілька паралельно виконуваних процесів, це збільшить їх продуктивність.

Розглянемо перехід від класичного генетичного алгоритму до паралельного. Для цього будемо використовувати турнірний відбір. Створемо *N*/2 процесу (тут і далі процес мається на увазі як деяка машина, процесор, який може працювати незалежно). Кожен з них буде вибирати випадково з популяції 4 індивідуума, проводити 2 турніра і схрещувати переможців. Отримані діти будуть записуватися в нове покоління. Таким чином, за один цикл роботи одного процесу буде змінюватися ціле покоління.

Острівна модель

Острівна модель (island model, рис. 60) – це теж модель паралельного генетичного алгоритму. Вона полягає в наступному: нехай у нас є 16 процесів і 1600 індивідуумів. Розіб'ємо їх на 16 підпопуляцій по 100 індивідуумів. Кожна з них буде розвиватися окремо за допомогою якогось генетичного алгоритму. Таким чином, можна сказати, що ми розселили індивідуумів по 16-ти ізольованим островам.

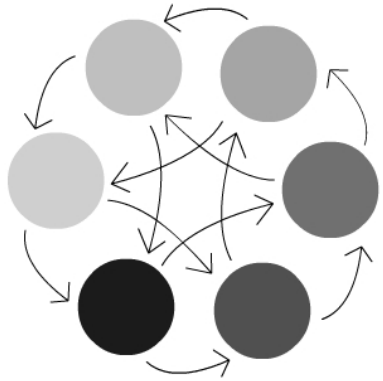


Рис. 60. Острівна модель генетичного алгоритму

Зрідка (наприклад, кожні 5 поколінь) процеси (або острова) обмінюватимуться декількома хорошими індивідуумами. Цей процес називається міграцією. Міграція дозволяє островам обмінюватися генетичним матеріалом.

Так як населеність островів зазвичай буває невелика, підпопуляції будуть схильні до передчасної збіжності. Тому важливо правильно встановити частоту міграції. Надто часта міграція (або міграція занадто великого числа індивідуумів) призведе до змішання всіх підпопуляцій, і тоді острівна модель буде несильно відрізнятися від звичайного генетичного алгоритму. Якщо ж міграція буде занадто рідкою, то вона не зможе запобігти передчасному сходженню підпопуляцій.

Генетичні алгоритми стохастичні, тому при різних запусках популяція може сходитися до різних рішень (хоча всі вони в деякій мірі «хороші»). Острівна модель дозволяє запустити алгоритм відразу кілька раз і намагатися поєднувати «досягнення» різних островів для отримання в одній з підпопуляції найкращого рішення.

Комірчастий генетичний алгоритм

Комірчастий генетичний алгоритм (Cellular Genetic Algorithms) – модель паралельних генетичних алгоритмів. Нехай дано 2500 процесів, розташованих на сітці розміром 50×50 комірок, замкнутої, як показано на рис. 61 (ліва сторона замикається з правою, верхня – з нижньою, виходить тор).

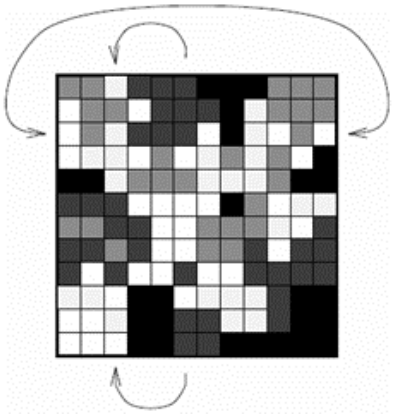


Рис. 61. Комірчастий генетичний алгоритм

Кожен процес може взаємодіяти тільки з чотирма своїми сусідами (зверху, знизу, зліва, справа). У кожній комірці знаходиться рівно один індивідуум. Кожен процес обиратиме найкращу комірку серед своїх сусідів, схрещувати з нею індивідуума з своєї комірки і одної отриманої дитини поміщати в свою комірку замість батька.

У міру роботи такого алгоритму виникають ефекти, схожі на острівну модель. Спочатку всі індивідууми мають випадкову пристосованість. Через кілька поколінь утворюється невелика область схожих індивідуумів з близькою пристосованістю. По мірі роботи алгоритму ці області ростуть і конкурують між собою.

6.5. Використання генетичного алгоритму (приклад розв'язку задачі)

Розглянемо на прикладі процес розв'язку задачі з застосування генетичного алгоритму.

Умова. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, поділена на середнє значення суми біт індивідуумів популяції; метод відбору – рулетка з принципом елітизму; оператор схрещування – двоточковий кросовер; оператор мутації – одиночна мутація.

Опис процесу розв'язку

Для використання генетичного алгоритму необхідно:

1) Визначити набір ознак, що характеризують рішення задачі оптимізації або моделювання. Визначити фенотип, закодувати ознаки (можна використовувати код Грея).

2) Використовувати послідовність кроків генетичного алгоритму з відповідними операторами.

Розв'язок

1) Фенотип (задаємо десяткові значення випадковим чином):

Ознака	Двійкове значення ознаки	Десяткове значення ознаки	Код Грея
Ознака 1	0011	3	0010
Ознака 2	1100	12	1010
Ознака 3	1110	14	1001

Ознака	Двійкове значення ознаки	Десяткове значення ознаки	Код Грея
Ознака 4	0111	7	0100
Ознака 5	1001	9	1101

1 крок. Формування початкової популяції. Визначено п'ять ознак, нехай індивідуум містить будь-які 2 з них (два перших – значення першого критерію, три останніх другого), випадковим чином згенеруємо 10 індивідуумів, кожний індивідуум довжиною 8 біт:

Індивідуум 1: 00111110
Індивідуум 2: 11001110
Індивідуум 3: 00111001
Індивідуум 4 : 11001001
Індивідуум 5: 00110111
Індивідуум 6: 00111110
Індивідуум 7: 11000111
Індивідуум 8: 00110111
Індивідуум 9: 10101010
Індивідуум 10: 01010101

2 крок. Оцінка індивідуумів популяції (фітнес-функція, яка використовується, дорівнює сумі біт індивідуума).

Індивідуум	Сума біт в індивідуумі	Пристосованість індивідуума
1	5	1,389
2	5	1,389
3	4	1,112
4	4	1,112
5	5	1,389
6	5	1,389
7	5	1,389
8	5	1,389
9	4	1,112
10	4	1,112

Середнє значення суми біт в популяції = 3,6.

3 крок. Відбір (використовується метод відбору – рулетка з принципом елітизму). Будуємо рулетку (сектора пропорційні пристосованості, рис. 62) і запускаємо її 8 разів (обираємо 4 пари, рис. 63):

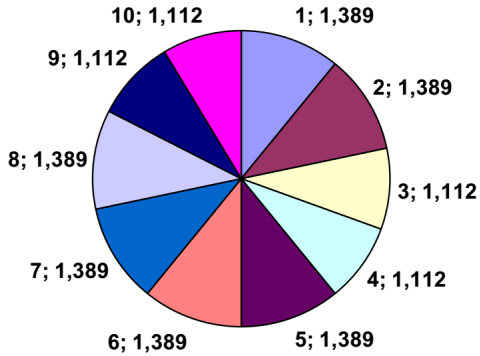


Рис. 62. Рулетка для завдання генетичного алгоритму

Таким чином, утворилися наступні пари: 1 і 5; 7 і 5; 10 і 2; 8 і 6.

4 крок. Схрещування (використовується оператор – двоточковий кросовер). Обираємо дві точки розриву (випадковим чином, але числа повинні відрізнятися хоча б на 2 і не бути рівними 1 або довжині індивідуума): 2 і 5 та застосовуємо оператор до обраних пар індивідуумів.

