Міністерство освіти і наукиУкраїни

Рівненський державний гуманітарний університет

Г.О. Шліхта, В.А. Сяський

МЕТОДИ ТА ЗАСОБИ ІНЖЕНЕРІЇ ЗНАНЬ ДЛЯ ШТУЧНОГО ІНТЕЛЕКТУ

НАВЧАЛЬНИЙ ПОСІБНИК

2022

УДК 004.82 (075.8)

С99

Г.О.Шліхта. Методи та засоби інженерії знань для штучного інтелекту: навчальний посібник / Г.О.Шліхта, В.А. Сяський. - РДГУ. – Рівне: РВВ РДГУ, 2022. – 93 с.

Рецензенти:

Турбал Ю.В., доктор технічних наук, професор, в.о. завідувача кафедри комп'ютерних наук та прикладної математики Національного університету водного господарства та природокористування

Мартинюк П.М., доктор технічних наук, кандидат фізико-математичних наук, професор, директор Навчально-наукового інституту автоматики, кібернетики та обчислювальної техніки Національного університету водного господарства та природокористування

Затверджено Вченою радою Рівненського державного гуманітарного університету протокол № 7 від 29 вересня 2022р.

Методи та засоби інженерії знань для штучного інтелекту: навчальний посібник для здобувачів вищої освіти / за ред. В.А. Сяський, Г.О.Шліхта. - РДГУ. – Рівне: РВВ РДГУ, 2022. – 95с.

В навчальному посібнику викладено загально-теоретичні відомості про сучасні методи представлення знань, їх розробки та технології, а також розкрито питання основних інструментальних засобів для створення експертних систем та систем обробки знань. Описано технології для використання в інтелектуальних інформаційних системах. Посібник призначено для здобувачів вищої освіти, аспірантів, фахівців галузі цифрових технологій.

ЗМІСТ

ВСТУП

> Системи штучного інтелекту дозволяють успішно вирішувати надскладні проблеми, яких до створення цих систем просто не було.
>
> *З комп'ютерного фольклору*

> Дані – це лише сира нафта, що є цінною, проте без переробки не може бути реально застосована, доки не буде перетворена в газ, пластик, хімікати тощо, щоб створити цінність, яка зумовлює прибутковість; аналогічно дані треба проаналізувати та зрозуміти, щоб вони стали дійсно цінними.
>
> *Michael Palmer, «Data is the New Oil»*

Штучний інтелект – це наука про концепції, принципи та засоби, що дозволяють обчислювальним машинам робити такі речі, які людям видаються розумними. Але що ж являє собою інтелект людини? Чи є це здатністю міркувати? Чи є це здатністю засвоювати і використовувати знання? Чи є це здатністю оперувати й обмінюватися ідеями? Безсумнівно, всі ці здатності притаманні інтелекту. Однак насправді дати строге визначення цього поняття практично неможливо, тому що інтелект – це сплав багатьох навичок в галузі представлення, обробки та використання інформації.

Практично всі сфери життя та діяльності сучасної людини в тій чи іншій мірі містять компоненти, які реалізують різноманітні технології штучного інтелекту, зокрема: керування, розпізнавання, прийняття рішень, прогнозування тощо. Особливо актуальним є використання систем штучного інтелекту в стратегічних галузях та для керування складними системами (військова галузь і оборона, національна безпека, ядерні технології, космічна галузь, транспорт тощо). Для стратегічних глобальних задач використовуються потужні інтелектуальні системи, що базуються на унікальних комп'ютерних комплексах і відповідному програмному забезпеченні. Для нескладних задач інтелектуальні системи можна проектувати та реалізовувати на звичайних ЕОМ при застосуванні традиційних мов програмування.

При розробці ефективних інтелектуальних систем (інтелектуальних програм) визначальним є формалізація даних та представлення знань деякої

предметної області для подальшої їх обробки з метою породження нових знань. Під терміном *знання* (англ. knowledge) мається на увазі сукупність відомостей, що утворює цілісний опис, відповідний деякому рівню поінформованості про описуваний об’єкт, явище або процес. Використання знань (англ. knowledge deployment) означає ефективне застосування отриманих знань для досягнення конкретних переваг. Представлення знань і пошук рішень утворюють ядро штучного інтелекту. Тому важливим є освоєння принципів формулювання знань: факти, правила, семантичні мережі, фрейми, а також моделювання реальних інтелектуальних процесів.

Сучасна *інженерія знань*, яку ще називають *когнітологія*, вивчає визначальні принципи накопичення, представлення, трансформації та виведення нових знань для систем штучного інтелекту. Інженерія знань – це процес виявлення в сирих даних раніше невідомих, нетривіальних, практично корисних і доступних інтерпретації знань, необхідних для прийняття рішень у різних сферах людської діяльності. Нерідко цей процес ототожнюють з виявленням знань у базах даних (англ. Knowledge Discovery in Databases – KDD), хоча більш правильно вважати такий пошук знань лише частиною або одним із кроків великого і складного процесу. Інженерія знань тісно пов’язана із багатьма різними науковими галузями та дисциплінами: Бази даних, Інтелектуальний аналіз даних, Машинне навчання, Методи оптимізації, Статистика, Теорія інформації, Штучний інтелект тощо.

Інженерія знань для систем штучного інтелекту – це не просто область наукових досліджень, це область діяльності людей. Вона одночасно є науково-теоретичною, оскільки охоплює цілий ряд наукових дисциплін, та прикладною, оскільки фахівцю в сфері інтелектуального аналізу даних та штучного інтелекту необхідні навички алгоритмізації і програмування. Крім цього, це в якомусь сенсі навіть культура та мистецтво, оскільки виклики сьогодення вимагають постійно шукати нові шляхи вирішення інтелектуальних проблем.

В основу посібника покладено вибрані лекції з декількох дисциплін («Логічне програмування», «Інтелектуальні інформаційні системи», «Інтелектуальний аналіз даних», «Нейронні мережі», «Методи та системи

штучного інтелекту»), які на протязі останніх років читалися авторами на факультеті математики та інформатики РДГУ для студентів спеціальностей 113 Прикладна математика, 122 Комп’ютерні науки. Віднедавна компіляція цих лекцій покладена в основу викладання дисципліни «Методи та засоби інженерії даних та знань» для студентів спеціальності 015 Професійна освіта (Цифрові технології).

У посібнику висвітлені визначальні принципи представлення знань в інтелектуальних системах, властивості даних і знань, моделі представлення знань: семантичні мережі, фрейми, продукційні схеми на правилах і на логіці. Особлива увага приділена реалізації експертних систем та систем прийняття рішень, зокрема механізму логічного виведення висновку. Для різних елементів інженерії знань у посібнику наведені приклади програмної реалізації. Кожен із прикладів ілюструє можливості різних моделей представлення знань у системах штучного інтелекту для вирішення конкретних прикладних задач.

ТЕМА 1

Вступ штучний інтелект. Базові поняття штучного інтелекту і напрямів його розвитку. Історичний огляд розвитку штучного інтелекту. Основні напрями досліджень в області штучного інтелекту. Інтелектуальні інформаційні системи (ІІС) та їх основні властивості. Приклади інтелектуальних інформаційних систем

Підвищення ефективності впровадження інформаційних систем у різні галузі діяльності тісно пов'язане з їх рівнем інтелектуалізації. На сучасному етапі розвитку інформаційних систем і технологій більшість рутинних операцій з перетворення інформації вже автоматизовано і подальше підвищення ефективності роботи потребує автоматизації інтелектуальної та творчої діяльності людини.

Щоб дати визначення поняття "інтелектуальна інформаційна система" (ІІС), необхідно зупинитися на тлумаченні терміну "інтелект".

Інтелектом *(від лат. intellectus – пізнання) називають здатність міркувати, діяти цілеспрямовано, правильно реагувати на ситуацію. Відповідно, інтелектуальними задачами є задачі, для розв'язання яких немає чітко заданого алгоритму, що завжди приводить до потрібного результату, а інтелектуальною діяльністю – процес вирішення інтелектуальних задач.*

Інтелектуальним задачам властиві неповнота, неточність і суперечливість знань, а також велика розмірність простору рішень, що не дає змоги розв'язувати їх простим перебором. У таких задачах часто немає чітких критеріїв для вибору оптимального рішення, а сама задача не завжди цілком формалізується.

У зв'язку з цим потрібно розглянути термін "алгоритм". Поняття алгоритму є базовим для всіх галузей комп'ютерного програмування. Термін алгоритм (*algorithm*) у виданні словника Webster's New World Dictionary, що вийшов у 1957 р. (правда, дещо в іншому звучанні – *algorism*), трактується як стародавнє слово, що означає "виконання арифметичних операцій за допомогою арабських цифр" і походить від імені автора знаменитого перського підручника з математики IX ст. Мухаммеда аль-Хорезмі. У ширшому трактуванні алгоритм – це точний набір інструкцій, що описують послідовність дій виконавця для досягнення результату, рішення певної задачі. Іншими словами, це точний і зрозумілий, сформульований певною мовою кінцевий опис загального способу рішення певного класу задач з використанням елементарних кроків, яких знайдені відповідні алгоритми, практично не потребує інтелектуальних зусиль і тому його може здійснювати об'єкт (людина або комп'ютер), здатний виконувати елементарні операції, з яких складається алгоритм.

Автоматизована інформаційна система (АІС) – програмна реалізація конкретного алгоритму. ІІС – реалізація алгоритму, який не існує або нам не відомий. На перший погляд, маємо протиріччя. Проте це не так. Можна запрограмувати не безпосередньо сам алгоритм, а засоби, за допомогою яких

інтелектуальна система автономно за прикладами навчиться цьому алгоритму (цей прийом часто, зокрема, застосовують при розробці програмних агентів та нейромереж). Якщо алгоритм рішення задачі надто складний, то можна реалізувати його спрощений варіант, який дає можливість отримати рішення з точністю, задовільною для практичного застосування.

Історія ІІС починається з середини XX століття, що пов'язано з розвитком Штучного інтелекту як нового наукового напрямку, появою терміна «Artificial Intelligence». У зв'язку з цим, для подальшого вивчення ІІС, розуміння її призначення, потрібно детально розглянути поняття які стосуються терміну "Штучний інтелект".

Штучний інтелект *– розділ комп'ютерної лінгвістики та інформатики, що займається формалізацією проблем та завдань, які нагадують завдання, виконувані людиною.* При цьому, у більшості випадків алгоритм розв'язання завдання невідомий наперед. Точного визначення цієї науки немає, оскільки у філософії не розв'язане питання про природу і статус людського інтелекту. Немає і точного критерію досягнення комп'ютером «розумності», хоча перед штучним інтелектом було запропоновано низку гіпотез, наприклад, тест Тюринга або гіпотеза Ньюела-Саймона. Нині існує багато підходів як до розуміння задач штучного інтелекту, так і до створення інтелектуальних систем.

Одна з класифікацій виділяє два підходи до розробки штучного інтелекту:

- ***низсхідний, семіотичний*** – створення символьних систем, що моделюють високорівневі психічні процеси: мислення, судження, мову, емоції, творчість і т.д.;
- ***висхідний, біологічний*** – вивчення нейронних мереж і еволюційні обчислення, що моделюють інтелектуальну поведінку на основі менших «не інтелектуальних» елементів.

Ця наука пов'язана з психологією, нейрофізіологією, трансгуманізмом та іншими. Як і всі комп'ютерні науки, вона використовує математичний апарат. Особливе значення для неї мають філософія і робототехніка.

Штучний інтелект (ШІ) – дуже молода область досліджень, започаткована 1956 року. Її історичний шлях нагадує синусоїду, кожен «зліт» якої ініціювався деякою новою ідеєю. На сьогодні її розвиток перебуває на «спаді», поступаючись застосуванню вже досягнутих результатів в інших областях науки, промисловості, бізнесі та навіть повсякденному житті. Основні щаблі розвитку ШІ зображено у таблиці 1.

Таблиця 1. Основні історичні шаблі у дослідженнях по ШІ

№ етапу	Автори, роботи, напрямок	Основні ідеї
1.	Аристотель, „Метафізика”, розділ „Логіка”	Основа знань – вивчення самої думки. Питання істинності суджень на основі їх взаємозв’язку з іншими ствердженнями (силогізми)
2.	Декарт, „Роздуми”	Розділення розуму та фізичного світу і як наслідок – побудова ідей про світ не обов’язково відповідає предмету, що вивчається. Це основа методології ШІ, психології, вищої математики: ментальні процеси існують самі по собі і можуть вивчатись своїми засобами
3.	Лейбніц, „Обчислювальна філософія”	Система формальної логіки. Математична формалізація законів логіки
4.	Рассел, Уайтхед	Виведення з набора аксіом шляхом формальних операцій всієї математики. Аксіоми розглядались виключно як набори символів, доведення –як строго формалізовані правила маніпулювання цими наборами. Основа автоматизованих систем доведення теорем
5.	Тюрінг, „Обчислювальні машини та інтелект”	Тест Тюрінга як перевірка „інтелектуальності” системи
6.	Ньюелл, Саймон	Гіпотеза про фізичну символьну систему як модель інтелекту.
7.	Виттгенштейн, Хассерл	Альтернативний до логічного підходу: Інтелект полягає не в знанні істини, а у знанні, як вести себе в постійно змінюваному світі. Дослідження в області коннектіоністського навчання (інтелект – як явище соціуму), в основі якого моделювання архітектури реального мозку. Нейронні моделі.

Єдиної відповіді на питання чим займається штучний інтелект, не існує. Майже кожен автор, який пише книгу про штучний інтелект, відштовхується від якогось визначення, розглядаючи в його світлі досягнення цієї науки. Зазвичай ці визначення зводяться до наступних:

- штучний інтелект вивчає методи розв'язання задач, які потребують людського розуміння. Грубо кажучи мова іде про те, щоб навчити ШІ розв'язувати тести інтелекту. Це передбачає розвиток способів розв'язання задач за аналогією, методів дедукції та індукції, накопичення базових знань і вміння їх використовувати.
- штучний інтелект вивчає методи розв'язання задач, для яких не існує способів розв'язання або вони не коректні (через обмеження в часі, пам'яті тощо). Завдяки такому визначенню інтелектуальні алгоритми часто використовуються для розв'язання NP-повних задач, наприклад, задачі комівояжера.
- штучний інтелект займається моделюванням людської вищої нервової діяльності.

- штучний інтелект – це системи, які можуть оперувати з знаннями, а найголовніше – навчатися. В першу чергу мова ведеться про те, щоби визнати клас експертних систем (назва походить від того, що вони спроможні замінити «на посту» людей-експертів) інтелектуальними системами.

Існують різні підходи до створення систем штучного інтелекту. У наш час можна виділити 4 досить різних підходи:

1. **Логічний підхід**. Основою для вивчення логічного підходу слугує алгебра логіки. Кожен програміст знайомий з нею з тих пір, коли він вивчав оператор ***if***. Свого подальшого розвитку алгебра логіки отримала у вигляді числення предикатів, в якому вона розширена за рахунок введення предметних символів, відношень між ними. Крім цього, кожна така машина має блок генерації цілі, і система виводу намагається довести дану ціль як теорему. Якщо ціль досягнута, то послідовність використаних правил дозволить отримати ланцюжок дій, необхідних для реалізації поставленої цілі (таку систему ще називають експертною системою). Потужність такої системи визначається можливостями генератора цілей і машинного доведення теорем. Для того щоб досягти кращої виразності логічний підхід використовує новий напрям, його назва – нечітка логіка. Головною відмінністю цього напряму є те, що істинність вислову може приймати окрім значень так/ні (1/0) ще й проміжне значення – не знаю (0.5), пацієнт швидше за все живий, ніж мертвий (0.75), пацієнт швидше за все мертвий, ніж живий (0.25). Такий підхід подібніший до мислення людини, оскільки вона рідко відповідає так або ні.
2. Під **структурним підходом** ми розуміємо спроби побудови ШІ шляхом моделювання структури людського мозку. Однією з перших таких спроб був перцептрон Френка Розенблатта. Головною моделюючою структурною одиницею в перцептронах (як і в більшості інших варіантах моделювання мозку) є нейрон. Пізніше виникли й інші моделі, відоміші під назвою нейронні мережі (НМ) і їхні реалізації — нейрокомп’ютери. Ці моделі відрізняються за будовою окремих нейронів, за топологією зв'язків між ними і алгоритмами навчання. Серед найвідоміших в наш час варіантів НМ можна назвати НМ зі зворотнім розповсюдженням помилки, мережі Кохонена, мережі Хопфілда, стохастичні нейронні мережі. У ширшому розумінні цей підхід відомий як Конективізм. Відмінності між логічним та структурним підходом не стільки принципові, як це здається на перший погляд. Алгоритми спрощення і вербалізації нейронних мереж перетворюють моделі структурного підходу в явні логічні моделі. З іншої сторони, ще в 1943 році Маккалок і Піттс показали, що нейронна мережа може реалізувати будь-яку функцію алгебри логіки.
3. **Еволюційний підхід**. Під час побудови системи ШІ за даним методом основну увагу зосереджують на побудові початкової моделі, і правилам, за якими вона може змінюватися (еволюціонувати). Причому модель може бути

створена за найрізноманітнішими методами, це може бути і НМ, і набір логічних правил, і будь-яка інша модель. Після цього ми вмикаємо комп'ютер і він, на основі перевірки моделей відбирає найкращі з них, і за цими моделями за найрізноманітнішими правилами генеруються нові моделі. Серед еволюційних алгоритмів класичним вважається генетичний алгоритм.

4. **Імітаційний підхід**. Цей підхід є класичним для кібернетики з одним із її базових понять чорний ящик. Об'єкт, поведінка якого імітується, якраз і являє собою «чорний ящик». Для нас не важливо, які моделі в нього всередині і як він функціонує, головне, щоби наша модель в аналогічних ситуаціях поводила себе без змін. Таким чином тут моделюється інша властивість людини – здатність копіювати те, що роблять інші, без поділу на елементарні операції і формального опису дій. Часто ця властивість економить багато часу об'єктові, особливо на початку його життя.

У рамках гібридних інтелектуальних систем намагаються об'єднати ці напрямки. Експертні правила висновків можуть генеруватися нейронними мережами, а породжуючі правила отримують за допомогою статистичного вивчення. Багатообіцяючим є новий підхід, який ще називають підсилення інтелекту, де розглядають досягнення ШІ в процесі еволюційної розробки як поточний ефект підсилення людського інтелекту технологіями.

Аналізуючи історію ШІ, можна виділити наступні напрями досліджень в цій області:

Моделювання міркувань. Багато років розвиток цієї науки просувався саме цим шляхом, і зараз це одна з найрозвиненіших областей в сучасному ШІ. Моделювання міркувань має на увазі створення символьних систем, на вході яких поставлена деяка задача, а на виході очікується її розв'язок. Як правило, запропонована задача уже формалізована, тобто переведена в математичну форму, але або не має алгоритму розв'язання, або цей алгоритм надто складний, трудомісткий і т.д. В цей напрям входять: доведення теорем, прийняття рішень (теорія ігор), планування і диспетчеризація, прогнозування.

Таким чином, на перший план виходить **інженерія знань**, яка об'єднує задачі отримання знань з простої інформації, їх систематизацію і використання. Досягнення в цій області зачіпають майже всі інші напрями дослідження ШІ. Тут також необхідно відмітити дві важливі підобласті. Перша з них – *машинне навчання* – стосується процесу самостійного отримання знань інтелектуальною системою в процесі її роботи. Друга пов'язана з створенням *експертних систем* – програм, які використовують спеціалізовані бази знань для отримання достовірних висновків щодо довільної проблеми.

Великі і цікаві досягнення є в області **моделювання біологічних систем**. Сюди можна віднести кілька незалежних напрямків. Нейронні мережі використовуються для розв'язання нечітких і складних проблем, таких як розпізнавання геометричних фігур чи кластеризація об'єктів. *Генетичний підхід* заснований на ідеї, що деякий алгоритм може стати ефективнішим, якщо відбере кращі характеристики у інших алгоритмів («батьків»). Відносно новий

підхід, де ставиться задача створення автономної програми – агента, котрий співпрацює з довкіллям, називається *агентним підход*ом. А якщо належним чином примусити велику кількість «не дуже інтелектуальних» агентів співпрацювати разом, то можна отримати *«мурашиний» інтелект*.

Задачі **розпізнавання об'єктів** вже частково розв'язуються в рамках інших напрямків. Сюди відносяться розпізнавання символів, рукописного тексту, мови, аналіз текстів. Особливо слід згадати *комп'ютерне бачення*, яке пов'язане з машинним навчанням та робототехнікою. У загальному, **робототехніка** і штучний інтелект часто асоціюються одне з одним. Інтеграцію цих двох наук, створення інтелектуальних роботів, можна вважати ще одним напрямом ШІ.

У наш час у створенні штучного інтелекту (в буквальному розумінні цього слова, експертні системи і шахові програми сюди не відносяться) спостерігається інтенсивний перелом усіх предметних областей, які мають хоч якесь відношення до ШІ в базі знань. Практично всі підходи були випробувані, але до появи штучного розуму жодна дослідницька група так і не дійшла.

Застосування технологій штучного інтелекту до інформаційних систем привело до появи інтелектуальних інформаційних систем. Можна сказати, що ***ІІС*** *– природний результат розвитку звичайних інформаційних систем, зосередили в собі найбільш наукомісткі технології з високим рівнем автоматизації не тільки процесів підготовки інформації для прийняття рішень, а й самих процесів вироблення варіантів рішень, що спираються на отримані інформаційною системою дані.* ІІС здатні діагностувати стан підприємства, надавати допомогу в антикризовому управлінні, забезпечувати вибір оптимальних рішень щодо стратегії розвитку підприємства та його інвестиційної діяльності. Завдяки наявності засобів природно-мовного інтерфейсу з'являється можливість безпосереднього застосування ІІС бізнес користувачем, який не володіє мовами програмування, в якості засобу підтримки процесів аналізу, оцінки та прийняття економічних рішень. Також ІІС застосовуються для економічного аналізу діяльності підприємства, стратегічного планування, інвестиційного аналізу, оцінки ризиків та формування портфеля цінних паперів, фінансового аналізу, маркетингу і т.д.

У цьому випадку потрібен значно більший обсяг інформації як власне про підприємство, так і про його оточення, тобто природні, політичні, економічні та інші фактори, конкурентів, постачальників і т.д., а також значно складніші обчислення, необхідність врахування факторів, які слабо формалізуються, високий рівень інтерфейсу. Поставлені завдання реалізовані в системах підтримки прийняття рішень. Їх відмінна риса – значно вищий рівень «інтелекту», ніж у звичайних інтегрованих систем управління виробництвом; наявність спеціальних процедур для відбору та введення даних, у тому числі і за розкладом з різних зовнішніх систем. У системах підтримки прийняття рішень проводиться завчасне обчислення (з метою забезпечення зменшення часу реакції) агрегованих даних, часто використовуваних в запитах; використовується спеціальна організація зберігання даних, яка забезпечує

можливість багатоаспектного пошуку глибиною агрегування / дезагрегування даних. Ця технологія отримала назва сховищ і вітрин даних в поєднанні з оперативною аналітичною обробкою даних. Найбільш потужні фірми, які розробляють системи управління базами даних (СУБД) – Oracle, BASE, Microsoft – постачають на ринок системи, в які модулі підтримки прийняття рішень входять як компоненти. До їх складу входять технології штучного інтелекту, нейронні мережі, інтелектуальний аналіз даних. Об'єктно- орієнтована структура цих баз даних зробила реальністю ідеологію фреймів, розроблену в рамках штучного інтелекту. Технічні рішення, необхідні для створення повномасштабних інтелектуальних інформаційних систем - засоби ведення баз знань на основі об'єктно-орієнтованих баз даних, автоматизації формування баз знань на основі методів інтелектуального аналізу даних, повнотекстові системи пошуку і семантичні аналізатори природної мови для природно-мовного інтерфейсу – стали вироблятися серійно як програмні вироби. Не реалізованими в рамках таких СУБД поки залишаються технології реалізації правдоподібних (імовірнісних) і логічних (дедуктивних) висновків.

Спочатку ІІС використовували знання кількох експертів в кожній з областей. В даний час бази знань частково формуються у вигляді машинного навчання, використовуючи методи індукції, генетичні алгоритми і деякі інші методи добування знань. Менеджер, використовуючи таку схему, теоретично може приймати рішення більш ефективно і з меншою вартістю, ніж це зміг би зробити будь-який індивідуальний експерт в даній області. Найбільш очевидним перевагою інтеграції деяких форм штучного інтелекту в процесі прийняття рішень в порівнянні з постійним консультуванням з групою експертів зазвичай є більш низька вартість і більшу відповідність результатів завданню. На відміну від звичайних аналітичних та статистичних моделей, ІІС дозволяють отримати вирішення важко формалізованих слабо структурованих задач.