Індивідуум 1:	00|111|110	→	*Індивідуум 2.1:*	00|110|110
Індивідуум 5:	00|110|111		*Індивідуум 2.2:*	00|111|111
Індивідуум 7:	11|000|111	→	*Індивідуум 2.3:*	11|110|111
Індивідуум 5:	00|110|111		*Індивідуум 2.4:*	00|000|111
Індивідуум 10:	01|010|101	→	*Індивідуум 2.5:*	01|001|101
Індивідуум а 2:	11|001|110		*Індивідуум 2.6:*	11|010|110
Індивідуум 6:	00|111|110	→	*Індивідуум 2.7:*	00|110|110
Індивідуум 8:	00|110|111		*Індивідуум 2.8:*	00|111|111

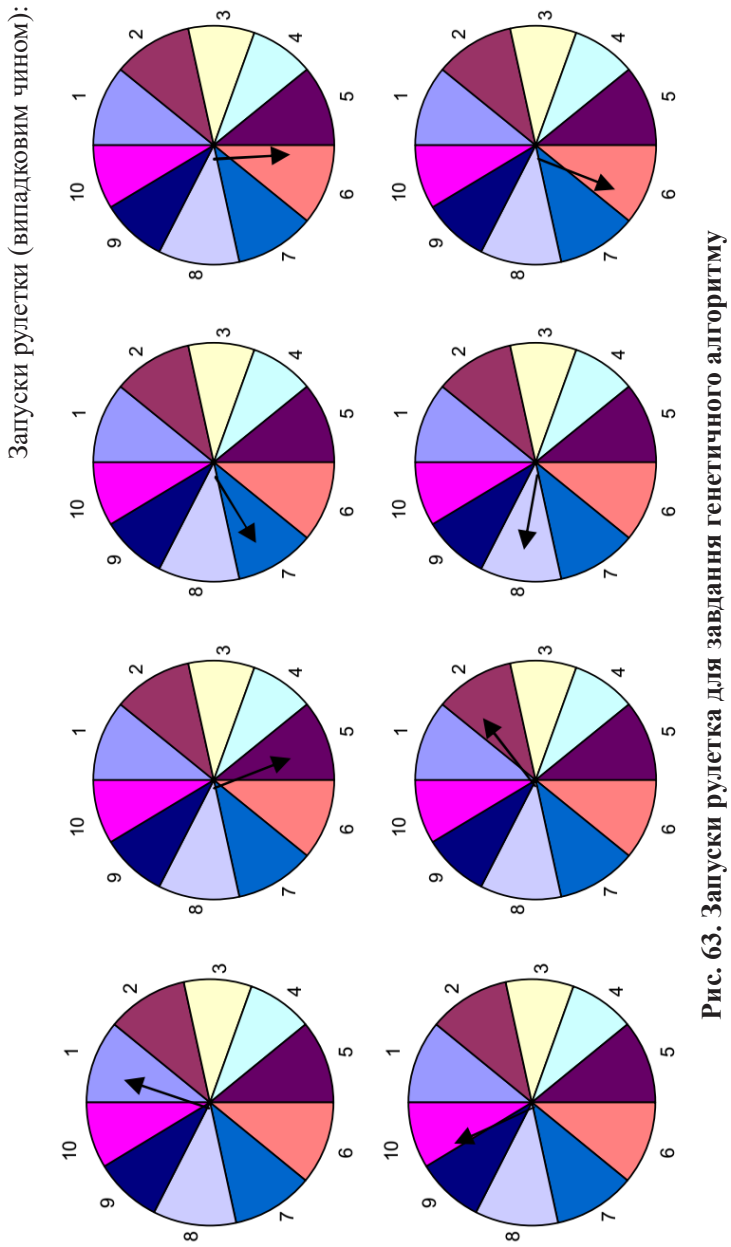


Рис. 63. Запуски рулетка для завдання генетичного алгоритму

5 крок. Мутація (використовується оператор – одноточкова мутація). Визначимо ймовірність мутації 30% і біт – третій, що піддається мутації.

Індивідуум	Випадкове число	Індивідуум	Мутований індивідуум
1	5	00111110	00011110
2	33	11001110	
3	67	00111001	
4	78	11001001	
5	90	00110111	
6	12	00111110	00011110
7	45	11000111	
8	53	00110111	
9	74	10101010	
10	29	01010101	01110101

6 крок. Формування нової популяції

Індивідуум 2.1: 00110110 Індивідуум 2.5: 01001101 Індивідуум 2.9: 00011110
Індивідуум 2.2: 00111111 Індивідуум 2.6: 11010110 Індивідуум 2.10: 00011110
Індивідуум 2.3: 11110111 Індивідуум 2.7: 00110110 Індивідуум 2.11: 01110101
Індивідуум 2.4: 00000111 Індивідуум 2.8: 00111111 Індивідуум 2.12: 00111110

1–8 – спадкоємці, 9–11 – матовані індивідууми, 12 – зберігаємо однин індивідуум з максимальною пристосованістю – принцип елітизму.

7 крок. Популяція досить різноманітна – немає ознак збіжності. Так як розглядається лише одна епоха генетичного алгоритму – вихід з алгоритму.

6.6. Завдання для самоконтролю

1. Хто вважається «батьком» генетичних алгоритмів?
А) Д. Голдберг
Б) Д. Холланд
В) К. Де Йонг
Г) немає правильної відповіді

2. Які методи відносяться до напряму «Еволюційне моделювання»?
А) метод групового урахування аргументів
Б) нейронні мережі
В) генетичні алгоритми
Г) еволюційне програмування
Д) евристичне програмування

3. Які поняття відносяться до генетичних алгоритмів?
А) індивідуум
Б) фенотип
В) ген
Г) ДНК
Д) нейрон
Е) функція активації

4. Які види відбору в генетичних алгоритмах існують?
А) дискретний відбір
Б) рангові відбір
В) поетапний відбір
Г) дуельний відбір
Д) турнірний відбір
Е) рулетка

5. Які бувають оператори генетичного алгоритму?
А) кроссинговер
Б) схрещування
В) транслітерація
Г) транслокация
Д) мутація
Е) конверсія

6. В яких видах генетичного алгоритму присутня паралельна обробка?
А) genitor
Б) CHC
В) гібридні алгоритми
Г) острівна модель
Д) немає правильної відповіді

7. З якого числа індивідуумів можна вибирати пару (другого з батьків) для індивідуума в острівній моделі?
А) m, де m – число індивідуумів в популяції
Б) m-1, де m – число індивідуумів в популяції

В) 4
Г) 8
Д) t, вибирається випадковим чином, найчастіше $t = 2$
Е) немає правильної відповіді

8. Який оператор застосований до індивідуума (0001000 → 0000000)?
А) інверсії
Б) кросовер
В) схрещування
Г) немає правильної відповіді

6.7. Завдання для самостійної роботи

1. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, поділена на максимум суми всіх біт серед індивідуумів популяції; метод відбору – рулетка; оператор схрещування – одноточковий кросовер; оператор мутації – одиночна мутація.

2. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, поділена на мінімум суми всіх біт серед індивідуумів популяції; метод відбору – турнірний відбір; оператор схрещування – двоточковий кросовер; оператор мутації – транслокація.

3. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – одиниця, поділена на мінімум суми всіх біт серед індивідуумів популяції; метод відбору – ранговий відбір; оператор схрещування – рівномірний кросовер; оператор мутації – інверсія.

4. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів).

Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, помножена на мінімум суми всіх біт серед індивідуумів популяції; метод відбору – відбір урізанням; оператор схрещування – рівномірний кросовер; оператор мутації – одноточкова мутація.

5. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – одиниця, поділена на максимум суми всіх біт індивідуума в популяції; метод відбору – пропорційний відбір; оператор схрещування – одноточковий кросовер; оператор мутації – інверсія.

6. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, поділена на кількість біт індивідуума; метод відбору – рулетка з використанням принципу елітизму; оператор схрещування – рівномірний кросовер; оператор мутації – інверсія.

7. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість біт індивідуума; метод відбору – пропорційний з використанням принципу елітизму; оператор схрещування – одноточковий кросовер; оператор мутації – одноточкова мутація.

8. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість індивідуумів в популяції; метод відбору – рангові з використанням принципу елітизму; оператор схрещування – одноточковий кросовер; оператор мутації – одноточкова мутація.

9. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів).

Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість індивідуумів в популяції; метод відбору – турнірний з використанням принципу елітизму; оператор схрещування – рівномірний кросовер; оператор мутації – транслокація.

10. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість індивідуумів в популяції; метод відбору – відбір урізанням з використанням принципу елітизму; оператор схрещування – двоточковий кросовер; оператор мутації – транслокація.

11. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, поділена на максимум суми всіх біт індивідуума в популяції; метод відбору – рулетка; оператор схрещування – двоточковий кросовер; оператор мутації – одиночна мутація.

12. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на максимум суми всіх біт індивідуума в популяції; метод відбору – турнірний відбір; оператор схрещування – рівномірний кросовер; оператор мутації – інверсія.

13. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – одиниця, поділена на мінімум суми всіх біт індивідуума в популяції; метод відбору – ранговий відбір; оператор схрещування – одноточковий кросовер; оператор мутації – інверсія.

14. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати

випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт, помножена на мінімум суми всіх біт індивідуума в популяції; метод відбору – відбір урізанням; оператор схрещування – рівномірний кросовер; оператор мутації – транслокація.

15. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – одиниця, поділена на максимум суми всіх біт серед індивідуумів популяції; метод відбору – пропорційний відбір; оператор схрещування – одноточковий кросовер; оператор мутації – транслокація.

16. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість біт індивідуума; метод відбору – рулетка з використанням принципу елітизму; оператор схрещування – одноточковий кросовер; оператор мутації – транслокація.

17. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість біт індивідуума; метод відбору – пропорційний з використанням принципу елітизму; оператор схрещування – двоточковий кросовер; оператор мутації – інверсія.

18. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість індивідуумів в популяції; метод відбору – ранговий з використанням принципу елітизму; оператор схрещування – рівномірний кросовер; оператор мутації – транслокація.

19. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість індивідуумів в популяції; метод відбору – турнірний з використанням принципу елітизму; оператор схрещування – рівномірний кросовер; оператор мутації – одноточкова мутація.

20. Описати функціонування однієї епохи генетичного алгоритму на прикладі довільної задачі (не менше п'яти ознак закодувати випадковим чином, початкова популяція містить не менше 10 індивідуумів). Використовувати такі параметри генетичного алгоритму: фітнес-функція – сума всіх біт індивідуума, поділена на кількість індивідуумів в популяції; метод відбору – відбір урізанням з використанням принципу елітизму; оператор схрещування – двоточковий кросовер; оператор мутації – одноточкова мутація.

6.8. Лабораторна робота 13. Генетичні алгоритми

Мета: навчитися реалізовувати генетичні алгоритми в додатку Genetic Algorithm and Direct Search Toolbox

Завдання: Використовуючи вбудовані функції пакета Genetic Algorithm and Direct Search Toolbox середовища MATLAB, навчитися реалізовувати генетичні алгоритми у завданнях оптимізації.

Теоретичні відомості

Генетичний алгоритм (англ. genetic algorithm) – це евристичний алгоритм пошуку, що використовується для розвязання завдань оптимізації й моделювання шляхом випадкового підбору, комбінування й варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію, є різновидом еволюційних обчислень.

Відмінною рисою генетичного алгоритму є акцент на використанні оператора «схрещування», що виконує операцію рекомбінації рішень-кандидатів, роль якої аналогічна ролі схрещування в живій природі.

Функція пристосованості – це функція, що підлягає оптимізації. У випадку стандартних оптимізаційних алгоритмів вона відома як цільова функція. Функцію пристосованості можна записати в М-файл і передати у вигляді якогось аргументу в основний генетичний алгоритм.

Індивідуум – деяка точка, у якій можливий розрахунок функції пристосованості. Значення функції пристосованості індивідуалізованого об'єкта саме і є її кількісний показник. Наприклад, якщо функція пристосованості має такий вигляд:

$$f(x_1, x_2, x_3) = (2\,x_1 + 1)^2 + (3\,x_2 + 4)^2 + (x_3 - 2)^2,$$

то вектор (2, 3, 1), розмірність якого дорівнює числу змінних даної задачі, і є індивідуум. Кількісний показник індивідуума як об'єкта (2, 3, 1) буде $f(2, -3, 1) = 51$. Об'єкт індивідуума іноді називається геном, а компоненти вектора називаються генами.

Сімейство – це масив індивідуалізованих об'єктів. Наприклад, якщо розмір сімейства дорівнює 100 і число змінних у функції пристосованості дорівнює 3, тоді дане сімейство являє собою матрицю розміром 100x3. Той самий об'єкт індивідуума може з'являтися в даному сімействі більше одного разу.

Для того щоб одержати нове покоління, в генетичному алгоритмі на кожній ітерації проводиться деяка серія розрахунків на основі поточного сімейства. Кожне успішне сімейство називається новим поколінням. Для того щоб одержати наступне сімейство, у генетичному алгоритмі виконується відбір певних індивідуалізованих об'єктів у поточному сімействі, що називаються батьківськими і які далі використовуються для утворення об'єктів індивідуумів для наступного сімейства – дочірнього. Як правило, вибираються ті батьки, які мають більше значення функції пристосованості. Genetic Algorithm and Direct Search Toolbox Genetic Algorithm and Direct Search Toolbox призначений для розширення функціональних можливостей пакета MATLAB і, зокрема, Optimization Toolbox новими видами алгоритмів оптимізації. Подібні методи й алгоритми найчастіше використовуються у випадку, коли цільова функція є переривчастою, істотно нелінійною, стохастичною і не має похідних або ці похідні є недостатньо визначеними. Genetic Algorithm and Direct Search Toolbox допоможе

розв'язати задачі, які або неможливо розв'язати звичайними методами, або виникають нестійкі рішення при застосуванні стандартних математичних методів.

Genetic Algorithm and Direct Search Toolbox містить у собі програми для розв'язання задач оптимізації на основі використання наступних методів:

– генетичний алгоритм;
– метод прямого пошуку.

Завданням даного тулбоксу є пошук мінімуму функції пристосованості. Усі функції є М-файлами пакета MATLAB, складені з операторів MATLAB і являють собою реалізацію спеціалізованих алгоритмів оптимізації. Є можливість переглядати ці функції шляхом виконання звичайного оператора: type function_name.

Генетичний алгоритм – це метод розв'язання задач оптимізації на основі природного добору, аналогічно тому, як це відбувається в процесі біологічної еволюції. У генетичному алгоритмі відбувається багаторазова модифікація сімейства окремих розв'язків. На кожному кроці проводиться відбір вибраних навмання суб'єктів з отриманого поточного розв'язання, що називається батьківським і яке використовується для генерації наступного дочірнього покоління. За допомогою послідовного відбору поколінь відбувається «еволюція» у напрямку до оптимального розв'язання. Генетичний алгоритм можна застосовувати до різноманітних задач оптимізації, які недостатньо чітко вписуються в стандартні оптимізаційні алгоритми.

На кожному кроці для генерації наступного покоління в генетичному алгоритмі використовуються наступні три основні правила:

– правило відбору – відбирає об'єкти, що називаються батьківськими та які становлять основу наступного покоління з поточного розв'язання;
– правило схрещування, за яким відбувається вибір із двох батьківських об'єктів дочірнього для наступного покоління;
– правило мутації, за яким на основі ймовірнісних алгоритмів з батьківських об'єктів формуються дочірні.

Генетичний алгоритм відрізняється від стандартних методів оптимізації наступними особливостями (табл. 15):

Таблиця 15

Порівняння стандартного й генетичного алгоритмів

Стандартний алгоритм	Генетичний алгоритм
На кожній ітерації генерується одна єдина точка. Отримана послідовність точок сходиться до оптимального розв'язку	На кожній ітерації генерується сімейство точок. Отримане сімейство сходиться до оптимального розв'язку.
Вибір наступної точки здійснюється на основі детермінованих розрахунків.	Вибір наступного сімейства точок здійснюється на основі стохастичних розрахунків

Структура алгоритму

Генетичний алгоритм працює відповідно до наступної схеми:

1) для організації початку рахунку створюється довільне вихідне сімейство;

2) алгоритм робить деяку послідовність нових сімейств або поколінь. На кожному окремому кроці алгоритм використовує певні індивідууми з поточного покоління, для того щоб створити наступне покоління. При формуванні нового покоління в алгоритмі проводяться наступні дії:

– відмічається кожний член поточного сімейства за допомогою обчислення відповідного значення пристосованості;

– проводиться масштабування отриманого ряду значень функції пристосованості, що дозволяє побудувати діапазон значень більш зручний для наступного використання;

– вибираються батьківські значення на основі значень їхньої пристосованості;

– частина індивідуумів з батьківського покоління має більші значення функції пристосованості, які далі вибираються як елітні значення. Ці елітні значення передаються вже в наступне покоління;

– дочірні значення утворюються або шляхом якихось випадкових змін окремого одного батька – мутація, або шляхом комбінації векторних компонентів якоїсь пари батьків – кросовер;

– заміна поточного сімейства на дочірнє з метою формування наступного покоління;

3) зупинка алгоритму здійснюється тоді, коли виконується будь-який критерій зупинки.