Можливість ІІС працювати зі слабо структурованими даними передбачає наявність наступних якостей:

- можливість розв’язувати завдання, описані тільки в термінах «м'яких» моделей, коли залежності між основними показниками не є цілком певними або навіть невідомими в межах деякого класу;
- здатність до роботи з невизначеними або динамічними даними, що змінюються в процесі обробки, дозволяє використовувати ІІС в умовах, коли методи обробки даних можуть змінюватися і уточнюватися в міру надходження нових даних;
- здатність до розвитку системи і видобування знань з накопиченого досвіду конкретних ситуацій, що збільшує мобільність і гнучкість системи, дозволяючи їй швидко освоювати нові галузі застосування.

Можливість використання інформації, яка явно не зберігається, а виводиться з наявних у базі даних, дозволяє зменшити об’єми збереженої актуальної інформації при збереженні багатства доступної користувачеві

інформації. Спрямованість ІІС на вирішення слабоструктурованих, погано формалізованих задач розширює область застосування ІІС.

Наявність розвинених комунікативних здібностей у ІІС дає можливість користувачеві видавати завдання системі і отримувати від неї оброблені дані та коментарі мовою, близькою до природної. Система природно-мовного інтерфейсу (СПМІ) транслює природно-мовні структури на машинний рівень представлення знань. Включає морфологічний, синтаксичний, семантичний аналіз і відповідно у зворотному порядку синтез.

Програма інтелектуального інтерфейсу сприймає повідомлення користувача і перетворює їх у форму уявлення бази знань і, навпаки, переводить внутрішнє подання результату обробки в формат користувача і видає повідомлення на необхідний носій. Найважливіша вимога до організації діалогу користувача з ІІС - природність, що означає формулювання потреб користувача з використанням професійних термінів конкретній області застосування.

Для ІІС характерні такі ознаки:

1) розвинені комунікативні здібності: можливість обробки довільних запитів в діалозі на мові максимально наближеній до природної (система природно-мовного інтерфейсу – СПМІ);
2) спрямованість на вирішення слабоструктурованих, погано формалізованих задач (реалізація м'яких моделей);
3) здатність працювати з невизначеними і динамічними даними;
4) здатність до розвитку системи і добування знань з накопиченого досвіду конкретних ситуацій;
5) можливість отримання та використання інформації, яка явно не зберігається, а виводиться з наявних у базі даних;
6) система має не тільки модель предметної області, а й модель самої себе, що дозволяє їй визначати межі своєї компетентності;
7) здатність до аддуктивних висновків, тобто до висновків за аналогією;
8) здатність пояснювати свої дії, невдачі користувача, попереджати користувача про деякі ситуаціях, що призводять до порушення цілісності даних.

Найбільшого поширення ІІС отримали для економічного аналізу діяльності підприємства, стратегічного планування, інвестиційного аналізу, оцінки ризиків та формування портфеля цінних паперів, фінансового аналізу, маркетингу.

Традиційно вважається, що ІІС містить:

1) базу даних,
2) базу знань,
3) інтерпретатор правил або машину виведення,
4) компоненту пояснення і природного мовного інтерфейсу, які забезпечують зв'язний діалог користувача і системи з поперемінним переходом ініціативи.

Відмінні особливості ІІС в порівнянні зі звичайними ІС полягають у наступному:

1) інтерфейс з користувачем на природній мові з використанням понять, характерних для предметної області користувача;
2) здатність пояснювати свої дії і підказувати користувачеві, як правильно ввести економічні показники і як вибрати відповідні до його завдання параметри економічної моделі;
3) подання моделі економічного об'єкта та його оточення в вигляді бази знань і засобів дедуктивних і правдоподібних висновків у поєднанні з можливістю роботи з неповною або неточною інформацією;
4) здатність автоматичного виявлення закономірностей бізнесу в раніше накопичених фактах і включення їх в базу знань. Застосування ІІС спільно зі стандартними методами дослідження операцій, динамічного програмування, а також з методами нечіткої логіки для планування при комплексній автоматизації діяльності підприємства, приносить принципові вигоди: реально знижуються операційні витрати; підвищується якість управлінських рішень.

Інтелектуальні інформаційні системи особливо ефективні в застосуванні до слабо структурованих завдань, в яких поки відсутня сувора формалізація, і для вирішення яких застосовуються евристичні процедури, що дозволяють в більшості випадків отримати розв'язання. Частково цим пояснюється те, що діапазон застосування ІІС надзвичайно широкий: від управління безперервними технологічним процесами в реальному часі до оцінки наслідків від порушення умов поставки товарів за імпортом.

У міру вдосконалення принципів логічного і правдоподібного виведення, застосовуваних у ІІС за рахунок використання нечіткою, модальної, часової логіки, байєсівских мереж виведення, ІІС починають проникати в високоінтелектуальні області, пов'язані з розробкою стратегічних рішень щодо вдосконалення діяльності підприємств. Цьому сприяють більш сучасні алгоритми аналізу та синтезу пропозицій природної мови, що полегшують спілкування користувача з системою. Включення до складу ІІС класичних економіко-математичних моделей, методів лінійного, квадратичного та динамічного програмування дозволяє поєднувати аналіз об'єкта на основі економічних показників з урахуванням факторів і ризиків політичних і позаекономічних чинників, оцінювати наслідки отриманих їх ІІС рішень.

Наявність у складі ІІС об'єктно-орієнтованої бази даних дозволяє однорідними засобами забезпечити зберігання та актуалізацію як фактів, так і знань.

Інтелектуальні інформаційні системи можна класифікувати за різними умовами. Базові умови та класифікація за ними наведені нижче.

1. Предметна область застосування:

- ІІС менеджменту;
- ІІС ризик-менеджменту;
- ІІС інвестицій;
- ІІС у військовій сфері та ін.

2. Ступінь автономності від корпоративної ІС або бази даних:

- автономні у вигляді самостійних програмних продуктів з власною базою даних;
- спряжені з корпоративною;
- повністю інтегровані.

3. За способом та оперативністю взаємодії з об'єктом:

- статичні ІІС,
- динамічні ІІС:
- ІІС реального часу;
- ІІС, в контур яких залучений користувач.

4. По адаптивності:

- ІІС, що навчаються, тобто системи, параметри і структура яких, можуть змінюватися в процесі навчання або самонавчання (нейронні мережі, генетичні алгоритми та ін.);
- ІІС, параметри яких змінюються адміністратором бази знань (експертні системи та ін.).

5. По моделі подання знань:

- методи резолюцій числення предикатів;
- немонотонність, модальні та тимчасові логіки;
- марківські і Баєсовські мережі виводу;
- казуальні дерева і теорія віри;
- теорія Демпстера-Шейфера;
- нечіткі системи.

Приклади інтелектуальних інформаційних систем:

Intelligent Hedger – заснований на знаннях підхід в задачах страхування від ризику від фірми Information System Departm, New York University. Проблема величезної кількості постійно зростаючих альтернатив страхування від ризиків, швидке прийняття рішень менеджерами по ризиках, а також нестача машинної підтримки в процесі страхування від ризиків припускає різні оптимальні рішення для менеджерів по ризику. У даній системі розробка страхування від ризику сформульована як багатоцільова оптимізаційна задача, яка включає кілька складнощів, з якими існуючі технічні рішення не справляються.

Система міркувань у прогнозуванні обміну валют. Фірма: Department of Computer Science City Polytechnic University of Hong Kong. Представляє новий підхід у прогнозуванні курсів валют, заснований на акумуляції та міркуваннях з підтримкою ознак, присутніх для фокусування на наборі гіпотез про рух обмінних курсів. Представлений у прогнозуючої системі набір ознак – це заданий набір економічних значень і різні набори змінюються в часі параметрів, використовуваних в моделі прогнозування. Короткі характеристики: математична основа застосованого підходу базується на теорії Демпстера-Шейфера.

Nereid – система підтримки прийняття рішень для оптимізації роботи з валютними опціонами. Фірма: NTT Data, The Tokai Bank, Science University of Tokyo. Система полегшує дилерську підтримку для оптимальної відповіді як один з можливих представлених варіантів. Короткі характеристики: система розроблена з використанням фреймової системи CLP, яка легко інтегрує фінансову область в додаток ШІ. Запропоновано змішаний тип оптимізації, що поєднує евристичні знання з технікою лінійного програмування. Система працює на Sun-станціях.

PMIDSS – система підтримки прийняття рішень при управлінні портфелем. Розробники: Фінансова група Нью-Йоркського університету. Вирішуються завдання: вибір портфеля цінних паперів; довгострокове планування інвестицій. Короткі характеристики: змішана система подання знань, використання різноманітних механізмів виведення: логіка, спрямовані семантичні мережі, фрейми, правила.

ТЕМА 2

Способи представлення задач і пошук їх розв'язку. Представлення задач у просторі станів. Представлення, що зводить задачі до підзадач. Представлення задач у вигляді теорем. Методи пошуку в просторі станів. Пошук розв'язань при зведенні задач до під задач

Термін «розв'язання задач» вживається в штучному інтелекті в досить обмеженому змісті. Мова йде про добре визначені задачі, розв'язувані на основі пошукових алгоритмів.

Задача вважається добре визначеною, якщо для неї є можливість задати простір можливих розв'язань (станів), а також спосіб перегляду цього простору з метою пошуку кінцевого (цільового) стану,що відповідає розв'язуваній задачі. Пошук кінцевого стану задачі полягає в застосуванні до кожного стану алгоритмічної процедури з метою перевірки, чи не є цей стан розв'язанням задачі. Дана пошукова процедура триває доти, поки не буде знайдене розв'язання. Прикладами добре певних задач є доведення теорем, пошук маршруту на карті і т. д.

При виборі способу представлення задач звичайно враховують дві обставини: представлення задачі має досить точно моделювати реальність; спосіб представлення повинен бути таким, щоб розв'язувану задач було зручно з ним працювати.

Оскільки розв'язувачем задач в ШІ є комп'ютер, то необхідно використати способи представлення задач у вигляді, зручному для вирішення на ЕОМ. До таких способів належать такі:

- представлення задач у просторі станів;
- представлення, що зводить задачі до підзадач;
- представлення задач у вигляді теорем.

Підхід, шо базується на використанні простору станів, досить розповсюджений при розв'язанні інтелектуальних задач. Він передбачає:

- задання початкових станів задачі;
- задання кінцевих (цільових) станів задачі;
- задання операторів, що перетворюють одні стани задачі в інші.

Розв'язання задачі полягає в тому, щоб знайти послідовність операторів, що перетворять початковий стан задачі в кінцевий, котрий відповідає розв'язуваній задачі.

У загальному випадку задача, поставлена як задача пошуку в просторі станів, визначається сукупністю чотирьох складових (S_0, S, F, G), де

S — множина станів:

S_0 — множина початкових станів, $S_0 \in S$;

F — множина операторів, що перетворюють одні стани в інші;

G — множина цільових станів, $G \in S$.

Кожен оператор $f \in F$ є функцією, що відображає один стан в інший — $s_j = f(s_i)$, де $s_i, s_j \in S$. Розв'язанням задачі є послідовність операторів $f_i \in F$, що перетворюють початкові стани в кінцеві, тобто $f_n(f_{n-1}(\ldots(f_1(S_0))\ldots)) \in G$

Якщо така послідовність не одна й заданий критерій оптимальності, то можливий пошук оптимальної послідовності.

Зручно пошук розв'язань у просторі станів представляти у вигляді графу станів. Множина вершин графу відповідає станам задачі, а множина дуг (ребер) — операторам. У цьому випадку пошук розв'язання задачі може інтерпретуватися як пошук шляху на графі.

Іноді розглядають пошук з обмеженнями. Обмеження представляють деякі умови, які повинні виконуватися в ході досягнення цільового стану.

Граф станів задачі може бути заданий неявно. У цьому випадку за дається множина початкових станів S_0 і множина операторів F, які, будучи застосовні до вершини графу, дають всі її дочірні вершини. Процес застосування операторів до деякої вершини з метою одержання всіх її дочірніх вершин називається розкриттям вершин. При цьому можуть задаватися умови застосовності оператора до вершини (стану).

Основна ідея методу представлення зведення задач до підзадач полягає в розбитті основної задачі на підзадачі, які, у свою чергу, теж розбиваються на підзадачі, й так доти, поки початкова задача не буде зведена до елементарних підзадач, що мають розв'язання, або поки не буде доведено, що задача не має розв'язання. Процес розбиття (декомпозиція) задачі на підзадачі називається **редукцією**. Тому представлення задачі у вигляді множини підзадач — це представлення в системі редукцій, а пошук розв'язання — це пошук у просторі описів задач. Розбиття на підзадачі дає перевагу в тому випадку, коли отримані підзадачі взаємно незалежні і розв'язувати їх можна окремо одна від одної.

Простір задач представляється у вигляді орієнтованого графу, що зветься **графом редукції задачі**. З кожною вершиною цього графу зв'язується певна підзадача, а дуги відповідають операторам породження підзадач. У графі редукції задачі існують два типи вершин: **кон'юнктивні** й **диз'юнктивні**. Кон'юнктивиі вершини, або вершини типу «**І**», указують, що для розв'язання завдання потрібно розв'язати всі її підзадачі. Диз'юнктивні вершини — вершини типу «**АБО**» — відповідають альтернативним підзадачам. Для розв'язання задачі, шо представляється такою вершиною, досить розв'язати тільки одну з її підзадач. Граф редукції задачі є графом типу «**І—АБО**». З кожною вершиною такого графу можна зв'язати вираз у вигляді булевої функції, і тому його також називають **пропозиціональним графом**.

Граф редукції задачі відрізняється від графу станів тільки тим, що в ньому є «І»—вершини. Якщо «І»—вершини відсутні, то граф редукції трансформується в граф простору станів.

Початкова вершина (**корінь** «І—АБО» дерева) представляє початкову задачу, яку слід розв'язати. Вершини, які відповідають задачам, розв'язання

яких відомо, звуться заключними. Задачі й відповідні вершини, що не мають розв'язань, звуться тупиковими. Заключні вершини є розв'язними, а тупикові — нерозв'язними. Розв'язними є також І-вершини, в яких розв'язні всі дочірні вершини, і АБО-вершини, в яких розв'язна хоча б одна дочірня вершина. Мета пошуку на І—АБО графі — показати, що початкова вершина розв'язна. Для цього виконується побудова спеціального графу, що зветься графом розв'язання. Граф розв'язання — це підграф графу редукції задачі з початковою вершиною, що відповідає початковій задачі, яка містить розв'язні вершини. Перегляд графу розв'язання в напрямку від заключних вершин до початкової вершини визначає безліч підзадач, які необхідно розв'язати, і порядок їхнього розв'язання.

При представленні задач у формі доведення теорем можливі стани задачі, включаючи початкові й цільові стани, розглядаються як правильно побудовані формули вирахування предикатів. Оператори, що відображають один стан в інший, розглядаються як правила, які ви водять одне правильно побудоване вираження з іншого. Процес розв'язання задачі в цьому випадку відбувається так:

- формується множина вихідних істинних тверджень (аксіом) відповідно до умови задачі, що записуються формулами мови числення предикатів;
- будується гіпотеза щодо результату розв'язання задачі, що теж записується мовою числення предикатів, її називають цільовим твердженням;
- аксіоми комбінуються між собою на основі припустимих правил виводу з метою одержання множини нових тверджень;
- перевіряється, чи не містить множина нових тверджень цільове твердження або його заперечення; якщо містить, то теорема вважається доведеною і процес розв'язання задачі на цьому закінчується; якщо цільове твердження або його заперечення не входять у множину знову сформованих тверджень, то отримана множина тверджень поєднується з множиною вихідних аксіом і процес повторюється.

Такий процес пошуку розв'язань являє собою логічну формалізацію ходу розв'язання завдання в просторі станів методами повного перебору. Такий метод гарантує знаходження розв'язання, якщо воно існує. Але безпосереднє його застосування обмежене, тому що повний перебір може зажадати більших ресурсів часу й пам'яті.

Для керування пошуком використовують різні стратегії вибору пара аксіом або проміжних тверджень, що беруть участь у виводі, що дозволяє внести систематичність у процес пошуку й скоротити число можливих комбінацій різних тверджень у процесі виводу.

Методи пошуку в просторі станів розділяються на дві групи: методи «сліпого» й впорядкованого (евристичного) пошуку. Перевагою методів «сліпого» пошуку є простота алгоритмічної реалізації й гарантованість одержання розв'язання, якщо воно існує».

Це приводить до того, що в методах «сліпого» пошуку виконується повний перегляд усього простору станів. Для нетривіальних задач такий перегляд неможливий через занадто великий розмір простору станів. Ефективним засобом боротьби з комбінаторною складністю є методи евристичного пошуку, що дозволяють істотно знизити обсяг пошуку в більших просторах розв'язань.

До методів **«сліпого»** пошуку можна віднести такі:

— випадковий пошук;

— пошук у «глибину й ширину»;

— алгоритм рівних цін.

Випадковий пошук реалізується простим алгоритмом. Інформованість при випадковому пошуку обмежується топологією графу простору станів. Стратегія пошуку полягає в тому, що, починаючи з початкового стану, випадково вибирається оператор, що здійснює перехід до наступного стану завдання. Пошук триває доти, поки не буде виконаний перехід у цільовий стан. Випадковий вибір оператора не гарантує збіжність алгоритму. Однак такий метод пошуку забезпечує розв'язання простих задач з обмеженим простором пошуку при наявності шляху між початковою й цільовою вершиною.

Тому що граф станів обстежується випадковим образом, без запам'ятовування пройденого маршруту, можливий багаторазовий вибір тих самих станів. Поліпшити процедуру пошуку можна введенням певної систематичності вибору переходів стану.

Методи пошуку у **«глибину й ширину»** виключають марний повторний вибір тих самих вершин.

Для цього ведеться відповідно облік уже розкритих вершин і вершин, що підлягають розкриттю. Крім того, задається певний порядок (систематичність) обстеження графу стану, що дозволяє послідовно пройти всі маршрути по одному разу, без пропусків і повторень.

Суть пошуку в глибину (ПВГ) полягає в послідовному обстеженні кожної гілки графу-дерева станів. При досягненні кінцевої вершини обстежуваної гілки, що не задовольняє умову цільової вершини, здійснюється повернення до першої по дорозі назад точці розгалуження й обстежиться нова гілка. Процедура триває доти, поки не буде досягнута цільова вершина, якщо вона існує. Для побудови дороги назад (із цільової вершини в початкову) всі дочірні вершини забезпечуються покажчиками на відповідні батьківські вершини.

Основний недолік ПВГ полягає в тім, що при дослідженні дерева розв'язань із великою ймовірністю можна пройти повз ту галузь, на якій раніше всього з'являється остаточне розв'язання. Коли ПВГ не підходить, корисним може виявитися пошук завширшки.

При ПВГ вибирається певний шлях по дереву й виробляється його продовження до одержання розв'язання або доти, поки подальше продовження виявиться неможливим. Тоді процес пошуку відновляється від найближчої вершини дерева, що має суміжними досліджені вершини.

При використанні методів пошуку завширшки (ПВШ) розгалуження відбувається від рівня до рівня. Причому, якщо на першому рівні початкова задача Q розбивається на підзадачі $Q_1, Q_2, ..., Q_n$, то кожна з них досліджується раніше, ніж підзадачі 2-го рівня. Завдання першого рівня, які важкі для розв'язання, розбиваються на підзадачі другого рівня, і процес триває аналогічно.

При ПВШ всім альтернативам у кожній вершині дерева досліджуваного рівня приділяється рівна увага. Пошук завширшки ефективно використовується при встановленні ізоморфізму й ізоморфного вкладення графів, при розбивці й розміщенні схем.

При використанні цього методу гарантується знаходження шляху з мінімальною кількістю дуг (ребер) між вихідною й цільовою вершинами.

Алгоритм рівних цін. Інформованість про розв'язування задачі тут вище, ніж у випадку пошуку в глибину й ширину. Кожному операторові, що перетворює стан n_i. у стан n_j., ставиться у відповідність деяка функція вартості $c(n_i, n_j)$,яка характеризується позитивними значеннями. У ході пошуку шляху на графі враховуються сумарні витрати для досягнення тієї або іншої вершини тобто вартість шляху до цієї вершини. Вартість шляху до вершини n_j може бути визначена за такою формулою:

$$g(n_j) = g(n_i) + c(n_i, n_j)$$

де $g(n_i)$ — вартість шляху з початкової вершини у вершину n_j,

У ході пошуку вершини сортуються в порядку збільшення вартості. Щораз розкривається вершина з найменшою вартістю. Це дозволяє знайти шлях мінімальної вартості з початкової вершини до цільової.

Розглянута процедура пошуку гарантує знаходження шляху міні мальної вартості, якщо граф кінцевий й існує шлях з початкової вершини до цільової. Однак сам процес не є оптимальним, тому що пошук виконується без врахування положення цільових вершин.

Основна ідея **евристичного пошуку** складається у використанні додаткової інформації для керування процесом пошуку. Ця додаткова інформація формується на основі емпіричного досвіду, здогадів й інтуїції дослідника, тобто евристик. Використання евристик дозволяє скоротити кількість варіантів, що переглядають, при пошуку розв'язання задачі, шо веде до більш швидкого досягнення мети.

В алгоритмах евристичного пошуку список відкритих вершин упорядковується по зростанню деякої оцінної функції, формованої на основі евристичних правил. Оцінна функція може включати дві складові, одна з яких називається евристичною й характеризує близькість поточних і цільової вершин. Чим менше значення евристичної складової оцінної функції, тим ближче розглянута вершина до цільової вершини. Залежно від способу формування оцінної функції виділяють такі алгоритми евристичного пошуку:

алгоритм **«підйому на гору»**, **алгоритм глобального обліку відповідності мети**, **А-алгоритм**.

Алгоритм «підйому на гору» здійснює цілеспрямований пошук у напрямку найбільшого спадання евристичної оцінної функції $\hat{h}(n)$. Дана функція забезпечує оцінку (прогноз) вартості найкоротшого шляху від поточної вершини n до найближчої цільової вершини, тобто є мірою вартості шляху, що залишився. Чим менше значення $\hat{h}(n)$, тим більше «перспективний» шлях, на якому перебуває вершина n.

На черговому кроці алгоритму для кожної з дочірніх вершин $n_1, n_2, \ldots, n_m$ вершини n обчислюється значення оцінної функції $\hat{h}(n_j)$ для продовження шляху вибирається вершина з найменшим значенням $\hat{h}(n_i)$. Процедура пошуку зазнає невдачі, якщо для всіх дочірніх вершин

$\hat{h}(n_i) > \hat{h}(n)$.

Існують різні способи побудови евристичної оцінної функції, однак готових рецептів немає. При розв'язанні кожної конкретної задачі використовують раніше накопичений досвід розв'язання подібних задач.

Алгоритм «підйому на гору», хоча й забезпечує прискорення пошуку, але не гарантує досягнення цільової вершини. Передчасне завершення алгоритму відбувається у таких випадках: якщо в процесі пошуку зустрічаються локальні мінімуми оцінної функції $\hat{h}(n)$; якщо для групи сусідніх вершин оцінки $\hat{h}(n)$ рівні між собою.

Глобальний облік відповідності мети. Цей алгоритм пошуку розв'язання завдання схожий на попередній. Тут також оцінюється «перспективність» того або іншого шляху за допомогою евристичної оцінної функції $\hat{h}(n)$. Відмінність полягає в тому, що на кожному етапі виконується не локальне порівняння вершин, отриманих при розкритті поточної вершини, а глобальне порівняння всіх вершин кандидатів. Для цього список вершин, що підлягають розкриттю, сортується в порядку зростання значення $\hat{h}(n)$. Найкраща вершина для продовження шляху буде знаходитися на першому місці.

Алгоритм дозволяє успішно справлятися із проблемою локальних мінімумів, однак при цьому знижується ефективність самого пошуку. Ефективність процесу пошуку буде залишатися високою, якщо оцінка вартості найкоротшого шляху $\hat{h}(n)$ з вершини я у цільову вершину буде якнайближче до реальної вартості $h(n)$.

А-алгоритм. Достоїнство алгоритму рівних цін полягає в тому, шо він гарантує знаходження оптимального шляху. Достоїнством алгоритмів, шо використовують оцінку майбутніх витрат $\hat{h}(n)$, є можливість зрізання дерева пошуку, що підвищує ефективність пошуку. Природним є прагнення комбінувати ці алгоритми й ураховувати як зроблені, так і майбутні витрати. Цього можна домогтися, якщо скористатися оцінною функцією

$f(n)=\hat{g}(n)+\hat{h}(n)$

де і $\hat{g}(n)$ — оцінка вартості шляхи з початкової вершини у вершину n; $\hat{h}(n)$ — оцінка вартості найкоротшого шляху з вершини п у цільову вершину.

Алгоритм, що базується на використанні оцінної функції $f(n)$ зветься А-алгоритмом. Оцінки $\hat{h}(n)$ у ході пошуку не змінюють своїх значень. Оцінки $\hat{g}(n)$ також не змінюють своїх значень і завжди дорівнюють реальним витратам $g(n)$, якшо пошук шляху виконується на дереві станів. Однак при пошуку шляху на графі станів оцінки $\hat{g}(n)$ можуть змінюватися. Отже, у загальному випадку оцінки $f(n)$ змінюють свої значення в процесі пошуку.

А-алгоритм забезпечує пошук оптимального шляху, якщо виконується умова $\hat{h}(n)\le h(n)$. У зворотному випадку А-алгоритм не гарантує одержання оптимального розв'язання.

Пошук розв'язань у просторі станів може виконуватися в декількох напрямках:

— з початкового стану в цільовий стан (пошук, керований даними);

— із цільового стану в початковий стан (пошук, керований метою, або зворотний пошук);

— одночасно у двох напрямках (комбінований пошук).

Тому шо пошукові графи (дерева) для практичних задач великі, двоспрямований пошук може бути ефективнішим за односпрямований.

Пошук розв'язань при зведенні задач до під задач. При такому поданні завдання пошук розв'язань виконується з використанням І-АБО графів. Розв'язання в І-АБО графах відповідає підграфу або піддереву, тому шо при розкритті І-вершини необхідно визначити, чи є розв'язними всіїї дочірні вершини. Наявність АБО вершини приводить до того, що в ході пошуку розв'язання розглядають декілька можливих під графів розв'язань. Якшо необхідно знайти оптимальний граф розв'язань, то вводяться оцінні функції. Ці функції представляють вартості графів, зокрема графу розв'язання або його підграфу.

У процесі пошуку будуються підграфи, які виступають у ролі претендентів на участь у формуванні повного графу розв'язання. Такі підграфи називають потенційними підграфами розв'язань. Звичайно в ході пошуку будуються декілька потенційних підграфів, які певним чином представляються в пам'яті ЕОМ. Побудову зазначених підграфів виконують повторенням двох узагальнених дій:

— визначають, чи є вершини потенційного підграфу розв'язання розв'язними, нерозв'язними (тупиковими) або нерозкритими; якщо необхідно, то обчислюють вартості відповідних підграфів;

— розширюють потенційний підграф розв'язань.

яких відомо, то вони вважаються заключними й позначаються як розв'язні. Якщо кінцеві вершини є тупиковими, то вони позначаються як нерозв'язні. Потім аналізуються їхні батьківські вершини. Батьківська І-вершина позначається як розв'язна, якщо розв'язними є всі її дочірні вершини, інакше вона позначається

як нерозв'язна. Батьківська АБО-вершина одержує мітку, що збігається із влучною дочірньою вершиною, розглянутою в цей момент.

Якщо початкова вершина потенційного підграфу розв'язань позначається як розв'язна, то даний потенційний підграф стає можливим підграфом розв'язання.

У випадку пошуку оптимального графу розв'язання оцінюється вартість всіх вершин потенційних підграфів розв'язань, починаючи з кінцевих вершин.

При виконанні другої дії розкривають кінцеві вершини потенційного підграфу розв'язань, які не мають міток. Якшо кінцева вершина, шо розкриває, є І-вершиною, то генерується новий потенційний підграф шляхом додавання всіх дочірніх вершин до розширюваного потенційного підграфу. Якшо кінцева вершина, що розкриває, — це АБО-вершина, то до розширюваного потенційного підграфу додається тільки одна дочірня вершина. При цьому формується стільки по тенційних підграфів, скільки є вершин.

Аналогічно пошуку в просторі станів при пошуку в просторі редукції завдання виділяють алгоритми сліпого й упорядкованого (евристичного) пошуку.

Алгоритм пошуку в глибину. Для організації роботи цього алгоритму створюється список потенційних підграфів розв'язань, у якому містяться підграфи, початкова вершина яких не має мітки або має мітку можливості розв'язання (підграф розв'язання). У процесі пошуку знову формовані потенційні підграфи розв'язань уміщуються в початок цього списку. Для визначеності назвемо цей список ПОТЕНЦІАЛ.

Алгоритм пошуку в глибину може бути описаний у такий спосіб:

1. У список потенціал уміщується перший потенційний підграф, що складається тільки з початкової вершини S.

2. Здійснюється розмітка даного підграфу шляхом створення нових підграфів, кожний з яких формується з початкової вершини і її дочірніх вершин

3. Отримані підграфи вважаються потенційними підграфами розв'язань і уміщуються в список ПОТЕНЦІАЛ.

4. З отриманого списку взяти перший підграф і перевірити, чи не є кінцева вершина заключною. Якщо так, то початкова вершина вважається розв'язною й даний потенційний підграф стає можливим підграфом розв'язань. Процес припиняється. Якщо кінцева вершина цього підграфу є тупиковою, то підграф з подальшого розгляду виключається й здійснюється перехід до аналізу підграфу, що стоїть наступним в списку ПОТЕНЦІАЛ.

Якшо кінцева вершина поточного підграфу підлягає подальшому розширенню, здійснюється її розкриття й отримані нові підграфи уміщуються в початок списку ПОТЕНЦІАЛ.

При цьому потрібно враховувати, шо при розкритті І-вершини одержуваний потенційний підграф містить всі її дочірні вершини.

5. Процедура повторюється до ухвалення розв'язання. Якщо кінцеві вершини всіх сформованих потенційних графів є тупиковими, то завдання є нерозв'язним.

Недоліком розглянутої процедури є те, що в пам'яті ЕОМ необхідно зберігати кілька потенційних графів розв'язань. Якщо граф редукції складний, то це приводить до неефективного використання пам'яті ЕОМ. Для усунення зазначеного недоліку необхідно при розкритті чергової вершини потенційного підграфу розв'язання розглядати не всі дочірні вершини, а тільки одну.

Коли вершина позначається міткою нерозв'язна, то вона виключа ється з потенційного підграфу розв'язання. У підсумку в підграф розв'язання включаються тільки вершини, позначені міткою розв'язна. Щораз, коли виявлена нерозв'язна вершина, процедура виконує повернення до батьківської вершини, забезпечуючи обстеження інших альтернативних галузей графу редукції задачі. Якщо виявлено розв'язну вершину й при цьому ще не закінчена побудова підграфу розв'язання, то повернення виконується з метою обстеження кон'юнктивних галузей, шо залишилися. Завдяки поверненню немає необхідності розкривати всі дочірні вершини одночасно й запам'ятовувати альтернативні потенційні підграфи розв'язань.

Алгоритм евристичного пошуку в І-АБО графі. Алгоритм евристичного пошуку припускає впорядкування списку ПОТЕНЦІАЛ відповідно до деякої евристичної оцінної функції. На відміну від пошуку в просторі станів, де евристична функція являла собою оцінку вартості шляху між поточними й цільовою вершинами, при пошуку в І-АБО графі евристична функція відповідає вартості графу розв'язання.

Позначимо через h/n (n) вартість підграфу розв'язання, що починається у вершині n.

Вартість підграфу розв'язання визначається рекурсивно:

а) якшо n відповідає заключній вершині, то h (n)=0:

б) для будь-якої іншої вершини

$$h(n) = \sum^{k} [h(n_i) + c(n, n_i)]$$

де n_i, i= 1,2,..., k-дочірні вершини для вершини n;c(n, n_i) —вартість дуги між вершинами n і n_i. Визначення вартості підграфу розв'язання починається із заключних вершин і закінчується початковою вершиною.

У ході пошуку підграфу розв'язання розглядаються потенційні підграфи розв'язань, для яких кінцеві вершини — це, як правило, ще нерозкриті вершини. Дія визначення вартості потенційних під графів розв'язань уводяться оцінки h (n), позначувані як $\hat{h}(n)$. При цьому $\hat{h}(n) = 0$, якшо n — заключна вершина, і

$$\hat{h}(n)=\sum_{i=1}^{k}(\hat{h}(n_i)+c(n_i,n))$$

де n — розглянута вершина. Оцінка $\hat{h}(n)$ будується на основі евристичної інформації.

Евристичний алгоритм упорядкованого пошуку можна представити у вигляді процедури:

1. Помістити в список ПОТЕНЦІАЛ потенційний граф Р, що складається тільки з початкової вершини S.
2. Виконати розмітку графу Р.
3. Обчислити оцінку $\hat{h}(n)$.
4. Якщо ПОТЕНЦІАЛ — не порожній список, то виконати процедуру УСПІХ, якщо ПОТЕНЦІАЛ — порожній список,то ВИХІД (невдача).

Процедура УСПІХ

4.1. Вибрати перший граф (Р), якщо цей граф розв'язний, то граф Р зі списку видалити.

4.2. Розширити граф Р, виконати розмітку всіх знову сформованих потенційних графів P_i., визначити оцінки $\hat{h}(P_i)$.

4.3. Внести потенційні графи P_i. в список ПОТЕНЦІАЛ з урахуванням оцінок $\hat{h}(P_i)$.

4.4. Закінчення процедури УСПІХ.

5. Вихід (НЕВДАЧА).

6. Закінчення.

У розглянутій процедурі використовується певний порядок розкриття вершин. Якщо вершина n, що розкривається, є диз'юнктивною, то в новий потенційний підграф розв'язання включається тільки одна її дочірня вершина. Якщо вершина, шо розкривається, кон'юнктивна, то в новий потенційний підграф розв'язання включаються всі її дочірні вершини. Надалі, у першу чергу, розкривається та з дочірніх вершин, шо має максимальну оцінку $\hat{h}(n_i)$.

Підставою такого вибору є те, що необхідно якомога швидше з'ясувати, розв'язна або не розв'язна батьківська І-вершина n. І-вершина нерозв'язна, якщо нерозв'язна хоча б одна з її дочірніх вершин. Якщо цей факт буде встановлений, то підграф потенційного графу розв'язань, що починаються у вершині n з подальшого розгляду виключається. Імовірність виявлення не розв'язності І-вершини підвищується, якщо рухатися в напрямку максимальної оцінки й $\hat{h}(n_i)$.

Для підвищення ефективності використання пам'яті ЕОМ припошуку в графах редукції завдань із більшим числом вершин можна включити процедуру зберігання тільки однієї дочірньої вершини, що розкривається.

Якщо алгоритм пошуку знаходить оптимальний граф розв'язання (такий граф має мінімально можливу оцінку $\hat{h}(S)$, то він називається

гарантуючим. Алгоритм пошуку в І-АБО графі буде гарантуючим, якщо $\hat{h}(n) \le h(n)$, і вартості всіх дуг $c(n, n_i)$ більше деякої позитивної величини.

Якщо покласти вартість всіх дуг такою, що дорівнює нулю, то будуть мінімізуватися тільки майбутні витрати.

ТЕМА 3

Представлення знань в інтелектуальних системах. Властивості знань і даних. Дані і знання основні визначення. Властивості знань і даних. Мови представлення даних. Формалізація поняття знання. Моделі представлення знань

Представлення знань і пошук розв'язків утворюють ядро штучного інтелекту. При розробці інтелектуальних систем (експертних систем, систем підтримки прийняття рішень (СППР) та ін.) використовуються **дані** та **знання** з тієї предметної області, для якої їх створюють з метою розв'язування прикладних інтелектуальних задач.

Дані — це окремі факти, що характеризують об'єкт, процеси та явища предметної галузі, а також їх властивості. При розробці, проектуванні інтелектуальних систем дані проходять етапи трансформації від більш узагальнених множин до більш вузьких, конкретизованих множин, необхідних для розв'язування прикладних задач:

- D_1 – дані, як результат вимірювань та спостережень;
- D_2 – дані на матеріальних носіях інформації (таблиці, протоколи, довідники);
- D_3 – моделі (структури) даних у вигляді діаграм, графіків, функцій;
- D_4 – дані в комп'ютері на мові опису даних;
- D_5 – бази даних на комп'ютеризованих (електронних) носіях інформації.

Знання базуються на даних, що отримані емпіричним шляхом. Вони є результатом інтелектуальної діяльності людини, що спрямована на узагальнення досвіду, отриманого під час практичної діяльності.

Знання — це закономірності предметної галузі (принципи, зв'язки, закони), одержані в результаті практичної діяльності та професійного досвіду людини, що дозволяють фахівцям формулювати та вирішувати задачі з цієї галузі.

В даний час у штучному інтелекті не існує строго формалізованого визначення поняття «знання». Більшість фахівців, розробників інтелектуальних систем використовують таке визначення: **знання** — це добре структурована інформація, що зберігається в системі і містить усі відомості про предметну область та правила виводу, що необхідні для розв'язку безлічі завдань інтелектуальної системи. Під час проектування та розробки інтелектуальної системи знання проходять аналогічну даним трансформацію – від більш узагальнених множин до більш вузьких, конкретизованих для даної предметної області:

- R_1 – знання в пам'яті людини, як результат мислення;
- R_2 – знання на матеріальних носіях інформації (довідники, підручники та ін.);

- R_3 – поле знань – умовний опис основних об'єктів предметної області, їх атрибутів та закономірностей, що їх пов'язують;
- R_4 – формалізовані знання, що описані на специфічній мові представлення знань (продукційні моделі, семантичні мережі, фрейми, логічні моделі та ін.);
- R_5 – бази знань, як елемент прикладної інтелектуальної системи на комп'ютеризованих (електронних) носіях інформації.

Для зберігання даних використовуються **бази даних** (**БД**), для зберігання знань – відповідно **бази знань** (**БЗ**).

База знань – сукупність знань, представлених за допомогою стандартизованих моделей на визначених мовах програмування, які складають основу будь-якої інтелектуальної системи.

Між принципами організації БЗ і БД неможливо провести чіткої межі, оскільки БД певною мірою відображають знання про предметну область. Швидше термін БЗ можна вживати як більш загальний по відношенню до БД і розглядати БД як окремий випадок БЗ. Цим викликано те, що БЗ в системах ШІ доцільно будувати як дворівневу структуру, що включає два компоненти:

- концептуальну – БЗ (верхній рівень);
- інформаційну – БД (нижній рівень).

Проте є специфічні ознаки, що відрізняють знання від даних. В якості таких специфічних ознак знань у зв'язку з поданням їх в ЕОМ можна виділити наступні чотири ознаки (властивості):

- внутрішня інтерпритованість;
- структурованість;
- зв'язність;
- активність.

Якщо звернутися до структурованих даних, то деякі із зазначених ознак, властивих знанням, будуть справедливі і для баз даних.

Спільні ознаки знань і даних

Внутрішня інтерпритованість. Кожна інформаційна одиниця повинна мати унікальне ім'я, за яким ІС знаходить її, а також відповідає на запити, у яких це ім'я згадане. Коли дані, що зберігаються в пам'яті комп'ютера, позбавлені імен, то відсутня можливість їхньої ідентифікації системою. Дані, що добуваються з пам'яті за певною адресою, вказаною програмістом, можуть інтерпритуватися певним чином лише самим програмістом. Що розуміється під тим чи іншим бінарним кодом, системі фактично невідомо.

При переході до знань у пам'ять ЕОМ вводиться інформація про деяку **протоструктуру** інформаційних одиниць. По них можна здійснювати пошук потрібної інформації. Кожен рядок таблиці буде **екземпляром** протоструктури. Реляційні БД забезпечують реалізацію внутрішньої інтерпритованості всіх інформаційних одиниць, що зберігаються в базі даних, де імена стовпців є атрибутами відношень, наприклад:

ПІБ	Рік народження	Стать
Іванов І.І.	1938	Чол.

Петрова К.Д.	1964	Жін.

Структурованість – властивість декомпозиції складних об’єктів на більш прості та встановлення зв’язків між простими об’єктами. Інформаційні одиниці повинні мати гнучку структуру. Для них має виконуватися «принцип мотрійки», тобто рекурсивна вкладеність одних інформаційних одиниць в інші. Кожна інформаційна одиниця може бути включена до складу будь-якої іншої, і з кожної інформаційної одиниці можна виділити окремі її складові інформаційні одиниці. Іншими словами, повинна існувати можливість довільного встановлення між окремими інформаційними одиницями відношення типу «частина-ціле», «рід-вид», «екземпляр-клас» і т.д. Вони реалізуються в ієрархічних і мережевих БД.

Відмінні ознаки знань

Зв'язність: знання пов’язані не тільки в сенсі структури (цьому є аналог і у даних), але й відображають певні закономірності щодо фактів, процесів, явищ і причинно-наслідкових відношень між ними (цьому немає аналога у даних). У інформаційній базі між інформаційними одиницями має бути можливість встановлення зв’язків різного типу. Насамперед ці зв’язки можуть характеризувати відношення між інформаційними одиницями. Семантика відношення може носити декларативний чи процедурний характер. Наприклад, дві чи більш інформаційні одиниці можуть бути зв’язані відношенням «одночасно», дві інформаційні одиниці – відношенням «причина-наслідок» чи відношенням «бути поруч». Наведені відношення характеризують декларативні знання. Якщо між двома інформаційними одиницями встановлено відношення «аргумент-функція», то воно характеризує процедурне знання, зв’язане з обчисленням значень функцій. Далі будемо розрізняти **відношення структуризації, функціональні відношення, каузальні відношення і семантичні відношення**. За допомогою перших задаються ієрархії інформаційних одиниць, другі несуть процедурну інформацію, що дозволяє знаходити (обчислювати) одні інформаційні одиниці через інші, треті задають причинно-наслідкові зв’язки, четверті відповідають всім іншим відношенням. Між інформаційними одиницями можуть встановлюватися й інші зв’язки, наприклад, що визначають порядок вибору інформаційних одиниць з пам’яті, чи вказують на те, що дві інформаційні одиниці несумісні в одному описі.

Активність. Людині властива пізнавальна активність. Наприклад, виявлення суперечностей у знаннях спонукає до подолання цих протиріч і появі нових знань. Таким же стимулом активності є неповнота знань. Знання передбачають цілеспрямоване використання інформації та здатність керувати інформаційними процесами задля вирішення завдань предметної області.

Представлення знань на кожному рівні БЗ здійснюється застосуванням деякої **мови представлення знань** (МПЗ), тобто конкретного способу опису проблемного середовища, що задається синтаксисом опису, правилами співвідношення мовних виразів з відображуваних ними об'єктами предметного середовища і конкретним складом слів мови.

Вибір МПЗ – один з аспектів побудови системи представлення знань.

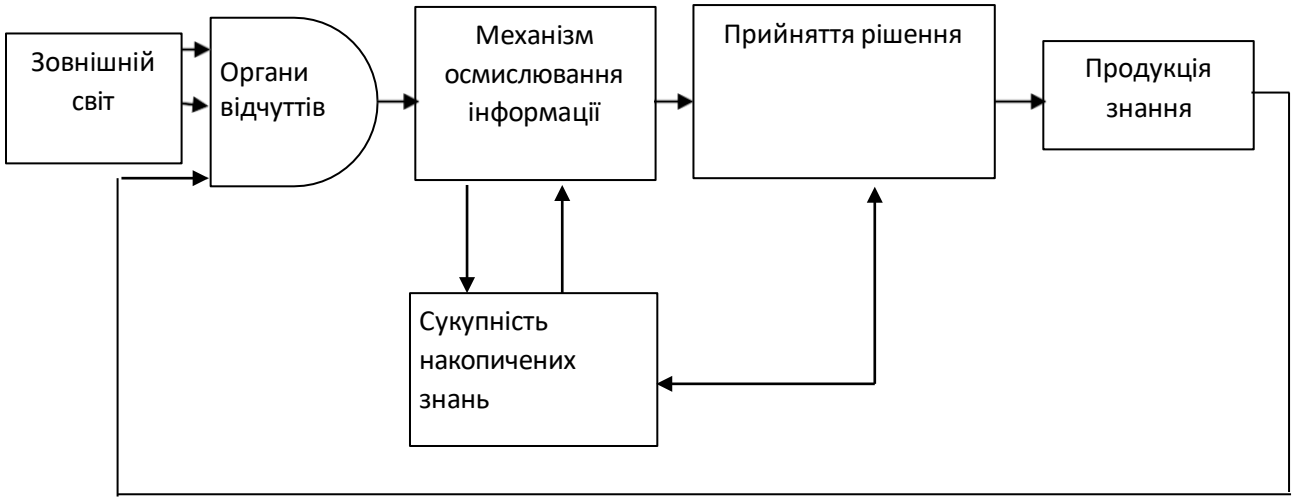
Так, для інформаційного рівня він визначається типом відповідної БД (СУБД).

В якості МПЗ БЗ концептуального рівня можуть бути використані:

- мови програмування високого рівня (імперативні мови Pascal, Delphi, C/C++, C#, тощо). У цьому випадку можливі значні складнощі і трудомісткість реалізації, що може призвести до зниження ефективності;
- мови програмування надвисокого рівня, розраховані на обробку символьної інформації (декларативні мови Lisp, Prolog, Simula та ін.);
- спеціалізовані мови представлення знань з відповідною програмною підтримкою (внесення в БЗ, вилучення, пошук і т.д.). Часто це деяка готова система подання знань з початково порожньою БЗ.

Формалізація поняття «знання»

Накопичення і застосування людиною знань можна формально представити схемою:



Насправді зворотні зв'язки можна було б визначити між усіма блоками.

Таким чином, людина розуміє мову, зображення і використовує знання для вирішення завдань в деякій предметній області. Для виконання аналогічних функцій машиною знання повинні бути якимось чином формалізовані і представлені в ЕОМ. За допомогою традиційних методів процедурного,

структурного чи об’єктно-орієнтованого програмування ці знання поміщалися в програму, тобто представляли собою процедурні знання. Нерідко важко було зрозуміти, яким чином у цьому випадку використовуються знання, і яку роль вони виконують.

Як уже зазначалося, в системах, заснованих на ШІ та інженерії знань, знання представлені в конкретній формі, а наявна БЗ дозволяє їх легко модифікувати, визначати і поповнювати.

Виникнення нового терміну – «інженерія знань» – визначається і пояснюється наступними міркуваннями. Е.Фейгенбаум (фахівець з ШІ із США), який в 1977 році ввів це поняття, зазначав: «… з досвіду відомо, що велика частина знань у конкретній предметній області залишається особистою власністю експерта. І це відбувається не тому, що він не хоче розголошувати своїх секретів, а тому, що він не в змозі зробити цього – адже експерт знає набагато більше, ніж сам усвідомлює». Інакше кажучи, необхідно витягти знання, якими підсвідомо володіє фахівець, і в цьому має допомогти інженер знань.

Так що ж таке знання? Ось одне з визначень:

ЗНАННЯ – результат, отриманий пізнанням навколишнього світу і його об'єктів.

Або більш докладно:

З**НАННЯ** – система суджень з принциповою і єдиною організацією, яка заснована на об’єктивній закономірності.

З погляду ШІ та інженерії знань, поняття знання бажано пов’язати з логічним виведенням:

ЗНАННЯ – це формалізована інформація, на яку посилаються або використовують у процесі логічного виведення.

Проілюструємо це твердження:

Таким чином, якщо логічне виведення на підставі наявних даних робить комп’ютер, то знання в цьому випадку – обов’язково формалізована в певному вигляді інформація.

З точки зору вирішення завдань у деякій предметній області, знання можна розділити на дві категорії – **факти** та **евристики**.

Факти – зазвичай добре відомі в даній предметній області обставини (відомі всім, багатьом, висвітлені в літературі і т.д.).

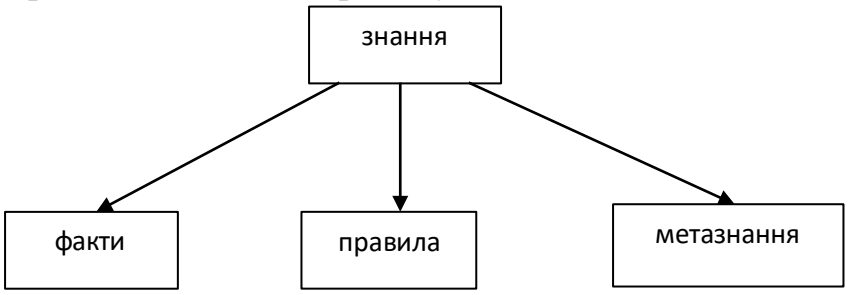
Приклад: *Яблуко має солодкий смак завдяки вмісту в ньому фруктози*.

Евристики – знання, засновані на власному досвіді фахівця в даній предметній області. Сюди входять такі знання, як «способи зосередження»,

«способи видалення непотрібних ідей», «способи використання нечіткої інформації» і т.д.

Приклад: *Постаратися встановити контроль над центром шахової дошки* (стратегія гри в шахи).

Крім того, можна запропонувати й інший спосіб класифікації знань:



Факти – (фактичні знання) – знання типу «А є Б». Вони характерні для БД. Приклад: СВода – мокра», «Земля – планета Сонячної системи».