Графічний інструментарій генетичного алгоритму

Інструментарій генетичного алгоритму – це фактично графічний інтерфейс користувача, за допомогою якого користувач має можливість роботи з генетичним алгоритмом без звернення до нього з командного рядка. Для звернення до Інструментарію генетичного алгоритму варто виконати команду *gatool*, при виконанні якої відкривається наступний інструментарій (рис. 64):

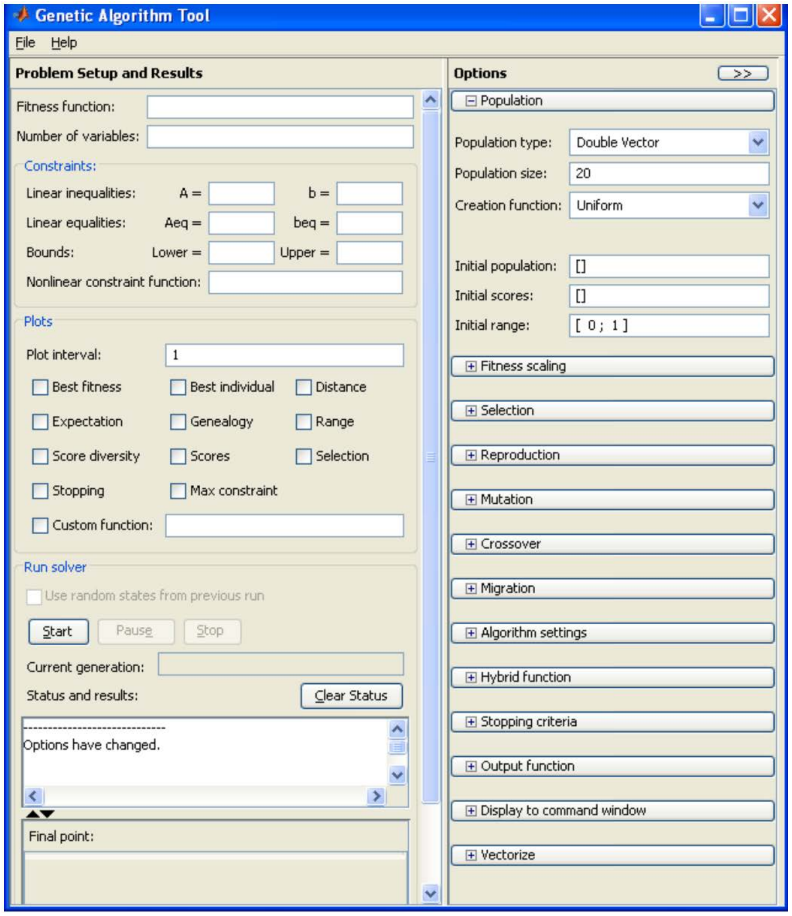


Рис. 64. Графічний інструментарій генетичного алгоритму

Для роботи з Інструментарієм генетичного алгоритму варто ввести наступну інформацію:

– *Fitness function* – підлягаюча мінімізації цільова функція. У форму @fitnessfun вводиться функція пристосованості, де fitnessfun.m – це М-файл для розрахунку функції пристосованості. Знак «@» створює описувач функції для fitnessfun.

– *Number of variables* – розмір вхідного вектора для функції пристосованості.

Для виконання генетичного алгоритму варто клікнути мишкою на кнопку «*Start*». Далі в інструментарії в панелі «*Status and Results*» здійснюється відображення результатів оптимізації. За допомогою панелі «*Options*» можливо налаштувати опції генетичного алгоритму. Для огляду опцій для заданого режиму в даній панелі варто клікнути на знак «+».

Основними параметрами в *GATool* є:

– популяція (вкладка «*Population*»);
– масштабування (вкладка «*Fitness Scaling*»);
– оператор відбору (вкладка «*Selection*»);
– оператор репродукції (вкладка «*Reproduction*»);
– оператор мутації (вкладка «*Mutation*»);
– оператор схрещування (вкладка «*Crossover*»);
– перенесення особин між популяціями (вкладка «*Migration*»);
– спеціальні параметри алгоритму (вкладка «*Algorithm settings*»);
– задання гібридної функції (вкладка «*Hybrid function*»);
– задання критерію зупинки алгоритму (вкладка «*Stopping criteria*»);
– відображення різної додаткової інформації у ході роботи генетичного алгоритму (вкладка «*Plot Functions*»);
– відображення результатів роботи алгоритму у вигляді нової функції (вкладка «*Output function*»);
– задання набору інформації для виведення в командне вікно (вкладка «*Display to command window*»);
– спосіб обчислення значень оптимізованої й обмежувальної функцій (вкладка «*User function evaluation*»).

У вкладці налаштування популяцій («*Population*») користувач має можливість вибрати тип математичних об'єктів, до якого будуть належать індивідууми всіх популяцій (подвійний вектор, бітовий рядок або користувацький тип). При цьому варто враховувати, що використання бітового рядка й користувацьких типів накладають обмеження на перелік припустимих операторів створення, мутації й схрещування особин. Також вкладка популяції дозволяє налаштовувати розмір популяції (*Population size* – зі скількох особин буде складатися кожне покоління) і яким чином буде створюватися початкове покоління (*Creation function*: *Uniform* – якщо відсутні обмеження, в іншому випадку – Feasible population). Крім того, є можливість задати вручну початкове покоління (використовуючи пункт *Initial population*) або його частину, початковий рейтинг індивідуумів (пункт *Initial scores*), а також задати обмежувальний числовий діапазон, якому повинні належати особини початкової популяції (*Initial range*).

У вкладці масштабування («*Fitness Scaling*») зазначається функція масштабування, яка конвертує значення, що досягаються оптимізуючою функцією в значення, що лежать у межах, припустимих для оператора відбору. При виборі як функції масштабування параметра *Rank* масштабування буде приводитися до рейтингу, тобто індивідуумам присвоюється рейтинговий номер (для кращого індивідуума – одиниця, для наступного – двійка і так далі). Пропорційне масштабування (*Proportional*) задає ймовірності пропорційно заданому числовому ряду для індивідуумів. При виборі опції *Top* найбільше рейтингове значення присвоюється відразу декільком найбільш значущим індивідуумам (їхнє число вказується у вигляді параметра). При виборі масштабування типу *Shift linear* є можливість указати максимальну ймовірність найкращого індивідуума.

Вкладка «*Selection*» дозволяє вибрати оператор відбору батьківських індивідуумів на основі даних з функції масштабування. Як доступні для вибору варіантів оператора відбору пропонуються наступні:

– *Tournament* – випадково вибирається зазначене число індивідуумів, серед них на конкурсній основі вибираються кращі;

– *Roulette* – імітується рулетка, у якій розмір кожного сегмента встановлюється відповідно до його ймовірності;

– *Uniform* – батьки вибираються випадковим чином відповідно до заданого розподілу й з урахуванням кількості батьківських індивідуумів і їхніх імовірностей;

– *Stochastic uniform* – будується лінія, у якій кожному батькові ставиться у відповідність її частина певного розміру (залежно від

імовірності батька), потім алгоритм пробігає по лінії кроками однакової довжини й вибирає батьків залежно від того, на яку частину лінії потрапив крок. Вкладка репродукції («*Reproduction*») уточнює, яким чином відбувається створення нових індивідуумів. Пункт *Elite count* дозволяє вказати число індивідуумів, які гарантовано перейдуть до наступного покоління. Пункт *Crossover* fraction указує частину індивідуума, які створюються шляхом схрещування. Інша частина створюється шляхом мутації. У вкладці оператора мутації («*Mutation*») вибирається тип оператора мутації. Доступні наступні варіанти:

– *Gaussian* – додає невелике випадкове число (відповідно до розподілу Гаусса) до всіх компонентів кожного вектора-індивідуума;

– *Uniform* – вибираються випадковим чином компоненти векторів і замість них записуються випадкові числа із допустимого діапазону;

– *Adaptive feasible* – генерує набір напрямків залежно від останніх найбільш удалих і невдалих поколінь і з урахуванням обмежень, що накладаються, просувається вздовж усіх напрямків на різну довжину;

– *Custom* – дозволяє задати власну функцію.

Вкладка схрещування («*Crossover*») дозволяє вибрати тип оператора схрещування: одноточкове, двоточкове, евристичне, арифметичне або розсіяне.

У вкладці «*Migration*» можна задавати правила, згідно з якими індивідууми будуть переміщатися між підпопуляціями в межах однієї популяції. Підпопуляції створюються, якщо як розмір популяції зазначений вектор, а не натуральне значення. У даній вкладці можна вказати напрямок міграції (*forward* – у наступну підпопуляцію, *both* – у попередню й наступну), частину мігруючих індивідуумів (*Fraction*) і частоту міграції (скільки поколінь проходить між міграціями).

Вкладка спеціальних опцій алгоритму («*Algorithm settings*») дозволяє задавати параметри розв'язку системи нелінійних обмежень, що накладаються на алгоритм.

Вкладка «*Hybrid function*» дозволяє задати ще одну функцію мінімізації, що буде використовуватися після закінчення роботи алгоритму. Як можливі гібридні функції доступні наступні вбудовані в саме середовище MATLAB функції:

– *none* (не використовувати гібридну функцію);

– *fminsearch* (пошук мінімального зі значень);

– *patternsearch* (пошук за зразком);

– *fminunc* (для необмеженого алгоритму);

– *fmincon* (для алгоритму із заданими обмеженнями).