Правила – (знання для прийняття рішень) – знання виду «ЯКЩО-ТО». Приклад: «ЯКЩО йде дощ, ТО потрібно взяти парасолю».

Метазнання – (знання про знання) – вказують на знання, що стосуються способів використання знань, і знання про властивості знань. Це поняття необхідно для управління БЗ, логічним виведенням, ототожненням, навчанням і т.д.

Дуглас Б. Ленат (США) давав наступне визначення:

Метаправила – це евристики, призначені для утворення нових евристик.

Приклад: «Якщо використання правила дається дорогою ціною, то варто зробити нове, більш конкретне, ефективне правило».

Класифікація знань по виду областей використання може бути представлена наступним чином:

- **понятійні знання** – набір основних понять, які використовуються при вирішенні задачі. Цей тип знань виробляється у фундаментальних науках і теоретичних розділах прикладних наук;
- **конструктивні знання** – знання про структури об’єктів і взаємодії між їх частинами;
- **процедурні знання** – методи, алгоритми, програми, які використовуються в конкретній предметній області;
- **фактографічні знання** – кількісні та якісні характеристики об’єктів і явищ;
- **метазнання** – знання про порядок і правила застосування знань.

Моделі представлення знань

Модель представлення знань повинна відбивати істотні характеристики розв’язуваної задачі й забезпечувати відповідною інформацією процедури, що виконують пошук розв’язань. При цьому вона повинна мати необхідну виразну здатність, щоб відображати всі цікаві деталі предметної області задачі, а також

бути досить ефективною з погляду пошуку розв'язань, що розглядають як вивід на знаннях. Виразна сила й обчислювальна ефективність — дві основні характеристики моделей представлення знань. Багато моделей представлення знань, що володіють більш виразними можливостями, не піддаються ефективній реалізації. Тому пошук оптимального співвідношення між виразною силою використовуваної моделі представлення знань й ефективністю її реалізації — основна задача розробника СШІ.

Існують два типи методів представлення знань (ПЗ):

- **формальні** моделі ПЗ**;**
- **неформальні (семантичні, реляційні)** моделі ПЗ.

Більшість методів представлення знань відносяться до неформальних моделей. На відміну від формальних моделей, в основі яких лежить строга математична теорія, неформальні моделі такої теорії не дотримуються. Кожна неформальна модель годиться тільки для конкретної предметної області і тому не має універсальності, що притаманна формальним моделям. Логічне виведення – основна операція в СШІ – у формальних системах строге і коректне, оскільки підлягає строгим аксіоматичним правилам. Виведення у неформальних системах багато в чому визначається самим дослідником, що і відповідає за його коректність.

Кожному з методів ПЗ відповідає свій спосіб опису знань. До неформальних моделей ПЗ відносять наступні:

- логічна модель;
- продукційна модель (модель, заснована на правилах);
- модель семантичної мережі;
- фреймова модель.

Лекція 4

Представлення знань семантичними мережами. Опис ієрархічної структури і діаграма представлення семантичних мереж. Типові об'єкти. Фундаментальні типи зв'язків. Процедурні семантичні мережі

Семантична мережа (**СМ**) – це один із способів представлення знань. Спочатку семантична мережа була задумана як модель представлення структури довготривалої пам'яті в психології, але згодом стала одним з основних способів представлення знань в інженерії знань.

Перш ніж ознайомитися зі способом подання знань семантичними мережами, розглянемо, що таке «семантична мережа». У тлумачному словнику слово «семантика» визначено так: **Семантика** – значення, сенс слова, художнього твору, дії, обставини і т.д., передані за допомогою будь-яких уявлень і виразів. Ще одне визначення поняття «семантика»: **Семантика** означає певні (загальні) відношення між символами та об'єктами, представленими цими символами. Нарешті, визначення «семантичної мережі»: **Семантичною мережею** називається система знань у вигляді спеціалізованої мережної структури даних, вузли якої відповідають поняттям і об'єктам предметної області, а дуги – іменованим відношенням між об'єктами. Семантична мережа являє собою ієрархічну мережу, у вершинах якої знаходяться інформаційні одиниці. Ці одиниці позначені унікальними іменами (ідентифікаторами). Дуги семантичної мережі відповідають різним зв'язкам між інформаційними одиницями. При цьому ієрархічні зв'язки відповідають відношеннями структуризації, а неієрархічні зв'язки – відношенням інших типів.

Основна ідея подання знань СМ полягає в тому, щоб розглядати предметну область (ПО) як сукупність базових понять ПО і зв'язків між ними. Перевага моделювання за допомогою семантичних моделей полягає в тому, що модель представляє дані про реальні об'єкти і зв'язки між ними прямим способом, що істотно полегшує доступ до знань: починаючи рух від деяких понять, по дузі відношення можна досягти інших понять. Об'єкти при цьому представляються іменованими вершинами, а відношення – напрямленими іменованими ребрами. У загальному вигляді це означає, що за допомогою СМ можна представити знання, викладені в текстах звичайною мовою.

Приклад: Розглянемо фразу «Програміст сів за комп'ютер і відлагодив програму». Тут виділяються три об'єкти: $\mathbf{a_1}$ – *програміст*, $\mathbf{a_2}$ – *комп'ютер*, $\mathbf{a_3}$ – *програма*. Ці об'єкти пов'язані відношеннями: $\mathbf{p_1}$ – *сів за*, $\mathbf{p_2}$ – *відлагодив*, $\mathbf{p_3}$ – *завантажена в*.

До переліку відношень також віднесено відношення «завантажена в» (програма завантажена в комп'ютер), яке відсутнє в тексті. Мережа, що відповідає цьому тексту, показана на рисунку 1.

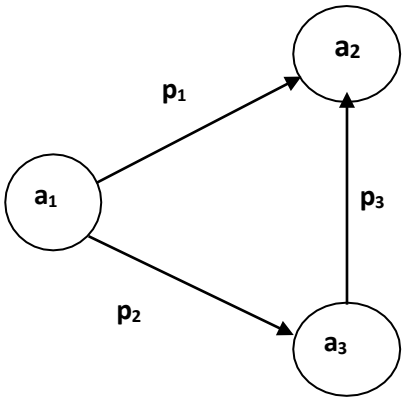


Рис. 1. Приклад семантичної мережі

З прикладу видно, що система знань, що відображаються СМ – орієнтований граф з іменованих вершин і ребер. Ребро і пов'язані ним вершини утворюють підграф СМ, що несе мінімальну інформацію – факт наявності відношення (зв'язку) певного типу між відповідними об'єктами. При використанні СМ для представлення знань можна виділити деякі типові об'єкти і фундаментальні типи відношень між об'єктами. Це властиво так званій **ієрархічній структурі понять** (ІСП).

Розглянемо три типових об'єкта:

Узагальнений об'єкт – це деяке відоме і широко використовуване поняття при представленні знань ПО, наприклад: «персона», «речовина», «машина», «предмет праці» і т.п. Узагальнений об'єкт фактично являє собою клас об'єктів ПО.

Конкретний об'єкт – це певним чином виділений одиничний об'єкт, наприклад: «співробітник Іванов», «сульфат натрію», «екскаватор», «комп'ютер» і тощо. Неважко бачити, що поняття узагальненого і конкретного об'єкта відносні і залежать від конкретного ПО. Так, наприклад, узагальнений об'єкт «машина» може виступати в якості конкретного об'єкта для узагальненого об'єкта «силові установки».

Агрегатний об'єкт – об'єкт ПО, який складений тим чи іншим способом з інших об'єктів, що є його частинами. Поняття агрегатного об'єкта також відносне, адже будь-який об'єкт зовнішнього світу можна розщепити (декомпонувати).

Розглянемо три фундаментальних типів відношень між об'єктами:

- **родовий зв'язок;**
- **«є представником»;**
- **«є частиною».**

Родовий зв'язок. Між двома узагальненими об'єктами може існувати родовий зв'язок «вид–рід» (a kind of – AKF). Це означає наступне: якщо між об'єктами **A** і **B** існує родовий зв'язок, то:

1) будь-який об'єкт, що відображається поняттям **B**, відображається і поняттям **A**;
2) існує такий об'єкт, який відображається поняттям **A**, але не відображається поняттям **B**. Тобто поняття **A** більш загальне, ніж **B**.

Приклад: Знання «Всі ластівки – птахи» може бути представлене СМ:

Інакше це буде: «Будь-яка ластівка – птах, але не всяка птиця – ластівка».

Говорять: **А є родом В**, або **А – родове поняття для В**.

Ще приклади: «тварина» є родом для «людина»; «транспорт» – родове поняття для «літак».

Видовий зв'язок – обернений для родового. Так, якщо **А є родом В**, то **В є видом А**.

Як правило, всі властивості родового поняття (**А**) притаманні видовому (**В**). Зворотне, взагалі кажучи, невірно. Це так зване «правило наслідування властивостей». Успадкування властивостей дозволяє більш компактно представляти систему знань семантичної мережею.

Приклад:

З того, що «всякий ссавець дихає» (властивість роду) випливає, що «людина дихає» (успадкована властивість);

З того, що «людина володіє інтелектом» (властивість виду) не випливає, що інтелектом володіють всі ссавці.

«Є представником». Між узагальненим і конкретним об'єктами може існувати зв'язок «є представником» (instance of) або коротко – «є» (is-a). Він має місце у випадку, якщо конкретний об'єкт належить класу, визначеного узагальненим об'єктом.

Приклад: Знання «Всі ластівки – птахи» може бути представлене СМ:

Для конкретного об'єкта притаманне правило «наслідування властивостей» узагальненого об'єкта (але не навпаки). Усі властивості, притаманні об'єктам «ластівка» і «птах» мають місце і для конкретного об'єкта – ластівки на ім'я Юко.

Неважко бачити, що зв'язки «родовий» і «є представником» дуже схожі і можуть переходити один в інший залежно від ПО і від класу вирішуваних завдань. Більше того, у ряді випадків між зв'язками AKF та IS-A не роблять різниці, відзначаючи, що ці зв'язки утворюють відношення «загальне– часткове».

«Є частиною». Між агрегатним і яким-небудь іншим об'єктом ПО може існувати зв'язок «є частиною» або коротко – «частина» (a part of), або інакше – «мати в якості частини» (has-part). Це відношення дозволяє представляти знання, що стосуються атрибутів об'єкта.

Приклад: Знання «Птахи мають крила» відображається такою СМ:

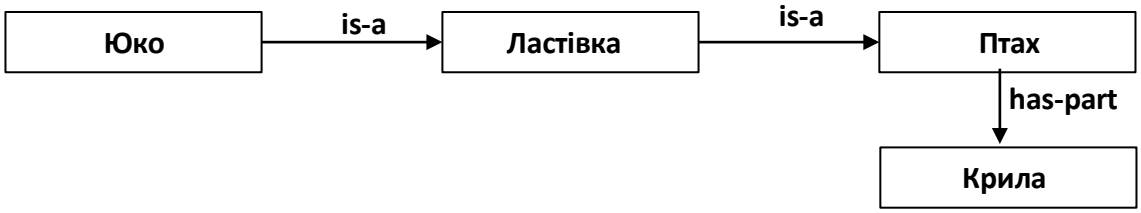
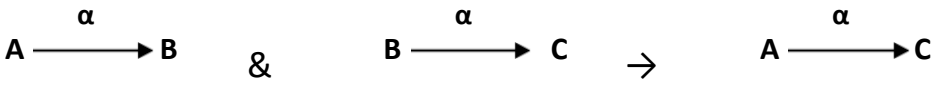


Рис. 2

Зауваження 1. Наведена типізація не вирішує всіх проблем представлення знань, так як реально існують об'єкти і зв'язки інших типів. Але така типізація дозволяє будувати основу (скелет) БЗ і вподальшому «стиснути» БЗ, зробивши її більш компактною. Дійсно, для кожного об'єкта є сенс вказувати в БЗ властивості тільки йому притаманні. Інші можуть бути виведені завдяки механізму успадкування. Механізм успадкування можна формалізувати, якщо ввести поняття **транзитивності відношення**.

Означення. Відношення **α** називається **транзитивним**, якщо для будь-якої трійки об'єктів **A, B, C** таких, що **A** знаходиться у відношенні **α** з **B**, та **B** знаходиться у відношенні **α** з **C**, одночасно **A** знаходиться у відношенні **α** з **C**:



Всі розглянуті нами фундаментальні відношення мають властивість транзитивності.

Зауваження 2. Транзитивність фундаментальних відношень ієрархічної структури понять (ІСП) дозволяє використовувати виведення для отримання багатьох нових знань (фактів). Таким чином, можна зробити висновок, що конкретна ластівка по імені Юко, що є представником роду птахів, успадковує всі властивості птахів, а отже у неї є крила. Як приклад відношення, що не володіє властивістю транзитивності, можна навести відношення «батько». На противагу йому відношення «предок» – транзитивне.

Зауваження 3. Операції модифікації бази знань на семантичних мережах зводяться до видалення існуючих та додавання нових вершин і ребер, встановлення нових зв'язків між вершинами із використанням механізму наслідування властивостей тощо.

Проілюструємо модифікацію СМ на наступному прикладі. Додамо в скелет БЗ новий факт «Юко володіє гніздом». Отримаємо нову СМ:

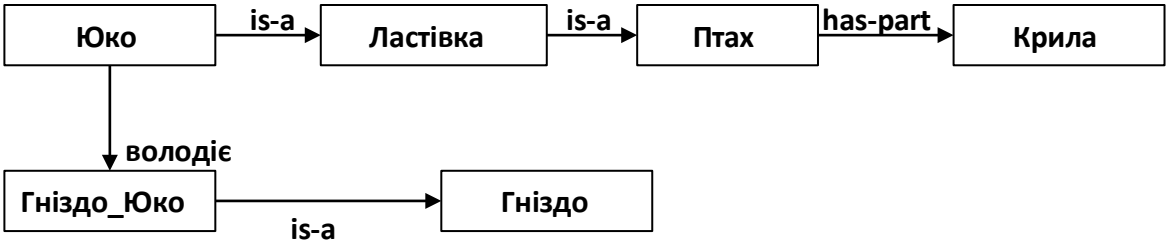


Рис. 3

Тут «Гніздо_Юко» – конкретне гніздо, яким володіє Юко. Воно є конкретним об'єктом узагальненого об'єкта «Гніздо».

Це буде пов'язано з появою вершин нових типів: вершини- ситуації і вершини-дії.

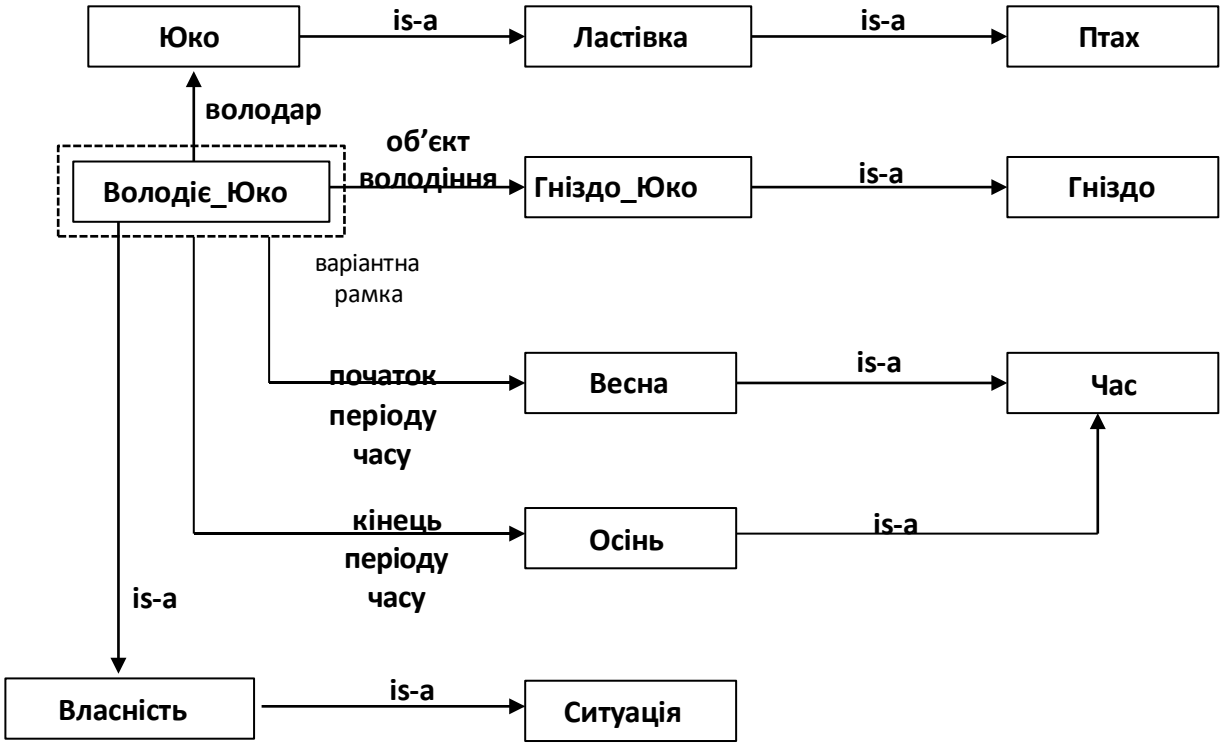


Рис. 4

Тут для вершини-ситуації «Володіти» («Володіє_Юко») визначено декілька зв'язків. Така вершина називається **варіантною рамкою** (case frame).

Процедурну семантичну мережу (ПСМ) можна представити четвіркою фундаментальних компонент:

$ПСМ = <A_i, P_i, F_i, Q_i>$,

де індекс i відповідає конкретній ПСМ. Тут:

A_i – множина базових понять (об'єктів) ПО, $A_i = \{a_1, a_2, ..., a_k\}$;

P_i – множина відношень між поняттями (об'єктами) з A_i, $P_i = \{p_1, p_2, ..., p_l\}$;

F_i – множина процедур перетворення над СМ, $F_i = \{f_1, f_2, ..., f_m\}$. У цю множину входять операції модифікації БЗ на СМ та базові операції пошуку (виводу) інформації;

Q_i – множина правил наслідування властивостей, $Q_i = \{q_1, q_2, ..., q_n\}$.

Наприклад:

a_1 – «ластівка», a_2 – «крила» і т.д.;

p_1 – родовий зв'язок; p_2 – має частину; p_3 – є представником; p_4 – володіє;

f_1, f_2 – додати/видалити вершину;

f_3, f_4 – додати/видалити ребро;

f_5 – підрахунок числа вершин, з'єднаних даним ребром (зв'язаних відношенням заданого типу);

f_6 – пошук ребра (вершини) по імені;

f_7 – перевірка наявності/відсутності зв'язку між заданими вершинами;

f_8 – перехід від однієї вершини до іншої по зв'язках;

f_9 – перехід від одного зв'язку до іншого через суміжні вершини тощо;

Потреба в ПСМ у множині Q_i обумовлена, зокрема, наступними міркуваннями. Розглянемо приклад СМ:

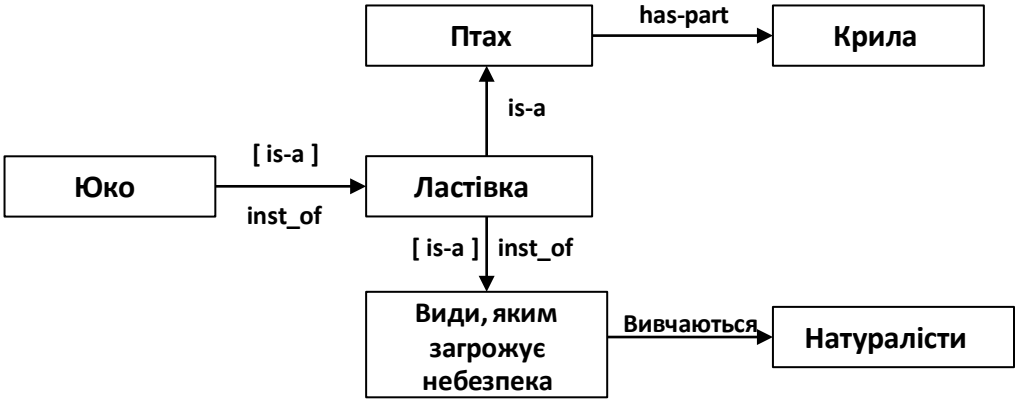


Рис. 5

Завдяки транзитивності відношень ми бачимо, що «Юко» успадковує властивості об'єкта «птах», в саме можемо вивести факт «Юко має крила». З іншого боку, проводячи аналогічні міркування, «Юко» успадковує властивості «Видів, яким загрожує небезпека» і це приводить до факту «Юко вивчається натуралістами».

З точки зору формальної логіки ми не допустили помилки, але прийшли до факта (твердження), достовірність якого не гарантується.

Замінимо зв'язок «є» (is_a) між об'єктами «Юко», «Ластівка» і «Види, яким загрожує небезпека» на інший зв'язок – «є представником» (inst_of). Раніше ми вважали їх одним видом зв'язку. Тепер збережемо семантику самих зв'язків, але змінимо для них правила успадкування властивостей:

q_1 – якщо об'єкт **A** знаходиться у відношенні «is_a» з об'єктом **B**, то **A** успадковує всі властивості **B**;

q_2 – якщо об'єкт **A** знаходиться у відношенні «inst_of» з об'єктом **B**, то **A** не успадкує властивості **B** (або успадковує властивості тільки вибірково);

q_3 – транзитивність властивостей «is_a» і «inst_of».

Неважко бачити, що це дозволить розв'язати проблемну ситуацію, описану раніше.

Зауваження 4. Часто правила Q_i не відокремлюються від процедур F_i, а закладаються в самі процедури. Так, якщо ми побудуємо деяку процедуру f^* переходу від однієї вершини до іншої на основі правила q_1, то це буде процедурним поданням знання про правило q_1. Отже, в ПСМ є можливість подання процедурних знань.

Переваги і недоліки семантичних мереж

Переваги СМ:

- природність представлення знань;
- можливість об'єднання різних фрагментів мережі;
- можливість виділення ділянок мережі, що охоплюють смислові

характеристики необхідні в даному запиті.

Недоліки СМ:

- складність розробки мережі для обширних предметних областей;
- складність технічної реалізації виведення;
- значний час пошуку по мережі.

ТЕМА 5

Фреймова модель представлення знань. Поняття фрейму. Структура даних фрейму

Фреймова модель представлення знань запропонована американським ученим Марвіном Мінським в 1975 р – форма представлення знань, заснована на об'єктно-орієнтованому підході. Знання представляються у вигляді об'єктів з їх властивостями.

Фрейм (англ. *frame* – каркас, рамка) – структура даних для опису концептуального об'єкта (поняття, явища, ситуації тощо).

Специфікація фрейму включає:

- структуру фрейму;
- опис зв'язків з іншими фреймами.

Графічно структуру фрейму можна представити у вигляді дерева (рис. 1), або у вигляді таблиці (рис. 2). Інформація, що відноситься до фрейму, міститься в слотах.

Слот (англ. *slot* – місце для вставки даних) – це одиниця інформації фрейму, елемент даних для фіксації знань про властивість об'єкта. Специфікація слота включає:

- ім'я слота;
- покажчик типу атрибута слота;
- покажчик успадкування;
- значення слота «за замовчуванням»;
- приєднані процедури;
- іншу інформацію, що відноситься до даної властивості об'єкта.

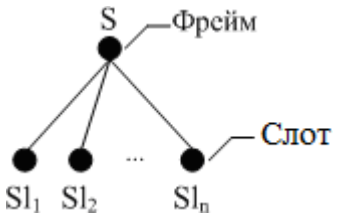


Рис. 1. Графічне представлення фрейму у вигляді дерева

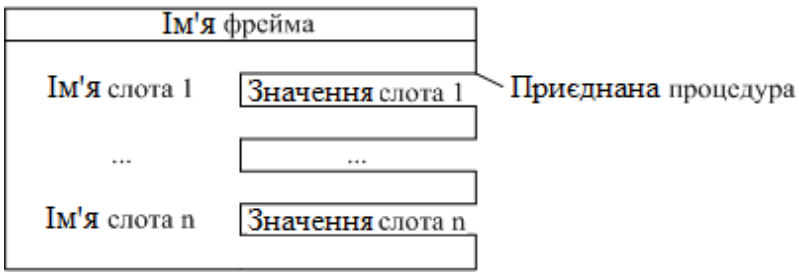


Рис. 2. Графічне представлення фрейму у вигляді таблиці

якого вже заповнені, є **фреймом-екземпляром**. У експертній системі, заснованої на фреймах, база знань представляється сукупністю фреймів-описів, а робоча пам'ять – сукупністю фреймів-екземплярів.

Фрейми, як правило, взаємопов'язані і утворюють єдину фреймову систему. **Фреймова система** – структура, вузлами якої є фрейми. Найчастіше фреймові системи мають ієрархічну структуру, яка характеризує ступінь абстракції поняття. В ієрархічній структурі можуть використовуватися, наприклад, такі відношення, як: **клас–підклас** (is kind of); **абстрактне– конкретне** (is a); **ціле–частина** (is part of). Особливістю фреймової системи є можливість наслідування значень слотів і використання значень за замовчуванням.

Розглянемо опис структури даних фрейму на прикладі фреймової моделі для класу об'єктів «Аудиторія» (Рис. 3).

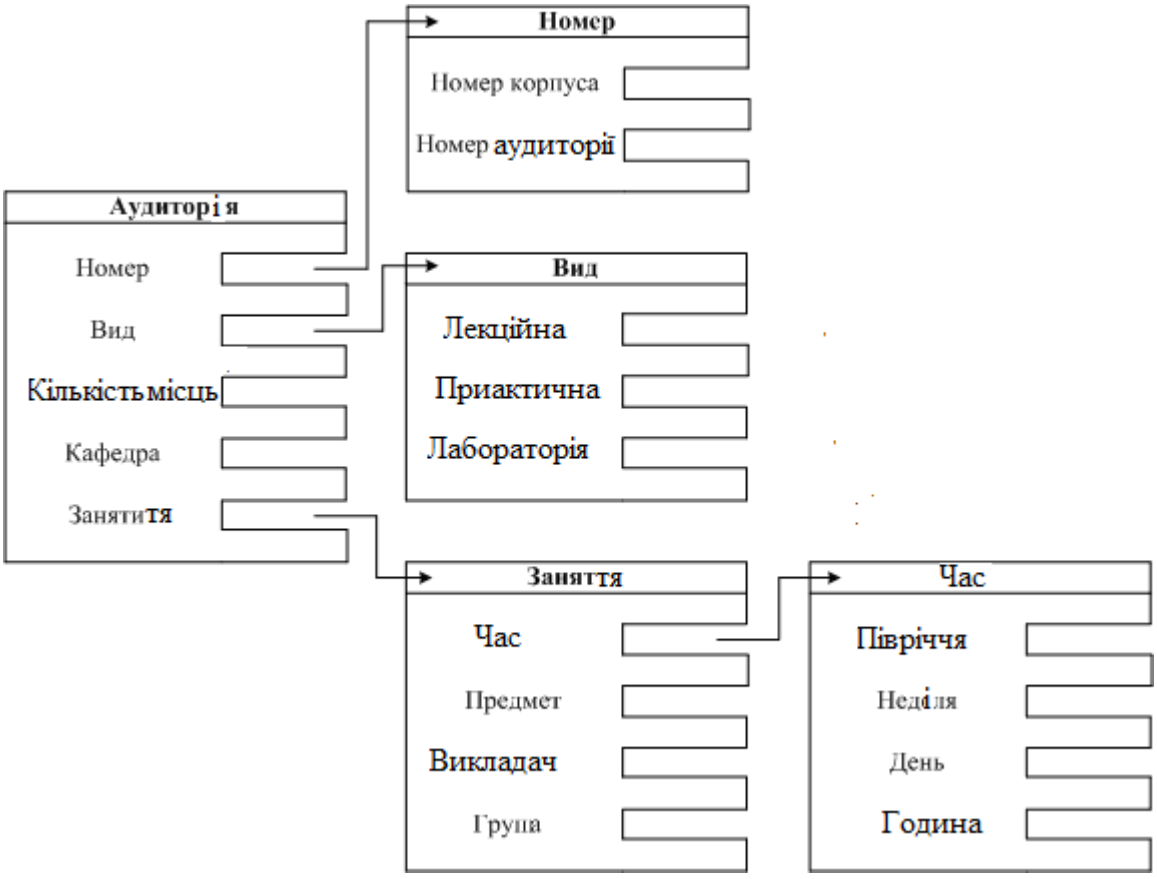


Рис. 3. Приклад фреймової моделі для класу об'єктів «Аудиторія»

Ім'я фрейму. Фрейм повинен мати ім'я, унікальне в даній фреймової системі. У розглядуваному прикладі іменами фреймів можуть бути: ***Аудиторія, Номер, Вид_аудиторії, Заняття, Час*** та ін.

Кожен фрейм складається з деякого числа слотів, причому значення одних слотів задаються користувачем, значення інших слотів визначаються системою в процесі логічного виведення.

Ім'я слота. Слот повинен мати унікальне ім'я у фреймі, до якого він належить. У розглядуваному прикладі іменами слотів можуть бути: ***Кількість_місць, Кафедра, Номер_корпусу, Викладач, Півріччя*** та ін.

Тип даних. Вказується, що слот має деяке значення або служить покажчиком іншого фрейму. Можуть використовуватися наступні типи даних: ***frame*** (покажчик); ***integer*** (цілий); ***real*** (дійсний); ***bool*** (булеві); ***text*** (текст); ***list*** (значення зі списку); ***table*** (таблиця); ***expression*** (вираз) тощо.

У розглядуваному прикладі тип ***frame*** мають слоти ***Номер, Заняття, Час***; тип ***list*** (значення зі списку) мають слоти ***Півріччя, Тиждень***; тип ***integer*** мають слоти ***Номер_аудиторії, Номер_корпусу, Кількість_місць***; тип ***text*** мають слоти ***Кафедра, Предмет, Викладач, Група*** та ін.

Значення слота. Значення слота має збігатися з вказаним типом даних цього слоту і відповідати умові успадкування.

Приєднана процедура. З кожним слотом може бути пов'язана одна або декілька процедур, виконання яких залежить від значення слота. Це приєднані процедури. Вони забезпечують процес логічного виведення.

Для стандартизації обробки приєднаних процедур використовуються **демони** (*demon*) – стандартні процедури, які автоматично запускаються при зверненні до відповідного слоту і виконанні деякої умови. До таких процедур відносяться:

- ***if_needed*** (якщо потрібно): процедура виконується, коли запитується інформація зі слота, але він порожній;
- ***if_added*** (якщо додано): процедура виконується, коли нова інформація поміщається в порожній слот;
- ***if_changed*** (якщо змінено): процедура виконується при зміні значення слота;
- ***if_removed*** (якщо видалено): процедура виконується при видаленні інформації зі слота.

Для розглянутого прикладу можна задати наступні процедури:

- ***if_needed***: якщо для слота ***Номер*** у фреймі ***Аудиторія*** значення не вказано, то виводиться повідомлення, що його потрібно вказати;
- ***if_added***: при занесенні значення в слот ***Номер_корпусу*** перевіряється, чи існує такий корпус, якщо немає виводиться повідомлення про помилку;
- ***if_removed***: підтвердження при видаленні значення.

Переваги і недоліки фреймового підходу.

переваги:

- здатність відображати концептуальну основу організації пам'яті людини;
- інтеграція декларативних і процедурних знань;
- простота і природність представлення знань;
- однорідність і хороша структурованість знань.

недоліки:

- складність налагодження системи;
- складність управління виведенням.

ТЕМА 6

Представлення знань правилами. Продукційні системи. Експертні системи. Структура експертної системи. Механізм логічного виведення. Стратегії управління виведенням. Алгоритм логічного виведення. Прямий ланцюжок логічного виведення. Обернений ланцюжок логічного виведення. Конфлікти в експертних системах

Програмні засоби, що базуються на технології та методах штучного інтелекту, набули значного поширення у світі. Їх важливість, у першу чергу експертних систем і нейронних мереж, полягає в тому, що дані технології істотно розширюють коло практично значущих завдань, які можна вирішувати на комп'ютерах, і їх рішення приносить значний економічний ефект. У той же час технологія експертних систем є найважливішим засобом у вирішенні глобальних проблем традиційного програмування: тривалість, а отже, висока вартість розробки додатків; висока вартість супроводу складних систем; повторна використовуваність програм і т.п. Крім того, об'єднання технологій експертних систем і нейронних мереж з технологією традиційного програмування додає нові якості до комерційних продуктів за рахунок забезпечення динамічної модифікації додатків користувачем, а не програмістом, більшої „прозорості" додатків (наприклад, знання зберігаються на звичайній мові, що не вимагає коментарів до них, спрощує навчання та супровід), кращих графічних засобів, користувальницького інтерфейсу і взаємодії.

На думку фахівців, в недалекій перспективі **експертні системи** (**ЕС**) будуть відігравати провідну роль у всіх фазах проектування, розробки, виробництва, розподілу, продажу, підтримки та надання послуг. Їх технологія, отримавши комерційне поширення, забезпечить революційний прорив при розробці різного роду додатків шляхом інтеграції із готових інтелектуально-взаємодіючих модулів.

Мета досліджень по ЕС полягає в розробці програм, які при вирішенні завдань, важких для експерта-людини, отримують результати, які не поступаються за якістю і ефективності рішень, одержуваних експертом. Дослідники в області ЕС для назви своєї дисципліни часто використовують також термін „інженерія знань2, введений Е. Фейгенбаумом як „використання принципів та інструментарію досліджень в області штучного інтелекту у вирішення важких прикладних проблем, що вимагають знань експертів". Експертні системи та системи штучного інтелекту відрізняються від систем обробки даних тим, що в них в основному використовуються символьний (а не числовий) спосіб представлення, символьний висновок і евристичний пошук рішення (а не виконання відомого алгоритму).

Експертні системи застосовуються для вирішення практичних завдань, які не піддаються простій формалізації. За якістю і ефективності вирішення

експертні системи не поступаються рішенням експерта-людини. Рішення експертних систем володіють „прозорістю”, тобто можуть бути пояснені користувачу на якісному рівні. Ця якість експертних систем забезпечується їх здатністю міркувати про своїх знаннях і умовиводах. Експертні системи здатні поповнювати свої знання в ході взаємодії з експертом. Необхідно відзначити, що в даний час технологія експертних систем використовується для вирішення різних типів завдань (інтерпретація, пророкування, діагностика, планування, конструювання, контроль, налагодження, інструктаж, управління) в найрізноманітніших проблемних областях, таких як фінанси, нафтова і газова промисловість, енергетика, транспорт, фармацевтичне виробництво, космос, металургія, гірнича справа, хімія, освіта, целюлозно-паперова промисловість, телекомунікації і зв'язок та ін.

ЕС, що засновані на правилах, називаються **продукційними системами**. Правила формулюють певні дії при виконанні деяких заданих умов. У найпростішому вигляді правила продукції близькі за змістом імплікації «ЯКЩО-ТО», тому для правил можна прийняти позначення

$\mathbf{P_i: A \rightarrow B.}$

Або розкривши ліву частину на складові умови, отримаємо:

$\mathbf{P_i: A_1 \wedge A_2 \wedge A_3 \ldots \wedge A_n \rightarrow B,}$

де **Ai** – умови застосовності висновку, що утворюють кон’юнкцію;

B – висновок або дія, яка має місце при істинності кон’юнкції умов.

Приклад: „ЯКЩО внутрішнє тестування пройшло і має місце багаторазова перезавантаження ОС, ТО залипання клавіш або збій ОЗУ”.

Експертні системи – одне з важливих напрямків діяльності людини у галузі штучного інтелекту. Можна дати кілька визначень, які відображають основні аспекти експертних систем.

Експертна система – це програма, яка поводиться подібно експерту в деякій прикладної області.

Експертна система – це система, що об'єднує можливості комп'ютера зі знаннями й досвідом експерта в такій формі, що система може запропонує розумну пораду або здійснити розумне рішення поставленого завдання.

Експертна система – це інтелектуальна програма, здатна робити логічні висновки на підставі знань у конкретній предметній області і забезпечує рішення специфічних завдань.

Класифікація ЕС здійснюється за різними ознаками, в залежності від області застосування, проблемою, що вирішується тощо.

За метою створення:

- для навчання фахівців;
- для вирішення задач;
- для автоматизації рутинних робіт;
- для тиражування знань експертів.

За ступенем складності структури:

- **Поверхневі системи** — подають знання про область експертизи у вигляді правил (*умова->дія*). Умова кожного правила визначає зразок деякої ситуації, при дотриманні якої правило може бути виконано. Пошук рішення полягає у виконанні тих правил, зразки яких зіставляються з поточними даними. При цьому передбачається, що в процесі пошуку рішення послідовність формованих у такий спосіб ситуацій не обірветься до одержання рішення, тобто не виникне невідомої ситуації, що не зіставиться з жодним правилом;
- **Глибинні системи** — крім можливостей поверхневих систем, мають здатність при виникненні невідомої ситуації визначати за допомогою деяких загальних принципів, справедливих для області експертизи, які дії варто виконати.

За зв'язком з реальним часом:

- **Статичні** ЕС розробляються в предметних областях, у яких база знань та інтерпретовані дані не змінюються в часі. Вони стабільні. (Діагностика несправностей в автомобілі.);
- **Квазідинамічні** ЕС інтерпретують ситуацію, що змінюється з деяким фіксованим інтервалом часу. (Мікробіологічні експертні системи, в яких знімаються лабораторні вимірювання з технологічного процесу один раз на декілька годин та аналізується динаміка одержаних показників по відношенню до попереднього виміру.);
- **Динамічні** ЕС працюють у поєднанні з датчиками об'єктів у режимі реального часу з постійною інтерпретацією даних, що надходять. (Управління гнучкими виробничими комплексами, моніторинг у реанімаційних палатах і т.д.).

За ступенем інтеграції з іншими програмами:

- **Автономні експертні системи** працюють безпосередньо в режимі консультацій з користувачем для специфічних „експертних" завдань, для вирішення яких не потрібно привертати традиційні методи обробки даних (розрахунки, моделювання і т.д.);
- **Гібридні експертні системи** представляють програмний комплекс, агрегують стандартні пакети прикладних програм (наприклад, математичну статистику, лінійне програмування або системи управління базами даних) та засоби маніпулювання знаннями. Це може бути інтелектуальна надбудова над ППП або інтегроване середовище для вирішення складного завдання з елементами експертних знань.

При розробці ЕС прийнято виділяти три основні модулі (Рис. 1):

- **база знань**;
- **машина логічного виведення**;
- **інтерфейс користувача**.

Формалізуємо ці поняття.

БАЗА ЗНАНЬ (БЗ) – виділені і певним чином структуровані знання про предметну область.

МАШИНА (МЕХАНІЗМ) ЛОГІЧНОГО ВИВЕДЕННЯ – загальні знання про технології, методи, способи, засоби вирішення завдань.

Механізм виведення, як правило, містить:

- **інтерпретатор**, що визначає, яким чином застосовувати правила для виведення нових знань;
- **диспетчер**, що встановлює порядок застосування правил.

ІНТЕРФЕЙС КОРИСТУВАЧА здійснює обмін інформацією між користувачем та системою і дає користувачеві можливість спостерігати за процесом рішення задачі.

Машину виведення та інтерфейс зазвичай називають **оболонкою експертної системи**.

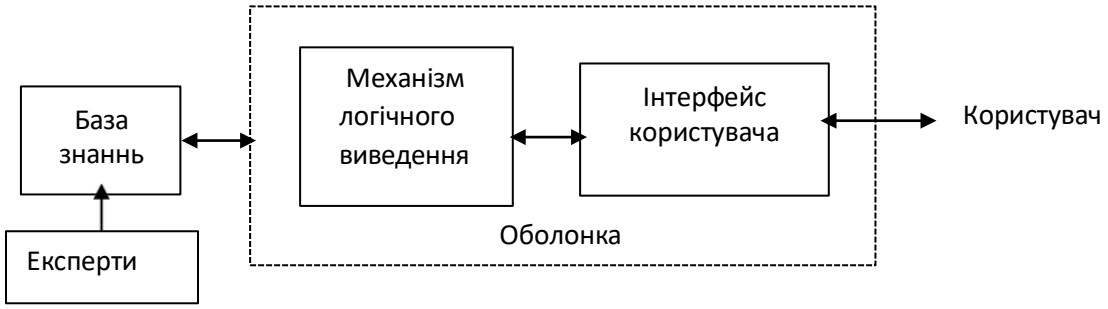


Рис. 1. Структура експертної системи

Експертні системи можна розглядати з інших позицій. З погляду інформаційної структури (інформаційних потоків), експертна система, як правило, складається з трьох основних компонентів (Рис. 2):

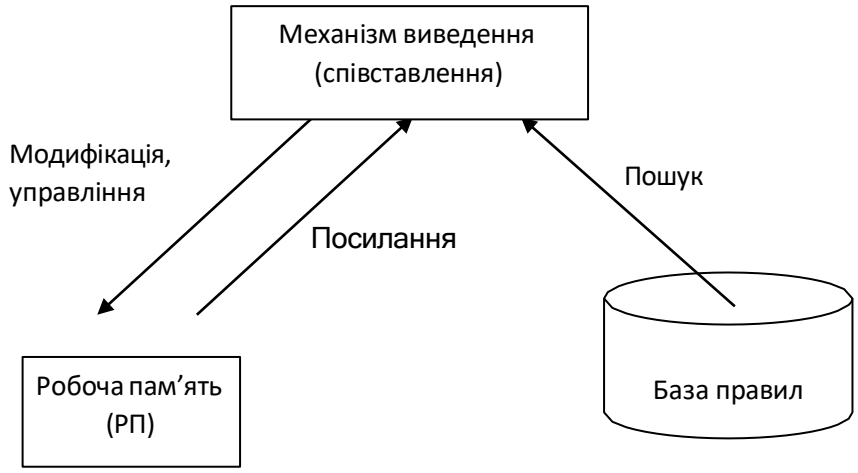


Рис. 2. Інформаційні потоки в експертній системі

База правил – це БЗ системи, що містить множину правил продукційного типу.

Робоча пам'ять (РП) (пам’ть для короткочасного зберігання) – пам’ять, що містить факти, що є передумовою вирішення завдання, і результати виведення, отриманих на їх основі.

Механізм виведення – процедура, що реалізує певний спосіб застосування правил та порядок їх використання для забезпечення логічного виведення.

База знань складається зі скінченного набору правил

$$\Pi = \{P_1, \dots, P_m\}$$

та скінченного набору фактів

$$A = \{a_1, \dots, a_n\},$$

якими може оперувати система.

Умова застосовності будь-якого з правил $P_i \in \Pi$

$$P_i : a_{i1} \wedge a_{i2} \wedge \dots \wedge a_{is} \rightarrow a_m, \quad (1)$$

полягає в одночасній наявності фактів $a_{i1}, a_{i2}, \cdots, a_{is}$ в РП. Де

$\wedge$ – кон'юнкція («І»);

a_m – новий факт, виведений з фактів-умов $a_{i1}, \dots, a_{is}$.

При цьому $a_m \in A$ і a_m є новим фактом у даній предметній області, що розширює робочу пам'ять системи при застосуванні правила P_i.

В (1) використовується $\wedge$ (кон'юнкція), але не використовується $\vee$ (диз'юнкція). Дійсно, якщо розглянути правило

$$P_j : a_{j1} \vee a_{j2} \rightarrow a_k ,$$

то воно може бути зведене до використання двох різних правил:

$$P_{j1} : a_{j1} \rightarrow a_k, P_{j2}$$

$$: a_{j2} \rightarrow a_k,$$

що не є ефективним, і ми надалі це будемо мати на увазі.

Оскільки продукційні ЕС в основному є людино-машинними системами, то вихідні дані для висновку, тобто факти, які мають місце в даній конкретній ситуації, надходять в ЕС від користувача (можливо інтелектуальних сенсорів). Щоб отримати ці факти, система посилає користувачеві запит (наприклад, у вигляді альтернативного меню)

$$q_i = \{a_{j1} \vee a_{j2} \vee \dots \vee a_{jk}\}.$$

Після вибору користувачем будь-якого факту, можуть бути застосовані продукції. Які саме факти запитувати у користувача, можна визначити при розробці системи або реалізувати інтелектуальний інтерфейс з користувачем, заснований на використанні **метазнань**.

Таким чином, продукційну ЕС можна представити трійкою об'єктів:

$$<A, \Pi, Q>,$$

де $Q=\{q_j\}$ – множина запитів до користувача.

Машина виведення (**інтерпретатор правил**) виконує дві функції:

- по-перше, перегляд існуючих фактів з робочої пам'яті і правил з бази знань і додавання (в міру можливості) в робочу пам'ять нових фактів;
- по-друге, визначення порядку перегляду і застосування правил.

Цей механізм управляє процесом консультацій, зберігаючи інформацію про отримані відповіді від користувача, і запитує в нього інформацію, коли для виконання чергового правила в робочій пам'яті виявляється недостатньо даних.

Механізм (машина) виведення являє собою програму, що об'єднує два

компоненти: один реалізує саме виведення, інший керує цим процесом.

Компонент виведення базується на використанні правил. Правила

якщо істинна передумова, то має бути істинним висновок. Компонент виведення повинен функціонувати навіть при нестачі інформації. Отримане рішення може бути неточним або неповним, проте система не повинна зупинятися через те, що відсутня якась частина вхідної інформації.

Керуючий компонент визначає порядок застосування правил і виконує чотири функції:

- **зіставлення** – зразок правила зіставляється з наявними фактами;
- **вибір** – якщо в конкретній ситуації може бути застосовано відразу декілька правил, то з них вибирається одне, найбільш оптимальне за обраним критерієм (вирішення конфлікту);
- **застосування** – якщо зразок правила при зіставленні збігся з фактами із робочої пам'яті, то правило застосовується;
- **дія** – робоча пам'ять піддається зміні шляхом додавання в неї результату застосування правила. Якщо в правій частині правила містяться вказівки на якісь дії, то вони виконуються.

Інтерпретатор продукцій працює циклічно. Проглядаються всі правила, щоб виявити ті, передумови яких збігаються з відомими на даний момент фактами з робочої пам'яті. Після вибору правило застосовується, його висновок заноситься в робочу пам'ять, і цикл повторюється спочатку. За одну ітерацію циклу може застосуватися тільки одне правило. Якщо кілька правил успішно зіставлені з фактами, то інтерпретатор робить вибір одного із них за певним критерієм оптимальності.

Інформація з РП послідовно зіставляється з передумовами правил для виявлення успішного зіставлення. Сукупність відібраних правил складає так звану **конфліктну множину**. Для вирішення конфлікту інтерпретатор має критерій, за допомогою якого він вибирає єдине правило, після чого воно застосовується. Це передбачає занесення в РП фактів, що утворюють висновок правила, або зміну критерію вибору конфліктуючих правил. Якщо ж висновок правила передбачає якусь дію, то вона виконується.

Робота машини виведення залежить від стану РП і від складу БЗ. На практиці часто враховується історія роботи, тобто поведінка механізму виведення. Інформація про поведінку механізму виведення запам'ятовується в пам'яті станів. Пам'ять станів містить протокол системи.

Від обраного методу пошуку, тобто стратегії виведення, залежить порядок використання та застосування правил. Процедура вибору зводиться до визначення напряму пошуку і способу його здійснення.

При розробці стратегії управління виведенням важливо визначитись з двома питаннями:

1) яку точку в просторі станів прийняти в якості вихідної? Від вибору цієї точки залежить і метод здійснення пошуку – в прямому чи зворотному напрямі;
2) якими методами можна підвищити ефективність пошуку рішення? Ці методи визначаються вибраною стратегією перебору: в глибину, в ширину, по підзадачах тощо.

У експертних системах, заснованих на правилах, є два основні способи побудови ланцюжка виведення:

- прямий ланцюжок міркувань;
- зворотний ланцюжок міркувань.

У системах зі зворотним порядком виведення спочатку висувається гіпотеза, а потім механізм виведення ніби повертається назад, переходячи до фактів, намагаючись знайти ті, які підтверджують гіпотезу. Якщо вона виявилася правильною, то вибирається наступна гіпотеза, яка деталізує першу і є по відношенню до неї підціллю. Далі відшукуються факти, що підтверджують істинність підпорядкованої гіпотези.

У системах з прямим виведенням на основі відомих фактів відшукується висновок, який слідує з цих фактів. Якщо такий висновок вдається знайти, то він заноситься в РП. І т.д., поки не буде отриманий потрібний результат.

Існують системи, в яких виведення базується на поєднанні обох методів.

Приклад.

База знань:

Факти:

$\mathbf{a_1}$ – "відпочинок влітку";

$\mathbf{a_2}$ – "людина активна";

$\mathbf{a_3}$ – "їхати в гори";

$\mathbf{a_4}$ – "любить сонце".

Правила:

$\mathbf{P_1 – a_1 \wedge a_2 \rightarrow a_3;}$

$\mathbf{P_2 – a_4 \rightarrow a_1.}$

Нехай в систему надійшли факти: "людина активна" і "любить сонце", тобто $\mathbf{РП = \{a_2, a_4\}}$. Мета – підтвердити (встановити) істинність факту $\mathbf{a_3}$.

Прямий виведення: на основі фактичних даних, отримати рекомендацію.