У вкладці критерію зупинки («*Stopping criteria*») зазначаються ситуації, при яких алгоритм робить зупинку. При цьому задаються наступні параметри:

– *Generations* – максимальне число поколінь, після перевищення якого відбудеться зупинка;

– *Time limit* – ліміт часу на роботу алгоритму;

– *Fitness limit* – якщо оптимізоване значення менше або дорівнює даному ліміту, то алгоритм зупиниться;

– *Stall generations* – алгоритм зупиняється у випадку, якщо немає поліпшення для цільової функції в послідовності наступних один за одним поколінь довжиною Stall generations;

– *Stall time limit* – алгоритм зупиняється у випадку, якщо немає поліпшення для цільової функції протягом інтервалу часу в секундах рівного Stall time limit;

– *Function tolerance* і *Nonlinear constraint tolerance* – мінімальні значення змін оптимізованої та обмежувальної функцій відповідно, при яких алгоритм продовжить роботу.

Вкладка «*Plot Functions*» дозволяє вибирати різну інформацію, що виводиться у ході роботи алгоритму й показує як коректність його роботи, так і конкретні результати, що досягаються алгоритмом. Найбільш важливими й використовуваними для відображення параметрами є:

– *Plot interval* – число поколінь, по закінченні якого відбувається чергове відновлення графіків;

– *Best fitness* – відображення найкращого значення оптимізованої функції для кожного покоління;

– *Best individual* – відображення найкращого представника покоління при найкращому результаті в кожному з поколінь;

– *Distance* – відображення інтервалу між значеннями індивідуумів у поколінні;

– *Expectation* – виводить ряд імовірностей і відповідні їм індивідууми поколінь;

– *Genealogy* – відображення генеалогічного дерева особин;

– *Range* – відображення найменшого, найбільшого й середнього значень оптимізованої функції для кожного покоління;

– *Score diversity* – вивести гістограму рейтингу в кожному поколінні;

– *Scores* – відображення рейтингу кожного індивідууму в поколінні;

– *Selection* – відображення гістограми батьків;

– *Stopping* – відображення інформації про стан усіх параметрів, що впливають на критерії зупинки;

– *Custom* – відображення на графіку деякої зазначеної користувачем функції.

Вкладка виведення результатів у вигляді нової функції («*Output function*») дозволяє включити виведення історії роботи алгоритму в окремому вікні із заданим інтервалом поколінь (прапор *History to new window* і поле *Interval* відповідно), а також дозволяє задати й вивести довільну вихідну функцію, що задається в поле Custom function.

Вкладка «*User function evaluation*» описує, у якому порядку відбувається обчислення значень оптимізованої та обмежувальної функцій (окремо, паралельно в одному виклику або одночасно).

Вкладка «*Display to command window*» дозволяє задавати інформацію, що відображається в основному командному вікні MATLAB при роботі алгоритму. Можливі наступні значення: *Off* – немає виведення в командне вікно, *Iterative* – виведення інформації про кожну ітерацію працюючого алгоритму, *Diagnose* – виведення інформації про кожну ітерацію й додаткові відомості про можливі помилки й змінені ключові параметри алгоритму, *Final* – виводиться тільки причина зупинки й кінцеве значення.

Практична частина

1. Необхідно знайти мінімум наступної функції:

$$y(x_1, x_2) = x_1^2 - 2x_1 x_2 + 6x_1 + x_2^2 - 6x_2 .$$

2. Для створення m-файла, що необхідний для розрахунку даної функції, слід створити порожній m-файл і ввести наступний код:

```
function z = my_fun(x)
z = x(1)^2 – 2*x(1)*x(2) + 6*x(1) + x(2)^2 – 6*x(2);
```


3. Зберегти m-файл у поточній робочій директорії MATLAB з ім'ям my_fun. m.

4. Для перевірки, що М-файл повертає точне рішення, варто виконати:

```
my_fun([2 3])
ans =
      -5
```

5. Відкрити інструментарій генетичного алгоритму. У поле Fitness function ввести ім'я цільової функції @my_fun, указати розмірність вхідного вектора для функції пристосованості.

6. Встановити значення параметрів генетичного алгоритму: кількість індивідуумів у популяції = 10, кількість поколінь = 100 (у вкладці критерію зупинки алгоритму), початковий відрізок = [–1; 1].

7. У розділі *Plots* установити прапорці для графіків *Best fitness*, *Best individual, Distance*. Запустити генетичний алгоритм, проаналізувати отримані графіки.

8. У результаті завершення процесу у вікні *Final point* з'явиться значення змінної x, що відповідає мінімуму функції, а у вікні *Status* and *result* можна побачити знайдене мінімальне значення цільової функції.

Завдання до лабораторної роботи 13

Знайти максимум функції:

$$y = n\sqrt[3]{(x+4)^2} - 2x - 8n$$

де n – номер варіанту.

Відобразити отримані графіки.

Перевірити правильність розв'язання алгоритму, побудувавши графік заданої функції та порівняти точки мінімуму (максимуму).

Контрольні запитання

1. Дайте визначення генетичного алгоритму.
2. Що таке функція пристосовуваності?
3. Назвіть критерії зупинки роботи генетичного алгоритму.
4. Яким чином створюється початкова популяція?
5. Чому значення найкращого представника покоління відрізняється від попереднього?

Література до розділу

1. Інтелектуальні інформаційні системи: навч.посібник / А. А. Смагін, С. В. Ліпатова, А. С. Мельниченко. – Ульяновськ: УлГУ, 2010. – 136 с. (рос.)

2. Генетичні алгоритми, штучні нейронні мережі і проблеми віртуальної реальності / Г. К. Вороновський та ін. – Харків: ОС-НОВА, 1997. – 112 с.

3. Збірник завдань з курсу «Інтелектуальні інформаційні системи» навчальний посібник / С.В. Ліпатова. – Ульяновськ: УлГУ, 2010. – 64 с. (рос.)

4. Степашко В. С., Зосімов В. В., Булгакова О. С. Ітераційні алгоритми індуктивного моделювання: [монографія] – Київ, Наукова думка 2018 – 190 с.

5. Ісаєв, С. А. Популярно про генетичні алгоритми [Електронний ресурс] / С. А. Ісаєв. – Режим доступу:
http://algolist.manual.ru/ai/ga/ga1.php

6. Каширіна, І. Л. Вступ в еволюційне моделювання: навч. посібник / І. Л. Каширіна. – Воронеж, 2007. – 40 с. (рос.)

7. Стариков А. Лабораторія BaseGroup. Генетичні алгоритми – математичний апарат [Електронний ресурс] / А. Стариков. – Режим доступу: http://www.basegroup.ru/genetic/

ДОДАТКИ

Додаток А.
Тестові завдання до розділів 1–3

1. Який різновид експертних систем заснований на інтеграції різних джерел даних?
   А. Класифікуючі
   Б. Трансформуючі
   В. Мультиагентні
   Г. Не має правильної відповіді

2. Назвіть метод навчання багатошарових нейронних мереж
А. Симплекс – метод
   Б. Зворотного поширення помилки
   В. Найменших квадратів
   Г. Дисперсійний аналіз

3. Якому терміну відповідає визначення: "Інтелектуальна система, заснована на знаннях фахівця в конкретній області"?
   А. Креативно-орієнтована система
   Б. Експертна система
   В. Освітня система
   Г. Предметна система
   Д. Немає правильної відповіді

4. Яке твердження, що протиставляють експертні та креативно-орієнтовані системи, вірні?
   А. ЕС призначена для вибору рішення з відомих рішень, КОС для створення нового рішення
   Б. ЕС призначена для рішення у нестандартних ситуаціях, КОС для стандартного рішення
   В. ЕС орієнтована на творчі здібності людини, КОС заснована на шаблонному використанні знань

Г. ЕС універсальні, КОС конкретні
Д. Немає правильної відповіді

5. Яке твердження, що протиставляють експертні та креативно-орієнтовані системи, вірні?
А. ЕС конкретні, КОС універсальні
Б. ЕС призначена для рішення у нестандартних ситуаціях, КОС для стандартного рішення
В. ЕС орієнтована на творчі здібності людини, КОС заснована на шаблонному використанні знань
Г. ЕС універсальні, КОС конкретні
Д. Немає правильної відповіді

6. Яке твердження, що протиставляють експертні та креативно-орієнтовані системи, вірні?
А. ЕС заснована на шаблонному використанні знань, КОС передбачає нешаблонність
Б. ЕС призначена для рішення у нестандартних ситуаціях, КОС для стандартного рішення
В. ЕС орієнтована на творчі здібності людини, КОС заснована на шаблонному використанні знань
Г. ЕС універсальні, КОС конкретні
Д. Немає правильної відповіді

7. Експертна система включає в себе:
А. Базу знань, підказки-стимули, інструменти генерування ідей
Б. Базу знань, факти, поняття, правила, механізм прийняття рішень, користувальницький інтерфейс
В. Підказки-стимули, інструменти фіксації ідей, факти, правила, інструменти генерування ідей
Г. Інструменти комбінування ідей, користувальницький інтерфейс, поняття, правила, механізм прийняття рішень
Д. Немає правильної відповіді