крок 1. Пробуємо $\mathbf{P_1}$ – не підходить;

крок 2. Пробуємо $\mathbf{P_2}$, якщо підходить, то в РП надходить факт $\mathbf{a_1}$:

$\mathbf{РП = \{a_2, a_4, a_1\}}$;

крок 3. Пробуємо $\mathbf{P_1}$ – підходить, отже активізується мета $\mathbf{a_3}$, що є рекомендацією, яку дає ЕС.

Зворотне виведення: підтвердити обрану ціль за допомогою наявних фактів і правил.

Припустимо, що задана мета "їхати в гори" при наявних фактах $\mathbf{a_2}$ і $\mathbf{a_4}$.

1-ий прохід:

- крок 1. Мета $\mathbf{a_3}$. Пробуємо правило $\mathbf{P_1}$ – факт $\mathbf{a_1}$, як передумова, відсутній, тому цей факт стає новою метою, і шукається правило, де мета $\mathbf{a_1}$ знаходиться в правій частині;

- крок 2. Мета $\mathbf{a_1}$. Пробуємо $\mathbf{P_2}$ – підтверджує мету і активує її. Факт $\mathbf{a_1}$ потрапляє в РП.

  2-ий прохід:
- крок 3. Знову пробуємо $\mathbf{P_1}$ – шукана мета підтверджується.

У результаті таких кроків породжується ланцюжок логічного виведення, що доводить поставлену мету.

Конфлікти в експертних системах. При логічному виведенні може виникати так званий „конфліктний набір правил”.

Конфліктний набір (КН) – це множина правил логічного виведення, які на деякому кроці виведення можуть бути застосовані одночасно.

Вирішення конфлікту – процедура вибору одного правила з конфліктного набору.

Виникнення конфліктів у реальних експертних системах – явище досить звичайне. Тому ефективність системи багато в чому залежить від способу вирішення конфліктів.

Яким же чином управляти виведенням у продукційних системах? Для цього можна запропонувати такі підходи:

- **встановлення обмежень** на генерацію конфліктного набору;
- використання того чи іншого **алгоритму вирішення конфліктів**.

Способи встановлення обмежень на генерації конфліктних наборів:

1) використовуються **метаправила**, в умовній частині яких містяться умови, за якими деяка категорія правил не розглядається. Ці обмеження можуть бути накладені на атрибути, що визначають пошук, на кількість умов умовної частини і т.д. Приклад метаправил для застосування продукцій:

МР1 : ЯКЩО правило містить атрибут "колір об'єкта", ТО правило не включати в конфліктний набір;

МР2 : ЯКЩО число умов правила більше K, ТО включити правило в конфліктний набір;

2) правила групуються за атрибутам. Для кожної групи правил вказується умова (метаправило), в якому випадку застосовувати правила цієї групи. Ця умова стосується вмісту робочої пам’яті. Якщо виконується умова метаправила, то в конфліктний набір включаються тільки правила з групи, що визначається цим метаправилом.

Алгоритми вирішення конфліктів:

1) правила застосовуються по порядку їх слідування в РП;

2) першим застосовується правило із строгішою умовною частиною (наприклад, що має багато умов в правій частині). Вважається, що в цьому випадку результат виведення буде більш якіснішим;

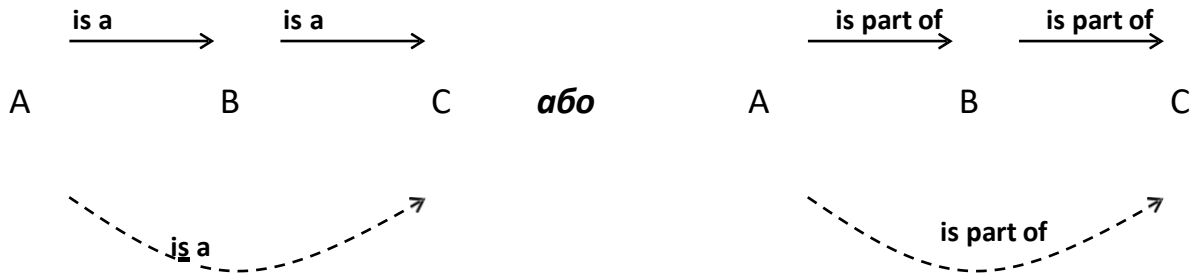
3) правила застосовуються залежно від їх пріоритету. Це правило узагальнює попередній підхід.

ТЕМА 7

Логічний вивід на фреймах і семантичних мережах. Процедури виводу на семантичних мережах. Процедури виводу на фреймах

Для організації **логічного висновку** в експертній системі, що використовує **семантичну мережу**, застосовуються такі механізми.

1. Механізм успадкування з властивістю транзитивності. Даний механізм дозволяє стискати базу знань. Представлена пунктиром дуга в явному вигляді в базі знань не задається.



Використання транзитивності в семантичних мережах

2. Висновок, заснований на базових операціях. До базових операцій відносять:

- Пошук вершини або ребра по імені;
- Перехід від однієї вершини до іншої по зв'язкам.

Мета пошуків і переходів - знайти вершину із заданими властивостями або знайти властивості заданої вершини.

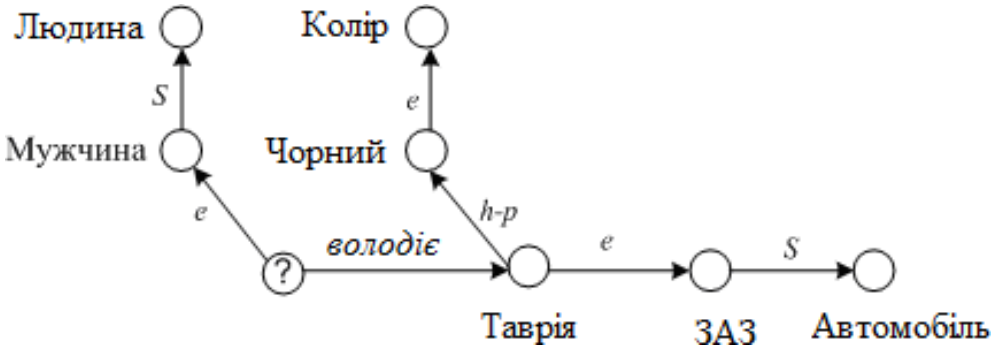
3. Висновок, заснований на зіставленні зі зразком:

запит будується у вигляді фрагмента семантичної мережі, при цьому деякі вершини можуть бути незаповненими;

далі йде зіставлення семантичних мереж запиту і бази знань,

в результаті чого в семантичної мережі бази знань знаходиться фрагмент, ізоморфний фрагменту мережі запиту;

вершини, в яких відсутня інформація, заповнюються у відповідності зі значеннями вершин семантичної мережі бази знань.



Семантична мережа запиту

4. Висновок, заснований на приєднаних процедурах. Процедури активізуються при проходженні по відповідній дузі або через відповідну вершину.

У **фреймовому підході** до подання знань відсутній спеціальний **механізм управління логічним висновком**, і для його організації застосовуються такі способи:

- Естафета приєднаних процедур;
- Успадкування;
- Використання значень за замовчуванням.

Розглянемо реалізацію логічного висновку на прикладі моделі системи, що допомагає корпорації в підготовці звітів про проекти.

Принцип роботи представленої моделі наступний. Якщо системі задано запит: «Мені потрібен звіт про виконання проекту" Нові технології "», то інтерфейсна програма аналізує його, виділяючи ключові слова, і вносить Нові технології в слот Назва звіту наступного порожнього вузла Звіт про виконання проекту (вузол № 15). Передбачається, що програмний інтерфейс дозволяє спілкуватися з системою на мові близькому до природного.

Далі автоматично виконується наведена нижче послідовність дій:

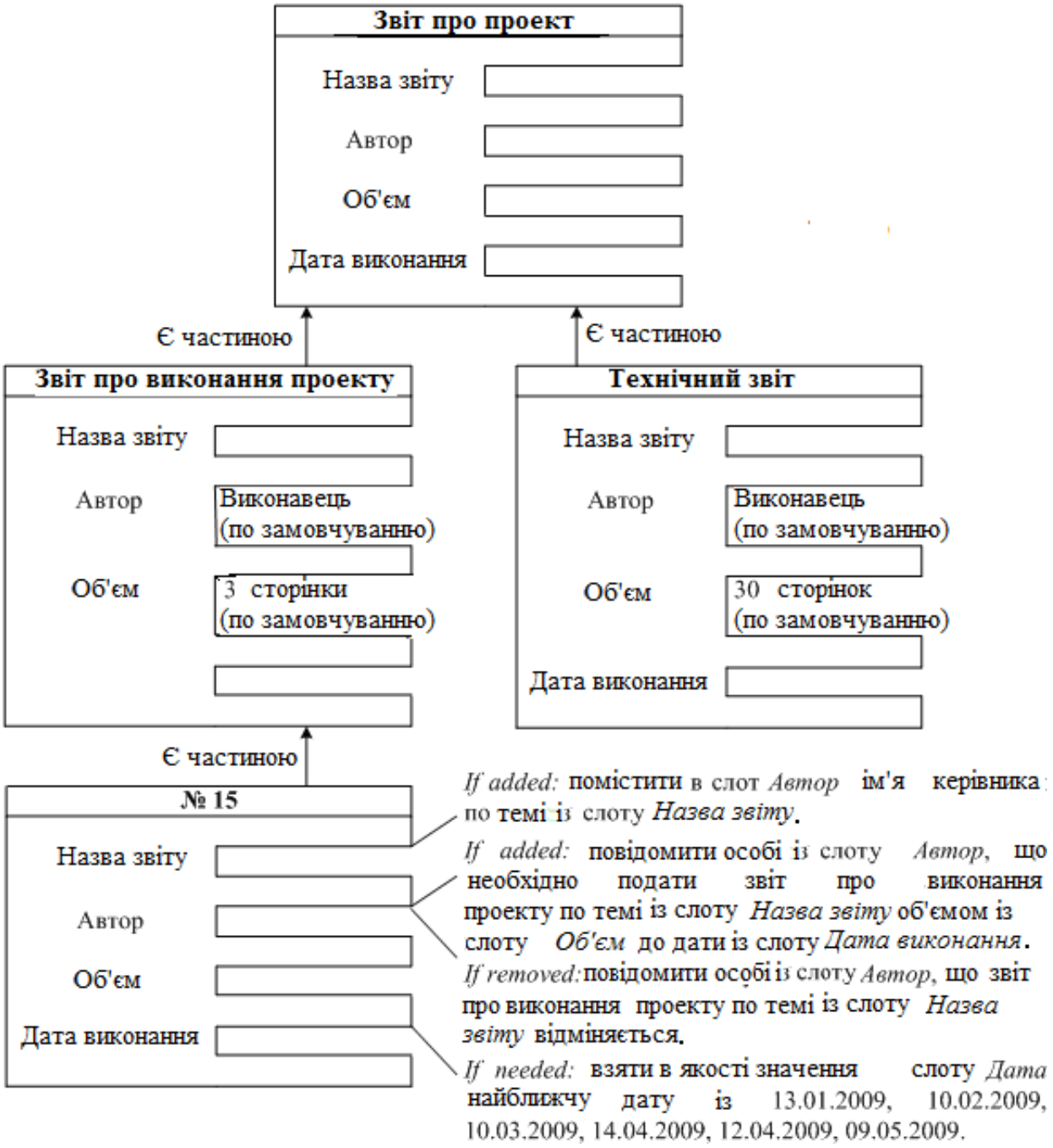
1. Процедура If added, пов'язана зі слотом Назва звіту здійснює пошук керівника цього проекту. Припустимо, що його прізвище Іванов. Процедура вписує його прізвище в слот Автор вузла № 15. Якщо керівник цього звіту не буде знайдений, в слот Автор буде успадковувати значення класу, а саме текст Керівник.

2. Виконується процедура If added, пов'язана зі слотом Автор, оскільки в слот тільки що було вписано нове значення. Ця процедура починає складати повідомлення, щоб відправити його Іванову, але виявляє, що значення об’єму відсутня. Слот Об’єм не пов'язаний з процедурами, однак вище вузла № 15 існує вузол загальної концепції звіту про виконання проекту, що містить значення об’єму. Процедура шляхом успадкування використовує значення об'єму - 3 сторінки.

3. Потім процедура If added, пов'язана зі слотом Автор, знайде, що ще одне значення, яке потрібно додати до повідомлення, т. Е. Дата, відсутня. Відбувається активація процедури If needed, пов'язаної з цим слотом. Аналізуючи поточну дату (наприклад, 01.02.2012), If needed вибере найближчу до неї (наприклад, 10.02.2012) і впише цю дату в слот Дата.

4. Отримавши всю відсутню інформацію, процедура If added, пов'язана зі слотом Автор, становить наступне повідомлення: «Пане Іванов, будь ласка, підготуйте звіт по проекту" Нові технології "до 10.02.2009 об'ємом 3 сторінки».

Якщо в якийсь момент ім'я Іванов буде видалено з слота Автор, то спрацює приєднана процедура If removed і система автоматично відправить йому повідомлення, що його звіт про виконання проекту по темі з слота Назва звіту скасовується.



Фреймова система для організації звітності

Використання фреймів і фреймових систем найбільш повно задовольняє чотирьом основним вимогам до подання знань: интерпретируемость, структурируемой, зв'язність і активність.

ТЕМА 8

Приклад проектування експертної системи. Інженерія знань. Проектування експертної системи. Постановка цілей. Побудова експертної системи. Визначення задач. Збір знань. Вибір експертів

Наука створення експертних систем, як і взагалі ШІ, більшою частиною експериментальна. Такою її робить «стрибок від відомого до невідомого». Будь-яку експертну систему потрібно розглядати як експеримент; який вимагає підтвердження на основі емпіричних результатів . Як і в будь-якому експерименті, можливість отримання задовільних результатів залежить від цілей і точності постановки задачі.

ФАЗА1 Предметна область	ПЛАНУВАННЯ • Постановка цілей • Визначення задач і підзадач • Розробка способів управління і оцінок
ФАЗА 2 Експерт	ІНЖЕНЕРІЯ ЗНАНЬ • Вибір експертів • Збір знань • Розробка бази знань
ФАЗА 3 Користувач	РЕАЛІЗАЦІЯ • Програмування • Попереднє тестування • Доопрацювання

Три фази розробки експертної системи

Крім того, на розвиток експертних систем впливають три характерні фактори:

1. Предметна область.
2. Експерт.
3. Середовище користувача.

Взаємодія цих факторів протягом трьох основних фаз розробки проілюстровано на рисисунку. Процес починається з встановлення цілей і визначення специфічної проблеми, яку належить вирішувати експертній системі.

Постановка цілей. Будь-який експеримент починається з аналізу цілей проекту. Розробники повинні знати природу проблем, які їм належить вирішувати. Вони повинні мати можливість описати задачу, наслідки або результати, очікувані від її вирішення, і значення цих результатів для їх організації. Коротше кажучи, вони повинні точно знати, яку користь принесе застосування даної експертної системи. Це потрібно для того, щоб усунути невизначеність і полегшити розробку спеціальних заходів, здатних збільшити достовірність результатів, розширити систему і зробити її більш визначену. Поставивши правильні цілі, планувальники програм отримують можливість вибору точних завдань і, отже, отримання корисних рішень.

Цілі системи можна розділити на три типи: *кінцеві, проміжні та допоміжні*. *Кінцева* мета описує, яку дію, результат повинні отримувати як наслідок консультації. *Проміжні* мети системи розділяють спільну проблему на підзадачі, описуючи проблеми, які повинні бути вирішені для досягнення кінцевої мети. *Допоміжні* мети допомагають планувальникам визначити конкретні галузі експертизи, що вимагаються для вирішення перерахованих завдань.

Кожна мета - це передбачуваний результат, який може бути отриманий , якщо програма пропонує рішення специфічних проблем. Встановлення їх в якості відправних точок потрібно для того, щоб дати розробникам ясне уявлення, які завдання повинна виконувати система і як вона повинна діяти при їх виконанні.

Визначення задач. Якщо цілі визначені, то типи проблем, які необхідно вирішувати і спосіб підходу програми до їх вирішення стають більш очевидними.

Крім того, у цій фазі велика потреба в обмеженнях. Хоча база знань повинна бути детально вичерпною, але тим не менш її слід обмежувати за фактами і правилами, які потребуються для досягнення поставлених цілей. Наприклад, вік і вагу хворого із захворюванням серця важливі для призначення лікарняного режиму, а колір очей і гострота слуху - ні.

При розробці інструментарію експертної системи бажано передбачити її реальну необхідність для організації. Чи варті передбачувані результати, щоб витрачати на них час, кошти і зусилля персоналу? Наприклад, навряд чи має сенс розробляти систему вартістю 100000 $ для зменшення числа працюючих по найму, якщо загальні витрати на них становлять 10000 $ на рік. Коротше кажучи, чи збільшиться дохід організації, якщо вартість розробки експертної системи відняти від прибутку? Якщо цінність інструменту не перевищує величини капіталовкладень, то його розробка невиправдана.

Збір знань. Друга фаза включає в себе збір, структуризацію і переклад основного вмісту експертизи, необхідної для вирішення поставлених завдань. У цьому процесі звичайно беруть участь як спеціалісти в предметній області, так і інженери-когнітологи. Тип і число експертів часто визначаються складністю нроблеми: чи може один експерт, консультант або керівник надати всі необхідні знання? Чи потрібна група експертів? Невеликі бази знань часто наповнюються з довідників або навчальних посібників. Експертні системи, які створюються на персональних комп'ютерах, найчастіше обмежуються невеликим колом нроблем, і базу знань можно накопичити самостійно, скориставшись допомогою невеликої групи фахівців, або вдавшись до допомоги довідкових посібників. Зміст такої бази знань обумовлюється визначенням проблеми.

Це означає, що база знань може охоплювати будь-яку предметну область, яка описується наборам правил «якщо - то» . Вона може бути проста як набір

інструкцій для вибору форми страхування життя або точна, як правило по використанню поліцейського кийка.

Вибір експертів. Незалежно від предметної області наступним завданням планувальників є вибір експертів. Головна проблема полягає в тому, щоб знайти експерта, що володіє знаннями, що відповідають вибраним цілям і завданням. Так як в основі машини виведення лежить послідовне просування до кінцевої мети, то планувальник повинен подбати про отримання інформації, набутої на основі глибокого досвіду і багаторазових спостережень. Для більшості експертних систем досвід практиків корисніше теорій академіків. У системах економічного планування, наприклад, професор економіки може висловити свій теоретичний погляд на визначальні чинники грошового обороту. Але така експертиза навряд чи замінить практичний досвід експерта, який щодня спостерігає за мінливістю фінансового життя корпорацій. Аналогічно знання фармацевта про хімічну динаміку можуть стати цінною складовою медичної рецептурної системи. Але ця інформація не замінить досвіду фахівця по внутрішнім хворобам, накопиченого шляхом спостереженні за реакцією пацієнтів на дію того чи іншого препарату.

Знання, необхідні для вирішення організаційних проблем, вимагають дотримання деякої пропорції між "внутрішнім" досвідом і "зовнішньою" експертизою. При побудові великих систем знань зі значною кількістю попередньо визначених підзадач , часто має сенс згрупувати інформацію за кількома рівнями організації , від рядового персоналу до адміністрації. Іншими словами, потрібно проаналізувати, як розвивається проблема, яку вигоду отримає організація від її вирішення , і хто повинен брати участь у вирішенні даної проблеми. Припустимо, що в інженерній дослідницькій фірмі ви очолите проект по створенню експертної системи визначення типу кіноплівок, щоб використовувати її при документуванні різних тестових процедур. Ви починаєте пошук знань на самому нижньому рівні ієрархії - з темної кімнати техніка, який "проливає світло" на характерні особливості обробки різних типів плівок. Потім ви розпитуєте кінооператора, який володіє багатим досвідом по поводженню з плівкою, її зарядкою і зйомкою в різних умовах тестування. На наступному рівні, від інженерів-розробників ви отримаєте інформацію, що стосується технічних вимог типових тестових процедур. Завідувач відділом збуту забезпечить вас даними відносно вартості і попиту на кіноплівки і нарешті завдяки Довіднику кінематографіста ви дізнаєтеся про технічні специфікації і підходящих критеріях.

Може виявитися необхідним проведення зовнішньої експертизи для поповнення бази потрібними знаннями з інформацією, отриманою від інших фірм. Перевага незалежного консультанта - в різноманітності досвіду, набутого різними організаціями, отже, він володіє більш глубоким розумінням предмета в області, що обов’язково виходить за рамки внутрішнього досвіду фірми. У наведеному вище прикладі, зовнішній консультант міг би дати додаткові

знання про нові плівки і технології, про яких співробітники організації тільки читали в рекламних оголошеннях.

У тому випадку , коли внутрішня експертиза обмежена або неповна, зовнішній консультант стає головним джерелом інформації.

При побудові невеликих баз знань з обмеженим колом проблем, бажане одне джерело. Об'єднання знань різних експертів ускладнене і їх рекомендації можуть бути навіть суперечливими. Вибирайте експерта - ентузіаста ; процес вилучення знань виснажливе.

Збір інформації та розробка бази знань. Крім пошуку хорошого експерта, група розробників стикається з проблемою ефективного вилучення інформації. Після вибору експерта просто витяг знань менш істотне, ніж витяг правильних знань. Для прискорення процесу перекладу накопиченої інформації в машинну базу знань, розробники часто ділять свої питання на дві окремі категорії: зовнішні і внутрішні.

Зовнішні питання встановлюють дійсний уміст бази знань. Твердження типу: "температура" є ознакою лихоманки і значення "температури" може бути "102 градуса за Фаренгейтом" є прикладами зовнішніх знання. Внутрішні питання, навпаки, фокусуються на ході подій - взаємозв'язках між об'єктами, атрибутами і значеннями. Твердження "Якщо температура = 102 градуси, то у пацієнта лихоманка " представляє внутрішнє знання.

Передача знань повинна ставитися до обох форм його подання. У системі, заснованої на правилах, ми задаємо питання, що відносяться до зовнішніх знань, для того, щоб побудувати пари "об'єкт - значення". Внутрішні знання витягуються для визначення правил, які керують вирішенням завдання в заданій предметній області.

Визначення інтерфейсу користувача третя складова передачі знань пов'язана з користувачам. Вже на ранній стадії розробки необхідно знати, що буде вводити кінцевий користувач. Це потрібно для того, щоб переконатися, чи буде система досить практична і чи забезпечить максимальну сумісність з середовищем, в якому їй належить працювати.

Участь користувача виражається в наступному:

Конкретні завдання. Користувач, стикаючись з конкретними проблемами, може пояснити виникнення проблем і запропонувати можливі варіанти їх вирішення.

Спілкування. Інтерфейс користувача повинен відповідати словнику користувача і рівню його підготовки.

Встановлення зв'язків. Знайомство користувача з причинами і наслідками безцінне в процесі визначення взаємозв'язків фактів в базі знань.

Зворотній зв'язок. Відмінною особливістю хорошою експертної системи є її здатність пояснювати кінцевому користувачеві хід своїх міркувань.

Побудова експертної системи. Після того як знання людини-експерта успішно перенесені в структуровану базу знань, програміст експертної системи може приступати до роботи. Найбільш прямий метод реалізації виконання

правил - безпосереднє включення їх у текст програми. Більш гнучкий підхід полягає у тому, щоб створити повноцінну систему накопичення пар "об'єкт - значення" і зв'язку їх з запропонованими користувачем правилами

ТЕМА 9

Приклад побудови експертної системи. Видобування інформації та розробка бази знань. Визначення інтерфейсу користувача. Представлення фактів в базі знань. Керування фактами в базі знань. Побудова простої експертної системи

Класичні програми - це спосіб зберігання і передачі знань. В програмах у вигляді коду, зрозумілому для ЕОМ, задаються дії, які необхідно виконати для досягнення поставленої цілі (можна сказати, що програма - це "know how"). Позитивною рисою такого способу зберігання знань на ЕОМ є швидкодія: алгоритм пошуку розв'язку задачі заданий чітко і недвозначно, не виконується ні яких лишніх операцій. Але він має суттєві вади: По самій програмі практично неможливо відновити алгоритм, вкладений у програму. Для ілюстрації цього положення достатньо взяти будь-яку програму у вигляді EXE файлу, відновити з неї відповідний асемблерний код (програму), а потім з нього відновити алгоритм, вкладений в цю програму. Навіть з тексту програми на мові високого рівня не просто це зробити. В програми, написані на алгоритмічних мовах тяжко вносити зміни. Забувається навіть власна програма, з якою не працював 3- 4- 5 місяців. Лише невелика, мізерна доля задач, які розв'язуються людьми, мають алгоритм свого розв'язку. Переважна більшість задач пов'язана з пошуком розв'язку. Приклад: медицина (діагноз), прийняття рішень (управління), пошук поламок в різних механізмах і т.д. Таким чином, переважна більшість складених алгоритмічних програм мають справу з так званим "іграшковим" світом, а не з реальними задачами. Для спілкування з ЕОМ за допомогою програм людині треба мати навички роботи з ЕОМ, тобто поруч з основною професією вивчити ще одну (що таке "миша", "гаряча клавіша", клавіша ESC і т.д.). Іншим альтернативним підходом до зберігання знань на ЕОМ, є так звані експертні системи, частина великого розділу науки "Штучний інтелект". Експертні системи - складні програмні комплекси, що збирають знання спеціалістів в конкретних предметних областях і застосовують отриманий досвід для некваліфікованих користувачів. Рішення експертних систем володіють "прозорістю", тобто можуть бути пояснені користувачеві на якісному рівні. Експертні системи здатні поповнювати свої знання в ході взаємодії з експертом. Технології експертних систем можуть з успіхом застосовуватись в різноманітних проблемних областях, таких, як фінанси, нафтова і газова промисловість, енергетика, медицина, освіта тощо. ЕС здатні розв'язувати неформалізовані задачі, що представляють великий і складний клас задач, яким притаманні особливості: помилковість, неоднозначність, неповнота і суперечність початкових даних, знань про предметну область і задачу, що вирішується; велика розмірність простору пошуку рішення; дані і знання, що динамічно змінюються . Експертні системи відрізняються від систем обробки даних тим, що в них в основному використовуються

символьний спосіб представлення, символьний вивід і евристичний пошук рішення.

В розробці ЕС беруть участь представники наступних спеціальностей: Експерт в предметній області, задачі якої буде вирішувати ЕС. Він визначає знання (дані і правила), що характеризують проблемну область, забезпечує повноту і правильність введених в ЕС знань. Інженер по знаннях - фахівець з розробки ЕС. Він допомагає експерту виявити і структурувати знання, необхідні для роботи ЕС і визначає спосіб представлення знань. Програміст з розробки інструментальних засобів (ІЗ), призначених для роботи ЕС. Програміст розробляє інструментальні засоби, що містить всі основні компоненти ЕС, і створює зручний інтерфейс з користувачем. Експертна система працює в двох режимах: режимі придбання знань і в режимі використання.

У режимі придбання знань спілкування з ЕС здійснює інженер по знаннях, який використовуючи компоненту придбання знань, наповнює систему знаннями, які дозволяють ЕС в режимі використання самостійно вирішувати задачі з предметної області. Експерт описує предметну область у вигляді сукупності даних і правил. Дані визначають об'єкти, їх характеристики і значення. Правила визначають способи маніпулювання з даними, характерні для предметної області. На відміну від традиційного підходу розробку експертної системи здійснює не програміст, а інженер по знаннях, який не вміє програмувати.

У режимі використання спілкування з ЕС здійснює користувач, якого цікавить результат і спосіб його отримання. В залежності від призначення ЕС користувач може не бути фахівцем в даній предметній області (він звертається до ЕС за результатом, не вміючи отримати його сам), або бути фахівцем (користувач може сам отримати результат, але він хоче прискорити процес отримання результату або покласти на ЕС рутинну роботу). У режимі використання дані про задачу після обробки їх діалоговою компонентою поступають в робочу пам'ять. Інтерпретатор на основі вхідних даних з робочої пам'яті, загальних даних про предметну область і правил з БЗ формує рішення задачі. ЕС при рішенні задачі не тільки виконує наказану послідовність операції, але і заздалегідь формує її. Якщо реакція системи не зрозуміла користувачеві, то він може отримати довідкову інформацію про кроки логічного виводу.

В самому найпростішому випадку, який ми будемо розглядати, експертна система складається з 3-х частин, запрограмуємо кожну з них.

1. База знань. В символьному виді зберігаються (задаються) знання людини.

В нашому випадку база знань - це текстовий файл, який можна проглядати і редагувати звичайним текстовим редактором. Правила мають таку структуру:

якщо пропуски=багато **і** викладач=злий **то** екзамен=2

Це типова продукційна модель представлення знань.

2. Програма інтерфейсу з користувачем - оболонка експертної системи. Тут користувач задає ім'я файлу (файлів) бази знань, задає цільове запитання, відповідає на запитання, які задає експертна система, отримує (не отримує) відповідь на цільове запитання.

3. Машина виводу. На основі бази знань здійснює пошук відповіді на поставлене запитання. В різних експертних системах пошук здійснюється за різними алгоритмами. В нашому випадку прямий ланцюжок міркувань.

На основі вище сказаного приклад оболонки експертних систем написаній на мові C#, основаної на продукційній моделі представлення знань має вигляд.

```
sing System;
using System.Collections.Generic;
using System.Text;
using System.IO;

namespace ekspertna_sustema
{
   class Program
   {
      static void read_from_file(string filename, out string[][] usloviya,out string[][] resultat)
      {
         string[] buffer = File.ReadAllLines(filename, Encoding.GetEncoding(1251));//копіюєм рядки із файлу в
рядковий масив          int num_str = 0; //кількість правил
         for (int i = 0; i < buffer.GetLength(0); ++i)
         {
            if (buffer[i] != "")
               num_str++;

         }


         //виділяємо пам’ять під масив умов і результатів
         usloviya = new string[num_str][];
         resultat = new string[num_str][];

         //починаємо розбір рядка
         for (int i = 0; i < num_str; ++i)
         {
            if (buffer[i] != "")//якщо рядок із файлу не пустий, починаємо його обробляти
            {
               int t2 = buffer[i].IndexOf("ТО"); //находим початок слова ТО  копіюємо частину рядка між якщо і то в
змінну tmp посинаємо копіювати з індекс 5 тому що 0 – індекс початку слова якщо , само слово якщо довжиною 4
символа, після нього йде пробіл, в сумі 5 символів другий аргумент – довжина підрядка, який копіюємо - t2-5
               String tmp = buffer[i].Substring(5, t2-5);
               String[] usl = tmp.Split('і');//розбираємо умову, розділяємо її на підрядки по символу «і»
               usloviya[i] = new string[usl.GetLength(0)];// виділяємо пам’ять для рядків масива умов
               for (int j = 0; j < usl.GetLength(0); ++j)
               {
                  usl[j] = usl[j].Trim();//видаляємо лишні пробіли
                  usloviya[i][j] = usl[j];//записуємо в масив
               }
               //обробляємо результат виразу (частина рядка після «то»)
               //копіюємо з індексу t2+3 тому що t2 – це початок слова то, семе слово то 2 символа, після нього
пробіл, в сумі 3 символа
```

```
                tmp = buffer[i].Substring(t2 + 3).Trim();
                string[] res = tmp.Split('И');//розбираємо умову, розділяємо на підрядки по символу «і»
                resultat[i] = new string[res.GetLength(0)];//виділємо пам’ять для рядка масиву результатів
                for (int j = 0; j < res.GetLength(0); ++j)
                {
                    res[j] = res[j].Trim();//видаляємо лишні пробіли
                    resultat[i][j] = res[j];//записуємо у масив
                }
            }
        }
    }
    // аргументи: string [] [] usloviya, string [] [] result - це база правил, лічена з фалйа і розібрана функцією read_from_file
    // int nomer_pravila - номер перевіряється правила, List <string> arg - умови, введені користувачем + обчислені в ході перевірки попередніх правил
    // List <string> означає, що це список, кожен елемент якого - рядок. Приблизно те ж саме, що і масив, але довжина може бути не відома заздалегідь,
    // що в даному випадку зручно, оскільки при перевірці бази додаються нові умови
    static bool proverka_pravila(string[][] usloviya,string[][] resultat,int nomer_pravila, List<string> arg)
    {
        // правило вважається виконаним, коли виконані всі його під умови, перераховані через "І"
        // правило ЯКЩО А = 2 І Б = 2 ТО С = 0 буде вважатися виконаним, якщо в списку arg будуть рядка "А = 2", "Б = 2 "
        int vypolneno = 0; //кількість виконаних передумов
        int i = 0;
        for (i = 0; i < usloviya[nomer_pravila].GetLength(0);++i)
        {
            int j = 0;
            while (j < arg.Count)
            {
                if (usloviya[nomer_pravila][i] == arg[j])
                {
                    vypolneno++;
                    break;
                }
                j++;
            }
        }
        if (vypolneno == i)//якщо виконані всі передумови даного правила, то правило виконане і ф-я повертає true
            return true;
        else
            return false;//якщо не всі, то правило не виконано, повертаєм false
}



    static void Main(string[] args)
    {
        //формат правила якщо ... то ...
        string[][] usloviya; // вирази між ЯКЩО і ТО, тобто умови виконання правила string[][]
        resultat; // вирази між ЯКЩО і ТО, тобто результат виконання правила

        string filename = "base.txt";
        string input;// ця змінна зберігає рядок, що введений користувачем з консолі – вхідні аргументи

        read_from_file(filename,out usloviya,out resultat);//зчитуємо базу з файлу, розбираємо, записуємо результат розбору в відповідну зміну
```

```
Console.WriteLine("Введіть вхідні умови в формате \"объект=значення\" через пробіл");

input = Console.ReadLine();
```

```
        string[] str = input.Split();//розбиваємо введений користувачем рядок на окремі умови по символу пробіл
        List<String> arg = new List<String>();//введені користувачем умови+вичеслені в ході перевірки правил try
        {
            for (int i = 0; i < str.GetLength(0); ++i) arg.Add(str[i]);


            Console.WriteLine("\nРезультати:");
            // починаємо послідовну перевірку бази правил, якщо правило виконується, додаємо його у вихідні
дані
            int s = 0; // кількість умов в списку arg до перевірки всієї бази правил
            int end = arg.Count;  // кількість умов в списку arg після перевірки всієї бази правил
            bool izmeneno = false; // показує, чи змінювалися якісь умови

            while (end > s || izmeneno==true) // перевіряємо базу правил до тих пір, поки не додаються нові умови
або не змінюються існуючі
            {
                s = arg.Count;
                izmeneno = false;

                for (int i = 0; i < resultat.GetLength(0); ++i)// послідовно перевіряємо всі наявні правила
                {
                    if (proverka_pravila(usloviya,resultat,i, arg))// якщо правило виконується, обробляємо його
результат
                    {
                        for (int j = 0; j < resultat[i].GetLength(0); j++)
                        {
                            if (arg.IndexOf(resultat[i][j]) == -1) // якщо такої умови ще не було в списку arg
                            {
                                Console.WriteLine(resultat[i][j]);// то виводимо його на консоль

// може бути, що наявного об'єкту присвоєно нове значення,
// наприклад, було F = 2, потім у результаті виконання правил стало F = 4
// тоді в списку arg повинно залишитися толлько F = 4
                                int t = resultat[i][j].IndexOf('=');
                                String a = resultat[i][j].Substring(0, t);// назва об'єкта - те, що до символу '='
                                for (int y = 0; y < arg.Count; ++y)
                                {
                                    t = arg[y].IndexOf('=');
                                    String b = arg[y].Substring(0, t);
                                    if (a == b)// якщо назви об'єктів збігаються, міняємо стару умову на нову
                                    {
                                        arg[y] = resultat[i][j];
                                        izmeneno = true; // і даній змінні присвоюємо значення істини
                                        break;
                                    }
                                }

                                if (!izmeneno)
                                    arg.Add(resultat[i][j]); // додаємо результат виконання правила в список arg

                            }
                        }
                    }
                }
                end = arg.Count; // кількість значень у списку arg після перевірки всієї бази
            }
        }
```

```
            catch
            {
                Console.WriteLine("Помилка у вхідному файлі! ");
            }
            Console.Read();
        }
    }
}
```

ТЕМА 10

Теоретичні та прикладні аспекти проектування баз знань. Інструментальні засоби побудови експертних систем. Мови штучного інтелекту. Оболонки експертних систем.

Розробка будь-якої системи управління БЗ і ЕС - велика і складна робота, потребує залучення кваліфікованих фахівців-експертів і розробників-програмістів. Природно, що для створення подібних систем розробляються різні інструментальні засоби. За інструментальним можливостям ці засоби можна розділити: на основні та допоміжні. Основні засоби утворюють мови, призначені для опису компонентів ЕС і системи програмування, що підтримують ці мови. Допоміжні засоби призначені для спрощення окремих етапів розробки або тестування тих чи інших компонентів ЕС. Наприклад, для наповнення і модифікації БЗ, що містить факти і правила.

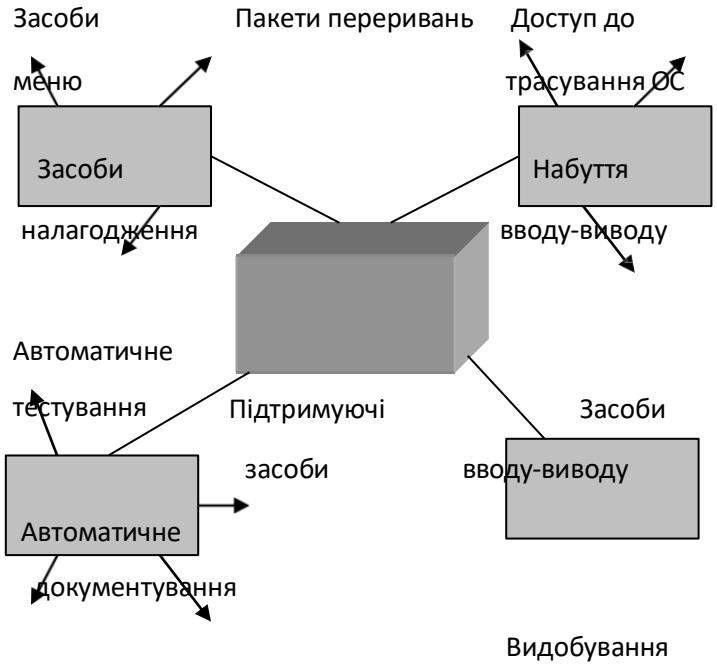
Процес створення експертних систем значно змінився за останні роки. Завдяки появі спеціальних інструментальних засобів побудови експертних систем значно скоротились терміни та зменшилась трудомісткість їх розробки. Спектр використовуваних інструментальних засобів для побудови ЕС дуже широкий та існує кілька класифікацій. Але всі вони мають загальні групи, які залежать від методів побудови ЕС.

Мови програмування, застосовувані для роботи в області ЕС, - це, як правило, проблемно-орієнтовані мови такі як PASCAL, FORTRAN або мови обробки текстів - LISP і PROLOG. Проблемно-орієнтовані мови розроблені для спеціального класу задач: наприклад наукових, математичних і статистичних досліджень. Мови обробки текстів розроблені для прикладних областей штучного інтелекту. Наприклад, LISP має механізми для маніпулювання символами у формі спискових структур. LISP і PROLOG неухильно завойовують популярність. Популярність мов ґрунтується на декількох їх властивостях: легкому маніпулюванні символами, автоматичному управлінню пам'яттю, розвиненому редагуванні і засобам налагодження, єдиному підході в поданні програм.

Лісп існує у двох варіантах: Інтерлісп, Меклісп (мають різні засоби підтримки).

Мови програмування подібні Ліспу надають гнучкість розробнику ЕС, але не підказують йому, як представляти знання або як побудувати механізм доступу до бази знань.

Мови інженерії знань володіють меншою гнучкістю, оскільки розробник системи повинен користуватися схемою управління за допомогою механізму висновків.



Мал. 1 Компоненти підтримуючого середовища.

Мови інженерії можна розділити на *скелетні* та *універсальні*. Скелетна мова - це «голі» ЕС, без спеціальних предметних знань.

Скелетні системи забезпечують структуризацію знань і готові механізми висновків. Але їм не вистачає загальності та гнучкості. Вони застосовуються тільки до вузького класу проблем і сильно обмежують можливості розробника ЕС.

Універсальні мови можуть бути застосовані до проблем різного типу в різних прикладних областях. Вони забезпечують більші можливості управління пошуком даних і доступом до них, ніж скелетні системи, але їх важче використовувати.

Допоміжні засоби побудови ЕС: складаються з програм, які надають допомогу в набуванні знань в експерта і подання їх, а також програм, які допомагають розробити проекти ЕС. Цих засобів значно менше і вони діляться на дві групи:

- засоби проектування систем;

- засоби набуття знань.

Допоміжні засоби включають також засоби підтримки - допоміжні програми, що додаються до засобів побудови ЕС, щоб спростити її.

Компоненти підтримуючого середовища також відносяться до допоміжних засобів.

Розглянемо докладніше засоби пояснення. Деякі системи мають вбудований механізм пояснення (такі як ENYCIN). В інших немає цього механізму міркування, а його потрібно використовувати. Існує кілька типів механізму пояснення:

- ретроспективний розподіл;

- гіпотетичний розподіл;

- контрфактичний розподіл.

При першому випадку пояснюється, як система досягла конкретного стану. Система може описати правило, яке призвело до висновку або частину ланцюжка правил. У другому випадку - система пояснює, що вийшло, якби конкретний факт або правило були іншими. У третьому випадку система пояснює, чому очікуваний висновок не вийшов.

Інструментальні засоби розвиваються в часі.

1 етап. Побудова експериментальних систем для спеціальних завдань. ЕС виходить неперевірена, вузькоспеціалізована, повільна, неефективна, тому розробник не перевіряє її на інших завданнях.

2 етап. Дослідницька система. Цей інструмент буде достатньо протестований розробником, але при цьому залишається повільним і неефективним. Розробникам подобається експериментувати, вносити нововведення, але вони не налагоджують ЕС на виконання декількох завданнь, не звертають увагу на швидкість і ефективність роботи.

3 етап. Комерційна система. Це інструментальний засіб є відшліфованим, добре підтримуваним і швидким. Вони розробляються з урахуванням побажань користувачів і полегшують проблему взаємодії людина-машина.

1. До представників експериментальних систем відносяться мови інженерії знань ROCET, SEEK, що є допоміжним інструментом при побудові та вдосконаленні ЕС.

ROCET допомагає предметному експерту розробити базу знань для діагностичної ЕС.

SEEK пропонує можливі способи узагальнити і спеціалізувати правила (пробні випадки). Система рекомендує і спосіб модифікації, але залишає користувачеві самому вирішувати, як узагальнити або конкретизувати ці компоненти. SEEK висуває ці припущення, грунтуючись на метаправилах.

2. Представниками дослідних систем є мови інженерних знань - EMYCIN, EXPERT, OPS-5, ROSIE, LOOPS.

3. Комерційні системи можна розбити на 3 категорії:

- Мови програмування;
- Мови інженерних знань;
- Допоміжні засоби.

4. Мови програмування - Лісп і його модифікації: MACLISP, ZETALISP, INTERLISP, а також мову ПРОЛОГ.

Допоміжні засоби розроблені для того, щоб надавати допомогу при набуванні знань. Це TIMM, RULEMASTER, EXPERT-EASE. Системи дуже схожі, користувач визначає проблему в термінах різних рішень, які можуть бути прийняті, а також назв і факторів, які слід враховувати при виборі рішення.

Мови інженерних знань. Ці мови є функціонально повними засобами розробки ЕС, які поєднують потужні мови й хитромудрі засоби підтримки. В основному засновані на продукційних системах (ART, KES, M1, KEE). EXPERT
- скелетна система.

OPS-5, ROSIE - універсальні системи.

База знань може бути дуже і дуже великою. Будучи введеною в машину один раз, знання зберігаються назавжди. Людина ж має обмежену базу знань, і якщо дані довгий час не використовуються, то вони забуваються і назавжди втрачаються.

5. Системи, засновані на знаннях, стійкі до "перешкод". Експерт користується побічними знаннями і легко піддається впливу зовнішніх факторів, які безпосередньо не пов'язані з розв'язуваним завданням. ЕС, які не обтяженні знаннями з інших областей, за своєю природою менш схильні "шумів". З часом системи, засновані на знаннях, можуть розглядатися користувачами як різновид тиражування - новий спосіб запису і поширення знань. Подібно до інших видів комп'ютерних програм вони не можуть замінити людину у вирішенні завдань, а скоріше нагадують знаряддя праці, які дають можливість вирішувати завдання швидше і ефективніше.

6. Ці системи не замінюють фахівця, а є інструментом у його руках.

ТЕМА 11

Генетичні алгоритми. Поняття генетичного алгоритму. Оцінка. Відбір. Рекомбінація. Недоліки генетичного алгоритму.

Генетичний алгоритм являє собою техніку оптимізації, яка є певною аналогією до біологічного процесу еволюції, тобто заснована на природному відборі. Генетичний алгоритм можна застосовувати для різноманітних задач оптимізації, які не завжди вдало вирішуються за допомогою стандартних оптимізаційних алгоритмів.

Еволюційна теорія стверджує, що кожен біологічний вид цілеспрямовано розвивається і змінюється для того, щоб найкраще пристосуватися до навколишнього середовища. У процесі еволюції багатьох видів комах і риб набули захисного забарвлення. Можна сказати, що еволюція - це процес оптимізації всіх живих організмів.

Основний механізм еволюції - це природний відбір. Його суть полягає в тому, що краще пристосовані особини мають більші можливості для виживання і розмноження, а, отже, приносять більше потомства, ніж погано пристосовані. При цьому, завдяки передачі генетичної інформації, нащадки успадковують від батьків їх основні якості. Після зміни кілька десятків або сотень поколінь середня пристосованість особин цього виду помітно зростає.

При розмноженні тварин відбувається злиття двох батьківських клітин та їх ДНК взаємодіють, створюючи ДНК нащадка. Основний спосіб взаємодії - кроссовер (cross-over, схрещування). При кроссовері ДНК предків ділиться на частини, а потім обмінюється своїми половинками.

При успадкування можливі мутації радіоактивності чи інших впливів, які можуть змінитися певні гени одного з батьків. Змінені гени передаються нащадку і надають йому нових властивостей. Якщо ці нові властивості корисні, вони, швидше за все, збережуться в даному виді - при цьому відбудеться стрибкоподібне підвищення пристосованості виду.

Генетичний алгоритм - це просто модель еволюції в природі. У ньому використовуються як аналог механізму генетичного наслідування, та і аналог природного відбору. Генетичний алгоритм, розроблений Джоном Холландом є програмою пошуку рішень заданої проблеми. Іншими словами відбувається циклічний процес схрещування індивідуумів (векторів рішень) і зміни поколінь.

При описі алгоритмів використовуються визначення, запозичені з генетики:

Популяція – скінченна множина особин

Особина - представляється хромосомами з закодованими в них множинами параметрів задачі.

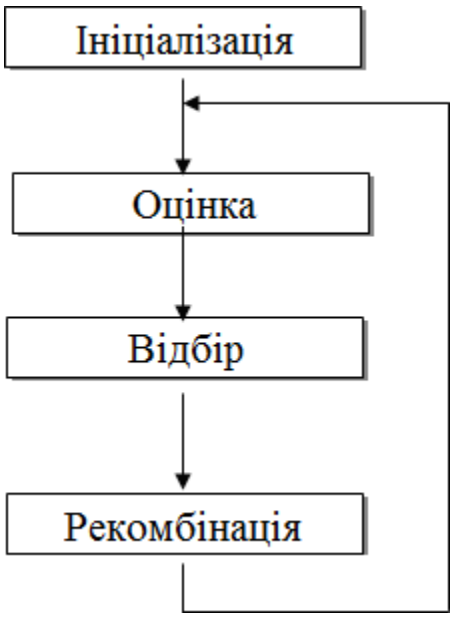
Хромосоми - впорядковані послідовності генів

Ген - атомарний елемент (зазвичай представляється окремою ознакою)

Також важливим поняттям в генетичних алгоритмах є функція здоров'я, яка являє собою міру пристосованості даної особини в популяції.

Особливістю генетичного алгоритму є те, що він не намагається оптимізувати єдине рішення. Він працює з групою рішень, які кодуються у вигляді так званих хромосом. Окремі гени хромосоми являють собою унікальні змінні досліджуваної проблеми.

Виконання генетичного алгоритму включає такі основні етапи:



Розглянемо більш детально кожен з етапів

1. Ініціалізація

Щоб змоделювати еволюційний процес необхідно визначити відправну точку роботи алгоритму. Зазвичай це виконується шляхом створення довільного набору хромосом, хоча можливий варіант додавання в популяцію свідомо «здорових» хромосом.

Важливо відзначити той факт, що варто безпосередньо подбати про те, щоб початкова популяція була різноманітна. Це означає, що значення функції здоров'я у кожної хромосоми повинно бути різним, в іншому випадку можна отримати незадовільні результати роботи алгоритму (такі як передчасна збіжність).

Приклад: Розглянемо задачу оптимізації функції:

$$Z = \frac{X}{X^2 + 2 \times Y^2 + 1}$$

Для застосування генетичного алгоритму визначимо початкову популяцію з чотирьох хромосом з двома генами x і y:

	A	B	C	D
X	-2	-1	0	2
Y	0	-2	-1	1

2. Оцінка

На етапі оцінки для кожної хромосоми зазвичай перевіряється значення функції здоров'я - як правило цільової функції конкретного завдання.