8. База знань містить у собі:
А. Механізм прийняття рішень, підказки-стимули, інструменти фіксації ідей

Б. Інструменти генерування ідей, інструменти комбінування ідей
В. Призначений для користувача інтерфейс, факти, поняття
Г. Факти, поняття, правила
Д. Немає правильної відповіді

9. На відміну від бази даних, база знань включає в себе:
А. Правила прийняття рішень
Б. Підказки-стимули
В. Інструменти фіксації ідей
Г. Інструменти генерування ідей
Д. Інструменти комбінування ідей

10. За допомогою якого алгоритму найчастіше реалізується механізм прийняття рішень?
А. Якщо-То-Інакше
Б. Якщо-Висновок
В. Якщо-Ні-Так
Г. Так-Ні-Так
Д. Істина-Неправда-Істина

11. Правила прийняття рішень входять в
А. Базу даних, базу знань
Б. Базу знань, експертну систему
В. Будь-яку інформаційну систему
Г. Експертну систему, креативно-орієнтовану систему
Д. Немає правильної відповіді

12. Формальна процедура, яка гарантує отримання оптимального або коректного рішення це:
А. Алгоритм
Б. Процедура виведення
В. Режим придбання знань
Г. Немає правильної відповіді

13. Частина системи, заснованої на знаннях, або експертна система, що містить предметні знання:
А. База даних
Б. База знань

В. Програма
Г. Немає правильної відповіді

14. Інформація, необхідна програмі для того, щоб ця програма вела себе інтелектуально:
А. Факти
Б. Правила
В. Знання
Г. Немає правильної відповіді

15. Предметні знання, знання про предметну область:
А. Факт
Б. Знання
В. Правило
Г. Немає правильної відповіді

16. Метод представлення знань за допомогою мережі вузлів, які відповідають концепціям або об'єктам, пов'язаних дугами, які описують відносини між вузлами:
А. Правила
Б. Фрейми
В. Семантичні мережі
Г. Немає правильної відповіді

17. Метод подання знань, коли властивості зв'язуються з вершинами, що представляють концепції або об'єкти:
А. Правила
Б. Фрейми
В. Семантичні мережі
Г. Немає правильної відповіді

18. Ознака або властивість, які характеризують об'єкт:
А. Атрибут
Б. Ознака
В. Властивість
Г. Немає правильної відповіді

19. Процес міркування, під час якого з відомих фактів виводяться нові факти:
А. вибір
Б. повернення
В. висновок
Г. Немає правильної відповіді

20. Приховані або віртуальні процедури в системах, заснованих на знаннях, що активізуються даними:
А. перевертні
Б. привиди
В. демони
Г. Немає правильної відповіді

21. Тип систем, заснованих на знаннях, які застосовуються для знаходження причин неполадок в технічних системах або захворювань у людини:
А. Діагностичні системи
Б. Прогностичні системи
В. Проекційні системи
Г. Немає правильної відповіді

22. Зовнішня пам'ять комп'ютера або база правил в системі, заснованої на знаннях:
А. Фотографічна пам'ять
Б. КЕШ-пам'ять
В. Довгострокова пам'ять
Г. Немає правильної відповіді

23. Спеціаліст, якому знайомі змістовна сторона завдання і методи структурування знань в експертних системах:
А. Інженер знань
Б. Експерт
В. Програміст
Г. Немає правильної відповіді

24. Дисципліна, націлена на завдання побудови експертних систем; засоби і методи, що забезпечують розробку таких систем:
А. Експертологія
Б. Інженерія знань
В. Методика знання
Г. Немає правильної відповіді

25. Автоматизована інформаційна система, забезпечена інтелектуальним інтерфейсом, що дозволяє користувачеві робити запити на природній або професійно-орієнтованій мові:
А. Інтелектуальна інформаційна система
Б. Інтелектуальна навчальна система
В. Експертна система
Г. Немає правильної відповіді

26. Автоматизована навчальна система, забезпечена інтелектуальним інтерфейсом, що дозволяє в процесі навчання вести діалог, відповідати на питання і виконувати завдання на природній мові:
А. Інтелектуальна інформаційна система
Б. Інтелектуальна навчальна система
В. Експертна система
Г. Немає правильної відповіді

27. Твердження про те, що спостерігається деяке конкретне співвідношення між об’єктами, називається:
А. факт
Б. процедура
В. правило
Г. запит

28. Щоб встановити відносини між об'єктами на основі наявних фактів використовують:
А. процедуру
Б. факт
В. правила
Г. запит

29. Факт в мові Пролог ...
А. потребує доведення
Б. є завжди істинним твердженням
В. є хибним твердженням

30. В основі нейрокібернетики лежить принцип, який орієнтований на:
А. пошук алгоритмів розв'язку інтелектуальних задач
Б. розробку спеціальних мов для розв'язку задач обчислювального плану
В. апаратне моделювання структур, подібних до структури людського мозку
Г. апаратне моделювання структур, не властивих людському мозку

31. Яке з правил записаних на мові Prolog (Pie) найбільш відповідає поняттю «донька»? (тут поняття «батько» в перекладі на російську – «родитель»). Всі поняття, які входять до правила визначені.
А. донька(X,Y):- батько(Y, X), чоловік (Y).
Б. донька(X,Y):-батько(Y, X), чоловік(X).
В. донька(X,Y):-батько(Y, X), жінка(X).
Г. Немає правильної відповіді

32. Яке з правил записаних на мові Prolog (Pie) найбільш відповідає поняттю «татусева донька»? (тут поняття «батько» в перекладі на російську – «родитель»). Всі поняття, які входять до правила визначені.
А. татусева_донька(X,Y):-батько(Y, X), жінка(X), чоловік (Y).
Б. татусева_донька(X,Y):-батько(Y, X), чоловік(X).
В. татусева_донька(X,Y):-батько(Y, X), жінка(X).
Г. Немає правильної відповіді

33. Яке з правил записаних на мові Prolog найбільш відповідає поняттю «спільна дитина»? (тут поняття «батько» в перекладі на російську – «родитель»). Всі поняття, які входять до правила визначені.
А. спільна_дитина(X,Y):-батько(Y, X), жінка(X).
Б. спільна_дитина(X,Y):-батько(Y, X), жінка(X), чоловік (Y).
В. спільна_дитина(X, Y, Z):-тато(X, Z), мати(Y, Z).
Г. спільна_дитина(X, Y, Z):-тато(X, Z), мати(Y, Z), жінка(Z,X).

34. Яке з правил записаних на мові Prolog (Pie) найбільш відповідає поняттю «спільна донька»? (тут поняття «батько» в перекладі на російську – «родитель»). Всі поняття, які входять до правила визначені.

А. спільна_донька(X,Y):-батько(Y, X), жінка(X).
Б. спільна_донька(X,Y):-батько(Y, X), жінка(X), чоловік (Y).
В. спільна_донька(X, Y, Z):-тато(X, Z), мати(Y, Z).
Г. спільна_донька (X, Y, Z):-тато(X, Z), мати(Y, Z), жінка(Z,X).

35. Яке з правил записаних на мові Prolog (Pie) найбільш відповідає поняттю «дідусь»? (тут поняття «батько» в перекладі на російську – «родитель»). Всі поняття, які входять до правила визначені.

А. дідусь(X, Y):-батько(X, Y).
Б. дідусь(X, Y):-батько(X, Z), предок(Z, Y).
В. дідусь(X, Y):-батько(X, Z), батько(Z, Y), чоловік(X).
Г. дідусь(X, Y):-батько(X, Z), батько(Z, Y), чоловік(X), жінка(Z).

36. Яке з правил записаних на мові Prolog (Pie) найбільш відповідає поняттю «тато мами»? (тут поняття «батько» в перекладі на російську – «родитель»). Всі поняття, які входять до правила визначені.

А. тато_мами(X, Y):-батько(X, Y).
Б. тато_мами(X, Y):-батько(X, Z), предок(Z, Y).
В. тато_мами(X, Y):-батько(X, Z), батько(Z, Y), чоловік(X).
Г. тато_мами(X, Y):-батько(X, Z), батько(Z, Y), чоловік(X), жінка(Z).

37. Надайте правильні відповіді. Що буде виведено на екран в результаті виконання ПРОЛОГ-програми:

```
Clauses
     вчиться ("Діма", інститут).
     вчиться ("Олена", університет).
     вчиться ("Вова", школа).
     вчиться ("Лана", інститут).
     вчиться ("Олексій", університет).
     вчиться ("Віра", школа).
Goal
     вчиться (X, університет), write (X).
```


А. X
Б. Вова
В. Олена
Г. Олексій

38. Надайте правильні відповіді. Що буде виведено на екран в результаті ПРОЛОГ-програми:

```
Clauses
     грає ("Саша", футбол).
     грає ("Інна", волейбол).
     грає ("Андрій", футбол).
     грає ("Серьожа", гандбол).
     грає ("Іра", теніс).
Goal
     грає (X, футбол), write (X).
```

А. Саша
Б. Іра
В. Андрій
Г. Інна

39. Надайте правильні відповіді. Що буде виведено на екран в результаті виконання програми:

```
Clauses
     любить ("Анна", яблука).
     любить ("Оля", банани).
     любить ("Світла", шоколад).
     любить ("Маша", апельсини).
     любить ("Анна", банани).
     любить ("Оля", яблука).
Goal
     любить (X, банани), write (X).
```

А. Анна
Б. Оля

В. Маша
Г. Світлана

40. Надайте правильні відповіді. Серед перерахованих речень правилами є:
А. любить (Іра, сливи).
Б. знає (Іван, Х).
В. мати (Х, Y): – батько (Х, Y), жінка (Х).
Г. вчиться (Х, школа), вчиться (Y, школа).
Д. студент (Х): – вчиться (Х, інститут).

41. Надайте правильні відповіді. Серед перерахованих речень фактами є:
А. любить (Іра, сливи).
Б. держава (Україна).
В. мати (Х, Y): – батько (Х, Y), жінка (Х).
Г. вчиться (Х, школа), вчиться (Y, школа).
Д. студент (Х): – вчиться (Х, інститут).

Додаток Б.
Тестові завдання до розділу 4

1. На рис. 1 зображена нейронна мережа (функції активації: порогова та гіперболічний тангенс). Чому дорівнює Y_1?

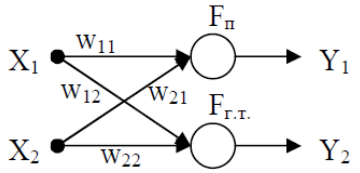


Рис. 1

А. $S_1 = x_1 w_{11} + x_2 w_{21};\ Y_1 = F_n(S_1) = \begin{cases} 1,\ S_1 \geq T, \\ 0,\ S_1, < T \end{cases}\ T = const$

Б. $S_1 = x_1 w_{11} + x_1 w_{12};\ Y_1 = F_n(S_1) = \begin{cases} 1,\ S_1 \geq T, \\ 0,\ S_1, < T \end{cases}\ T = const$

В. $S_1 = x_1 w_{11} + x_2 w_{21};\ Y_1 = F_n(S_1) = \begin{cases} 0,\ S_1 \geq T, \\ 1,\ S_1, < T \end{cases}\ T = const$

Г. Немає правильної відповіді

2. На рис. 2 зображена нейронна мережа (функції активації: порогова та гіперболічний тангенс). Чому дорівнює Y_2?

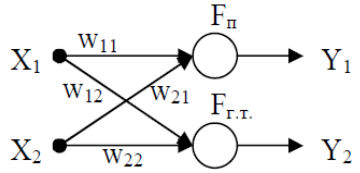


Рис. 2

А. $S_2 = x_1 w_{11} + x_2 w_{21};\ Y_2 = F_{г.т.}(S_2) = \begin{cases} 1,\ S_2 \geq T, \\ 0,\ S_2, < T \end{cases}\ T = const$

Б. $S_2 = x_1 w_{12} + x_2 w_{22};\ Y_2 = F_{г.т.}(S_2) = \dfrac{e^{s_2/k} + e^{-s_2/k}}{e^{s_2/k} - e^{-s_2/k}},\ k = const$

В. $S_2 = x_1 w_{12} + x_2 w_{22};\ Y_2 = F_{г.т.}(S_2) = \dfrac{e^{s_2/k} - e^{-s_2/k}}{e^{s_2/k} + e^{-s_2/k}},\ k = const$

Г. Немає правильної відповіді

3. На рис. 3 зображена нейронна мережа (функції активації: 1 слой – сигмоїдальна, 2 слой – лінійна). Чому дорівнює Y_1?

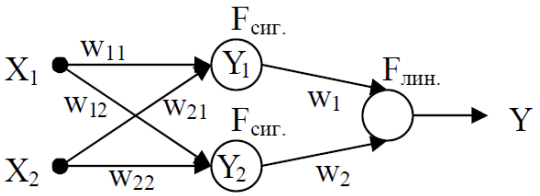


Рис. 3

А. $S_1 = x_1 w_{11} + x_2 w_{21};\ Y_1 = F_{сиг}(S_1) = \begin{cases} 1,\ S_1 \geq T, \\ 0,\ S_1, < T \end{cases}\ T = const$

Б. $S_1 = x_1 w_{11} + x_2 w_{21};\ Y_1 = F_{сиг}(S_1) = \dfrac{1}{1 + e^{-S_1 k}},\ k = const$

В. $S_1 = x_1 w_{12} + x_2 w_{22};\ Y_1 = F_{сиг}(S_1) = \dfrac{1}{1 + e^{-S_1 k}},\ k = const$

Г. Немає правильної відповіді

4. На рис. 4 зображена нейронна мережа (функції активації: 1 слой – сигмоїдальна, 2 слой – лінійна). Чому дорівнює Y_2?

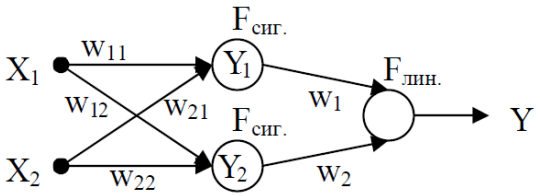


Рис. 4

А. $S_2 = x_1 w_{11} + x_2 w_{21};\ Y_2 = F_{сиг}(S_2) = \begin{cases} 1,\ S_2 \geq T, \\ 0,\ S_2, < T \end{cases}\ T = const$

Б. $S_2 = x_1 w_{11} + x_2 w_{21};\ Y_2 = F_{сиг}(S_2) = \dfrac{1}{1 + e^{-S_2 k}},\ k = const$

В. $S_2 = x_1 w_{11} + x_2 w_{22};\ Y_2 = F_{сиг}(S_2) = \dfrac{1}{1 + e^{-S_2 k}},\ k = const$

Г. Немає правильної відповіді

5. На рис.5 зображена нейронна мережа (функції активації: 1 слой – сигмоїдальна, 2 слой – лінійна). Чому дорівнює Y?

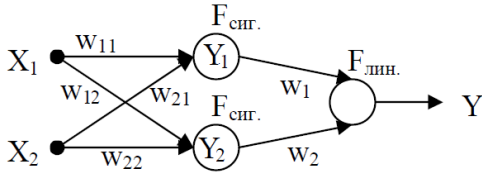


Рис. 5

А. $Y = k(Y_1 w_1 + Y_2 w_2),\ k = const$

Б. $S = x_1 w_{11} + x_2 w_{21};\ Y = F_{сиг}(S) = \dfrac{1}{1 + e^{-Sk}},\ k = const$

В. $Y = k(Y_1 w_1 k_1 + Y_2 w_2 k_2),\ k = const$

Г. Немає правильної відповіді

6. На рис. 6 зображена нейронна мережа. Якому з варіантів відповідей вона відповідає?

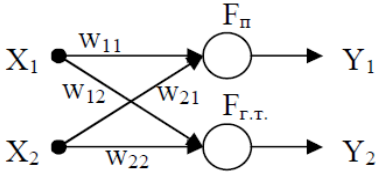


Рис. 6

А. Однослойна неоднорідна нейронна мережа, що складається з 2 нейронів і має функції активації: порогову та гіперболічний тангенс.

Б. Однослойна однорідна нейронна мережа, що складається з 2 нейронів і має функції активації: порогову та гіперболічний тангенс.

В. Однослойна однорідна нейронна мережа, що складається з 2 нейронів і має порогову функцію активації.

Г. Немає правильної відповіді

7. На рис.7 зображена нейронна мережа. Якому з варіантів відповідей вона відповідає?

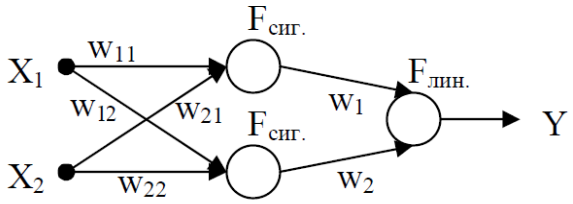


Рис. 7

А. Однослойна неоднорідна нейронна мережа, що складається з 2 нейронів і має функції активації: сигмоїдальну, лінійну.

Б. Багатошарова неоднорідна нейронна мережа, що складається з 2 слоїв, в 1-ому знаходиться 2 нейрона та використовується сигмоїдальна функція активації, в 2-ому знаходиться 1 нейрон та використовується лінійна функція активації.

В. Багатошарова однорідна нейронна мережа, що складається з 2 слоїв, в 1-ому знаходиться 2 нейрона та використовується сигмоїдальна функція активації, в 2-ому знаходиться 1 нейрон та використовується лінійна функція активації.

Г. Немає правильної відповіді

8. Прорахувати одну ітерацію циклу навчання за Δ-правилом однослойної бінарної неоднорідної нейронної мережі, що складається з 2 нейронів і має функції активації: лінійну (k =1) і порогову функцію

(T= 0,7). В якості навчальної вибірки використовувати таб. 1. (для навчання обрати вектор {0101}). Синаптичні ваги задані в таб. 2. Коефіцієнт навчання дорівнює 0,7. Надайте правильні відповіді.

Таб. 1

x_1	x_2	d_1	d_2
0	0	1	0
0	1	0	1
1	0	0	1
1	1	1	1

Таб. 2

w_{ij}	1	*2*
1	0.1	0.2
2	0.5	1

А. Y_1=0.5;

w_{ij}	1	*2*
1	0.1	0.2
2	0.85	1

Б. Y_1=0;

w_{ij}	1	*2*
1	0.1	0.2
2	0.5	1

В. Y_2=0;

w_{ij}	1	*2*
1	0.1	0.2
2	0.85	1

Г. Y_2=1;

w_{ij}	1	*2*
1	0.1	0.2
2	0.85	1

12. Прорахувати прямий прохід циклу навчання методом зворотнього поширення помилки багатослойної бінарної неоднорідної нейронної мережі, що складається з 2 слоїв, де в 1-ому слої знаходиться 2 нейрони та використовується сигмоїдальна функція активації (k =0,7), а в 2-му – 1 нейрон з лінійною функцією активації (k=1). В якості навчальної вибірки використовувати таб. 1. (для навчання обрати вектор {1001}). Синаптичні ваги для 1-го слоя задані в таб. 2, для 2-го – в таб. 3. Надайте правильні відповіді.

Таб. 1

x_1	x_2	d_1
0	0	1
0	1	1
1	0	1
1	1	0

Таб. 2

$w_{ij}(1)$	1	2
1	0.6	0.9
2	0.1	0.5

Таб. 3

w_{ij}	1	2
1	0.3	0.8

А. Для 1-го слоя: $Y_1 = 0{,}52$; $Y_2 = 0{,}61$;
Б. Для 2-го слоя: $Y = 0{,}45$;
В. Для 1-го слоя: $Y_1 = 0{,}52$; $Y_2 = 0{,}61$;
Г. Для 2-го слоя: $Y = 1$;

Додаток В.
Тестові завдання до розділу 5

1. Нехай А і В – нечіткі множини на універсальній множині Е.

$$A = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0}{x_3} + \frac{1}{x_4};\ B = \frac{0,7}{x_1} + \frac{0,9}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}.$$

Знайти перетин двох нечітких множин.

А. $A \cap B = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0}{x_3} + \frac{1}{x_4}$

Б. $A \cap B = \frac{0,7}{x_1} + \frac{0,9}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$

В. $A \cap B = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$

Г. Немає правильної відповіді

2. Нехай А і В – нечіткі множини на універсальній множині Е.

$A = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0}{x_3} + \frac{1}{x_4};\ B = \frac{0,7}{x_1} + \frac{0,9}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$. Знайти різницю двох нечітких множин *A*-*B*.

А. $A - B = \frac{0,6}{x_1} + \frac{0,8}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$

Б. $A - B = \frac{0,3}{x_1} + \frac{0,1}{x_2} + \frac{0}{x_3} + \frac{0}{x_4}$

В. $A - B = \frac{0,6}{x_1} + \frac{0,8}{x_2} + \frac{0,1}{x_3} + \frac{0}{x_4}$

Г. Немає правильної відповіді

3. Нехай А і В – нечіткі множини на універсальній множині Е.

$A = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0}{x_3} + \frac{1}{x_4};\ B = \frac{0,7}{x_1} + \frac{0,9}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$. Знайти різницю двох нечітких множин *B*-*A*.

А. $B - A = \frac{0,6}{x_1} + \frac{0,8}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$

Б. $B - A = \frac{0,3}{x_1} + \frac{0,1}{x_2} + \frac{0}{x_3} + \frac{0}{x_4}$

В. $B - A = \frac{0,6}{x_1} + \frac{0,8}{x_2} + \frac{0,1}{x_3} + \frac{0}{x_4}$

Г. Немає правильної відповіді

4. Нехай A і B – нечіткі множини на універсальній множині E.

$A = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0}{x_3} + \frac{1}{x_4}$; $B = \frac{0,7}{x_1} + \frac{0,9}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$. Знайти об'єднання двох нечітких множин.

А. $A \cup B = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0}{x_3} + \frac{1}{x_4}$

Б. $A \cup B = \frac{0,7}{x_1} + \frac{0,9}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$

В. $A \cup B = \frac{0,4}{x_1} + \frac{0,2}{x_2} + \frac{0,1}{x_3} + \frac{1}{x_4}$

Г. Немає правильної відповіді

5. Дано два нечітких відношення R_1 та R_2. Чому дорівнює $R_1 \bullet R_2$?

R_1	y_1	y_2	y_3
x_1	0.1	0.7	0.4
x_2	1	0.5	0

R_2	z_1	z_2	z_3	z_4
y_1	0.9	0	1	0.2
y_2	0.3	0.6	0	0.9
y_3	0.1	1	0	0.5

А.

$R_1 \bullet R_2$	z_1	z_2	z_3	z_4
x_1	0.3	0.6	0.1	0.7
x_2	0.9	0.5	1	0.5

Б.

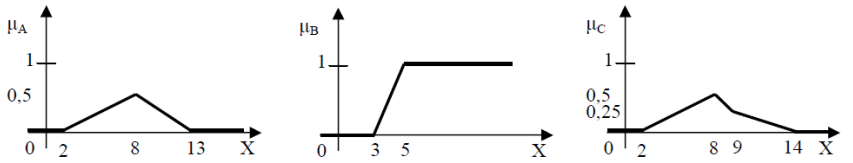
$R_1 \bullet R_2$	z_1	z_2	z_3	z_4
x_1	0.4	0.1	0.4	0.2
x_2	0.1	0.6	0	0.5

В.

$R_1 \bullet R_2$	z_1	z_2	z_3	z_4
x_1	0.4	0.1	0.3	0.2
x_2	0.1	1	0	0.5

Г. Немає правильної відповіді

6. Дано три нечітких множини A, B, C (задані їх функції належності). Визначити степінь належності елемента «8» до нечіткої множини $D = \overline{A} \cap (A \cup C \cup B)$.



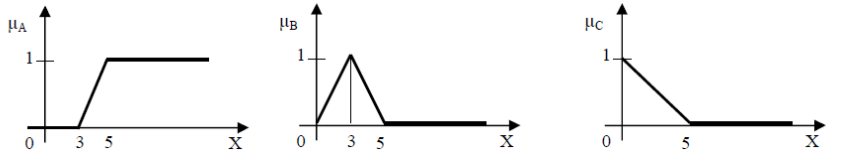
А. 0

Б. 1

В. 0.5

Г. Немає правильної відповіді

7. Дано три нечітких множини A, B, C (задані їх функції належності). Визначити степінь належності елемента «5» до нечіткої множини $D = A \cup (C \cap B)$.



А. 0

Б. 1

В. 0.5

Г. Немає правильної відповіді

НОТАТКИ

НОТАТКИ

Навчальне видання

Булгакова Олександра Сергіївна
Зосімов В'ячеслав Валерійович
Поздєєв Валерій Олександрович

МЕТОДИ ТА СИСТЕМИ ШТУЧНОГО ІНТЕЛЕКТУ: ТЕОРІЯ ТА ПРАКТИКА

Навчальний посібник

Верстка – Ю.С. Семенченко

Підписано до друку _____ р. Формат 60x84/16.
Папір офсетний. Гарнітура Times New Roman. Цифровий друк.
Ум. друк. арк. 20,69. Наклад _____. Замовлення № 2503-211.
Ціна договірна. Віддруковано з готового оригінал-макета.

Видавництво та друк: «ОЛДІ-ПЛЮС»
вул. Паровозна, 46-А, м. Херсон, 73034
Свідоцтво ДК № 6532 від 13.12.2018 р.

Тел.: +38 (0552) 399-580, +38 (098) 559-45-45,
+38 (095) 559-45-45, +38 (093) 559-45-45
Для листування: а/с 20, м. Херсон, Україна, 73021
E-mail: office@oldiplus.com