Приклад: У запропонованому вище прикладі обчислимо якість (функцію здоров'я) кожної хромосоми:

$Z_A = (-2)/(4+0+1) = -0.4$

$Z_B = (-1)/(1+2*4+1) = -0.1$

$Z_C = 0,$

$Z_D = 2/(4+2*1+1) = 0.286$

3. Відбір

Відбір в генетичному алгоритмі - це процес вибору з популяції особин, які підходять для формування нової популяції.

Відбір є найбільш важливим і важким етапом генетичного алгоритму. Він здійснюється на підставі здоров'я хромосом. Як правило, відбір (ймовірність участі в схрещуванні) особини береться пропорційно до її пристосованості (значення функції здоров'я), хоча даний процес є двостороннім: якщо включити в вибір тільки дуже здорові особини, рішення стає занадто обмеженим унаслідок недостатньої різноманітності, з іншого боку, якщо вибір здійснюється довільно, немає гарантії того, що здоров'я наступних поколінь буде поліпшуватися. Також часто використовується так звана стратегія елітизму, при якій кілька кращих індивідуумів переходить до наступне покоління без змін, не беручи участь в кросовері і відборі.

Приклад:

Відсортуємо хромосоми за якістю

	D	C	B	A
X	2	0	-1	-2
Y	1	-1	-2	0
Z	0.286	0	-0.1	-0.4

Хромосома A з найменшою якістю вибуває з процесу еволюції. Хромосома D з найвищою якістю буде брати участь у процесі рекомбінації з найвищим пріоритетом

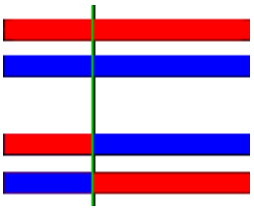
4. Рекомбінція

При рекомбінації частини хромосом переміщаються, можливо навіть змінюються, а отримані нові хромосоми повертаються назад у популяцію для формування наступного покоління. Перша група хромосом зазвичай називається батьками, а друга - дітьми. На даному етапі з однаковою ймовірністю застосовується один з так званих генетичних операторів.

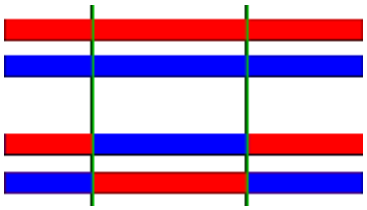
Серед основних генетичних операторів можна виділити таких два оператори:

Перехресне схрещування (Кросовер)

Даний оператор бере дві хромосоми, розділяє їх в довільній точці, а потім міняє місцями отримані "хвости". При цьому утворюються дві нові хромосоми.



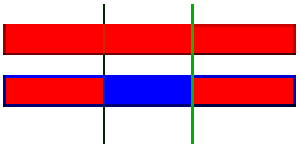
Крім поділу в одній точці також можливо застосовувати поділ відразу в декількох точках:



Важливо відзначити, що при перехресному схрещуванні в популяції не створюється новий матеріал, а виконується просто його зміна. Це дозволяє алгоритмом виконувати пошук вирішення проблеми серед існуючих. Оператор схрещування є найважливішим у процесі рекомбінації і використовується найчастіше

Мутація

Даний оператор вносить одну або кілька довільних змін до генів хромосоми. Саме він дозволяє створювати новий матеріал для популяції, розширюючи тим самим область пошуку рішення.



Вимогою рекомбінації є те, що кожне наступне покоління має бути в середньому кращим за попереднє. Коли пристосованість індивідуумів перестає помітно збільшуватися, процес зупиняють і як рішення задачі оптимізації беруть найкращого зі знайдених індивідуумів.

Приклад: З трьох відібраних хромосом проведемо схрещування і обчислимо відповідну для них функцію корисності (хромосома D буде схрещуватися з B і C, так як має найкращу функцію здоров'я):

	A’	B’	C’	D’
X	-1	0	2	2
Y	1	1	-2	-1
Z	-0.25	0	0.154	0.286

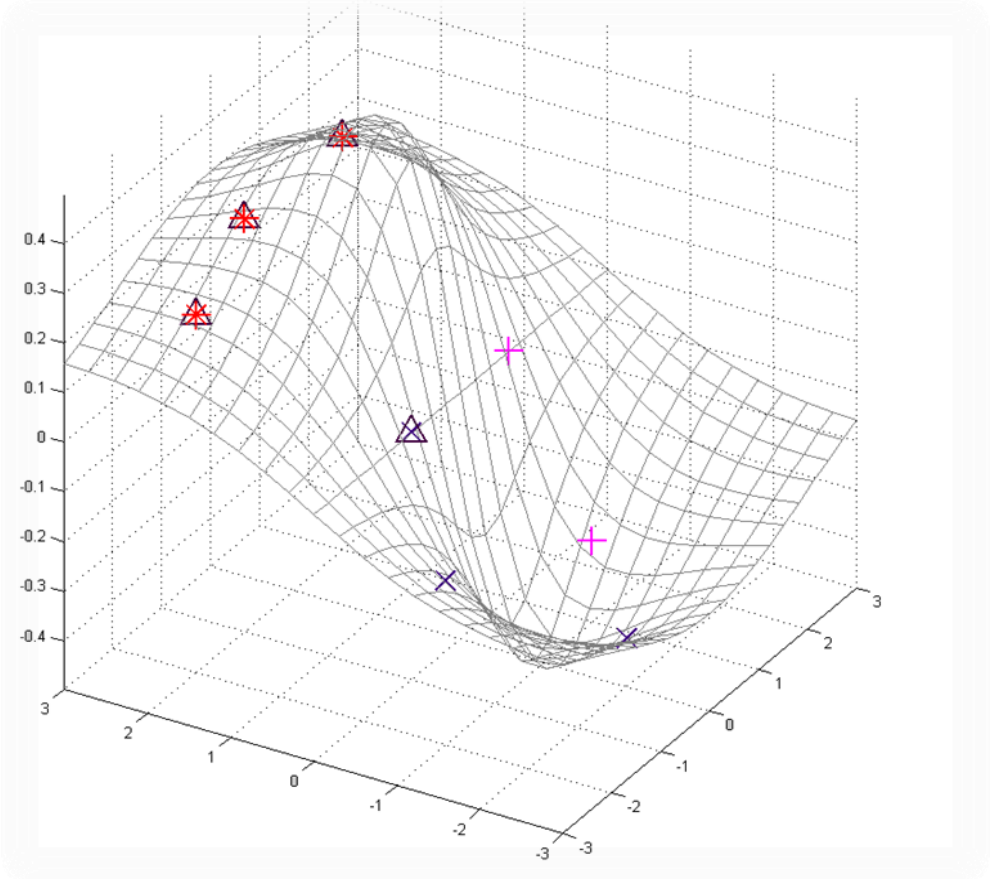
Сортуємо хромосоми за якістю. Перша хромосома вибуває, решта дають таке потомство (будемо схрещувати D'с B' і C'):

	A’’	B’’	C’’	D’’
X	0	2	2	2
Y	-1	-1	1	-2
Z	0	0.286	0.286	0.154

Хромосоми B" і C" мають однакову якість, виберемо в порядку проходження (B"). Хромосома A" вибуває

	A’’’	B’’’	C’’’	D’’’
X	2	2	2	2
Y	1	1	-2	-1
Z	0.286	0.286	0.154	0.286

Якість кращої хромосоми однакова 0.286 (вибирається одна з трьох).
Розглянемо графік даної функції:



x - початкова популяція
+ - Перша ітерація
Δ - друга ітерація
* - Третя ітерація

Видно, що сумарна якість популяції зростає на кожній ітерації

Генетичний алгоритм не позбавлений певних недоліків. Серед основних можна виділити такі дві проблеми:

1. Передчасна збіжність

Зазвичай ця проблема пов'язана з недостатньою різноманітністю хромосом в популяції. Найпоширенішою причиною даного ефекту є маленький розмір популяції. Іншою причиною може служити алгоритм відбору.

2. Епістазис

Епістазисом називають внутрішню залежність між змінними, закодованими в хромосомі. Якщо жоден ген не пов'язаний з іншим, епістазис дуже малий, при залежних генах епістазис високий. Зазвичай рішення даної проблеми полягає в тому, щоб зберігати гени, близько пов'язані один з одним. При групуванні залежних генів істотно знижується ймовірність того, що вони будуть зруйновані при застосуванні генетичних операторів.

Генетичний алгоритм використовується для вирішення багатьох завдань оптимізації. Так як його ефективність багато в чому залежить від подання рішення, ним можна оптимізувати як числові так і символьні дані. Генетичні алгоритми можуть застосовуватися для вирішення таких завдань:

- Оптимізація функцій
- Оптимізація запитів в базах даних
- Різноманітні задачі на графах
- Налагодження та навчання штучної нейронної мережі
- Складання розкладів
- Ігрові стратегії
- Теорія наближень
- Штучне життя
- Біоінформатика
- Та ін.

ТЕМА 12

Нечітка логіка в системах прийняття рішень. Поняття про нечітку логіку. Використання нечіткої логіки в експертних системах прийняття рішень.

Ми звикли до «жорстких», точних обчислень. Ми точно знаємо, що два на три - це шість. І це дуже добре, так як таблиця множення (а з нею правила додавання, віднімання, ділення) - прекрасна математична основа (та що там основа - математична модель!) для багатьох предметних областей. Вона прекрасно підходить, наприклад, для проведення фінансових розрахунків. Цей прекрасний чіткий світ! ...

Ну а хто нам відповість на питання: «А скільки буде приблизно два помножити на щось близьке до трьох»? Питання не пусте. Подібного роду завдання виникають в реальному житті часто-густо! Ми частіше оперуємо нечіткими поняттями, ніж чітко визначеними і формалізованими. Згадайте: молодий чоловік, висока ціна, гарячий чай, а не «чай температури 83 градусів».

Повертаючись до прикладу з множенням, неважко знайти нечітку задачу з життя. Наприклад, «Скільки кг картоплі потрібно запасти в магазині на день, якщо в магазин прийде середня кількість покупців і кожен купить трохи картоплі?». Причому «середня кількість покупців» може бути завтра 100 чоловік, а післязавтра - 80. Те ж саме і з поняттям «трохи картоплі». Оце завдання! Але, виявляється, воно цілком вирішуване. Тільки потрібно перейти до «м'яких обчислень» (термін введений Лофті Заде в 1994 році).

А історія питання така.

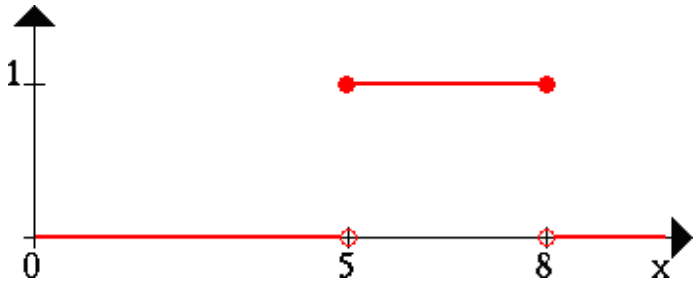
У 1965 році Лофті Заде (Lotfi A. Zadeh, професор технічних наук Каліфорнійського університету в Берклі) запропонував теорію нечітких множин (fuzzy set theory), що представляє собою формалізм, призначений для формування суджень про нечіткі поняття.

А де ж «собака зарита»? А ось де:

Визначимо чітку множину А всіх дійсних чисел від 5 до 8.

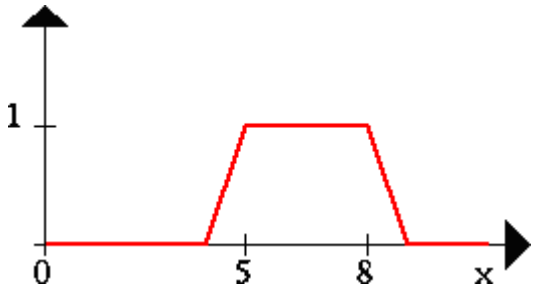
А = [5,8]

Характеристична функція (функція належності - Membership Function) множини А ставить у відповідність число 1 або 0 кожному елементу в X, залежно від того належить даний елемент підмножині А чи ні. Результат представлений на наступному малюнку:

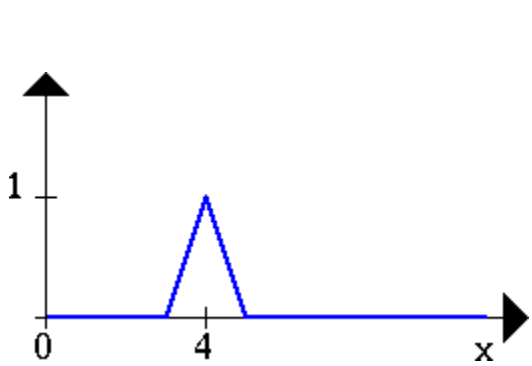


Можна інтерпретувати елементи, яким поставлена у відповідність 1, як елементи, що знаходяться в множині А, а елементи, яким поставлений у відповідність 0, як елементи, що не знаходяться в множині А. Все чітко!

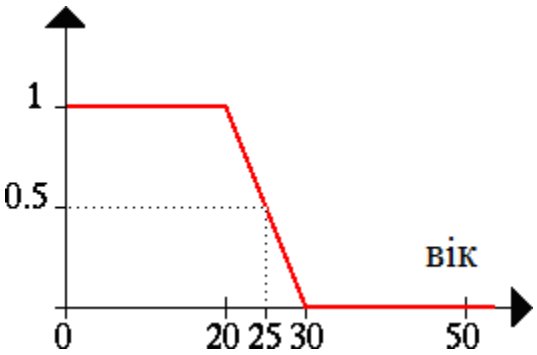
А що робити з нечітким інтервалом від 5 до 8? Як визначити нечітку множину А* «десь в районі 5, 8, може трохи більше або менше. А ось як - визначимо для нього характеристичну функцію так:



Аналогічно можна визначити характеристичну функцію для:



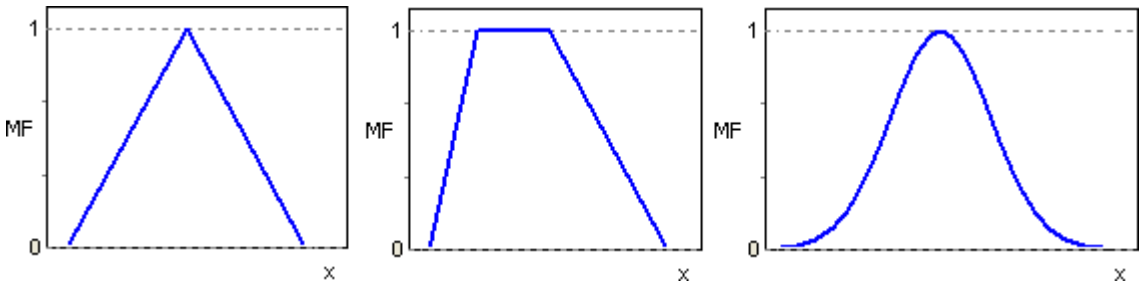
«В * - Близько чотирьох»



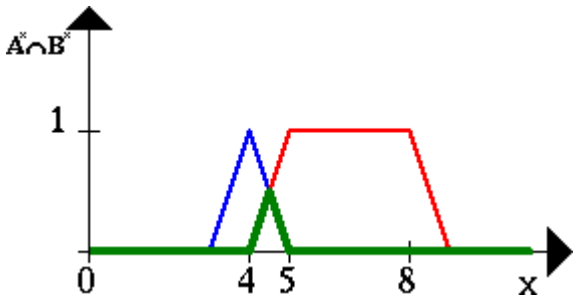
«Молода людина»

Тобто 18-річний чоловік молодий на всі 100%, а от 25-річний все ще молодий, та не зовсім - відсотків на 50 ...

Взагалі то існує понад десяток типових форм кривих для задання функцій належності. Однак найбільше поширення отримали: трикутна, трапеціїдальна і гаусова функції приналежності:



Для нечітких множин визначені і відповідні операції, такі як перетин, об'єднання і заперечення нечітких множин. Так, наприклад на питання «Що з себе представляє множина між 5 і 8 **І** близько 4» відповідає результат перетину нечітких множин А * і В * (визначається як нечітке "**І**": $A^*\cap B^*$: $MF_{A^*B^*}(x)=\min(MF_{A^*}(x),MF_{B^*}(x))$, де MF - функція належності відповідної нечіткої множини):



У 1975 році той же невтомний Заде на основі теорії нечітких множин запропонував теорію нечіткої логіки (fuzzy logic theory). На відміну від класичної, що розвивалася з давніх часів і оперує поняттями ІСТИНА і ХИБА (ТАК і НІ), нечітка логіка дозволяє оперувати поняттями, для яких потрібно мати більше значень, ніж тільки ТАК і НІ.

У теорії нечітких множин та нечіткої логіки вводиться поняття лінгвістичної (ЛЗ) і нечіткої змінної або терма.

Лінгвістична змінна - це набір п'яти елементів (X, T (X), U, G, M,), де X - ім'я змінної, T (X) – множина термів, тобто множина імен (позначень) лінгвістичних значень X, U - область міркувань (the universe of discourse), G - правило (the grammer) генерування імен і M – множина семантичних правил зв'язку кожного X з тим, що воно позначає.

ЛЗ визначає деяке поняття з предметної області (наприклад, «ВІК», «ТЕМПЕРАТУРА», «КІЛЬКІСТЬ ПОКУПЦІВ» і т.п.). З ЛЗ зв'язуються кілька термів - значень. Наприклад, для ЛЗ «ВІК» можуть бути визначені терми (нечіткі змінні) МОЛОДИЙ, СЕРЕДНІЙ, ЛІТНІЙ, для яких визначені відповідні функції належності.

Тепер ми можемо залучити можливості одного з напрямків систем штучного інтелекту - експертних систем - для побудови логічного висновку, що оперує нечіткими знаннями. Нечіткі знання формалізуються у вигляді нечітких правил виду

P_1: ЯКЩО $x_1 \in A_{11}$ ... I ... $x_n \in A_{1n}$, ТО $y \in B_1$

...

P_i: ЯКЩО $x_1 \in A_{i1}$ ... I ... $x_n \in A_{in}$, ТО $y \in B_i$

...

P_m: ЯКЩО $x_1 \in A_{i1}$ ... I ... $x_n \in A_{mn}$, ТО $y \in B_m$,

де x_k, k = 1..n - вхідні змінні; y - вихідна змінна; A_{ik} - задані нечіткі множини з функціями належності. Повертаючись до нашого завдання «А скільки буде приблизно два помножити на щось близьке до трьома?» Ми можемо визначити нечітку, «м'яку» таблицю множення, задану нечіткими правилами виду:

ЯКЩО число близько двох помножити на число близько трьох, ТО отримаємо щось близько шести.

Для нашого цілком реального прикладу «Скільки кг картоплі потрібно запасти в магазині на день, якщо в магазин прийде середня кількість покупців і кожен купить трохи картоплі?» Можна ввести такі поняття:

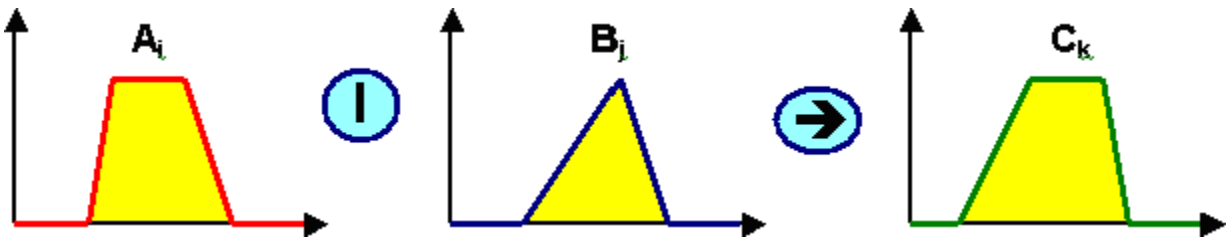
Лінгвістична змінна	Терми
A: КІЛЬКІСТЬ ПОКУПЦІВ	$МАЛО_A$, $СЕРЕДНЬО_A$, $БАГАТО_A$
B: ВАГА ПОКУПКИ КАРТОПЛІ	$МАЛО_B$, $СЕРЕДНЬО_B$, $БАГАТО_B$
C: ВАГА ЗАПАСУ КАРТОПЛІ	$МАЛО_C$, $СЕРЕДНЬО_C$, $БАГАТО_C$, $ДУЖЕ\_БАГАТО_C$,

Нехай не бентежить те, що використовуються семантично еквівалентні терми для різних лінгвістичних змінних - нижній індекс вказує на те, що для кожного з них визначена своя функція належності. Так $БАГАТО_B$ і $БАГАТО_C$ мають різний порядок - для покупки 10кг вже може бути багато, а для запасу в магазині - крапля в морі.

Нечіткі правила можуть бути такими:

№	ЯКЩО	І	ТО
P_1	A є $МАЛО_A$	B є $МАЛО_B$	C потрібно $МАЛО_C$
P_2	A є $МАЛО_A$	B є $ПОСРЕДНЬО_B$	C потрібно $МАЛО_C$
P_3	A є $ПОСРЕДНЬО_A$	B є $ПОСРЕДНЬО_B$	C потрібно $ПОСРЕДНЬО_C$
	…	…	…
P_N	A є $БАГАТО_A$	B є $БАГАТО_B$	C потрібно $ДУЖЕ\_БАГАТО_C$

Графічно правила можна було б представити у вигляді:

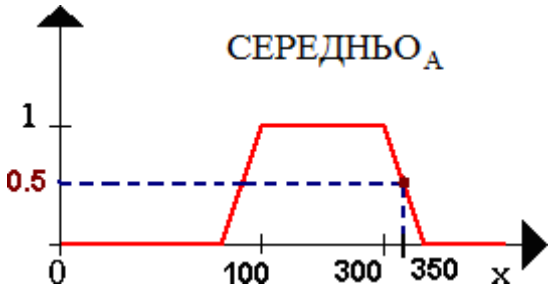


Тут A_i, B_j і C_k - відповідні функції належності для термів лінгвістичних змінних A, B, C.

Ну а тепер застосуємо ці правила грунтуючись на алгоритмі Мамдані. Технологія їх застосування наступна:

1. Визначаємося з завданням. Нехай в конкретний момент часу вона формулюється так: «На завтра прогнозується потік покупців в кількості 350 чоловік. У цю пору року зазвичай одним покупцем купується не більше 3-х кг картоплі. Який потрібно зробити запас картоплі в магазині на завтра? ». (Питання, «А чому б не вирішити цю задачу просто - помножити 350 на 3, отримати 1050 - і все!» - Буде розглянуто пізніше).

2. Виконується так звана фазифікація - приведення до нечіткості. У результаті чіткі значення X* = 350 для A і Y* = 3 для B приводяться до нечітких понять. Використовуючи відповідні функції належності їм приписується «коефіцієнт належності» до відповідного терму. Наприклад, 350 покупців можна віднести до нечітких множин $СЕРЕДНЬО_A$



з упевненістю в 50% (коефіцієнт власності = 0.5).

Тобто знаходяться рівні відсікання для передумов кожного з правил (з використанням операції мінімуму):

$$\alpha_1 = A_1(x_0) \wedge B_1(y_0),$$

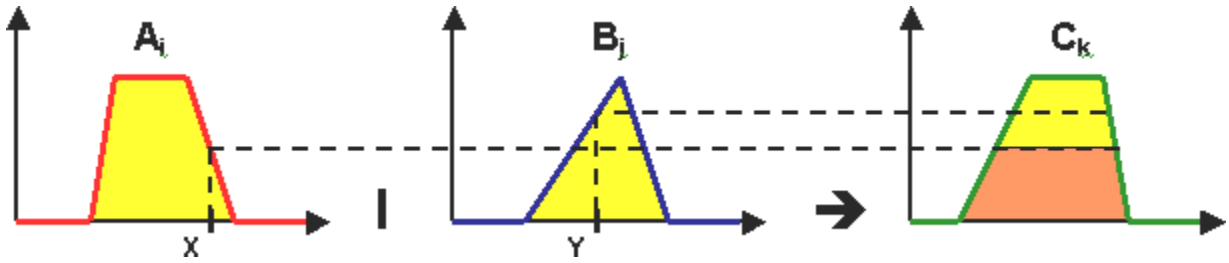
$$\alpha_2 = A_2(x_0) \wedge B_2(y_0),$$

де через "^" позначена операція логічного мінімуму.

3. Застосовуються всі наявні правила (точніше, всі, що можуть бути застосовані - тобто мають в лівій частині лінгвістичні змінні A і B, а в правій, заключній частині, змінну C.

Спочатку визначаються рівні 'відсікання' для лівої частини кожного з правил: для нашого прикладу це MP_n = min (Ai (x *), Bj (y *)) (для кожного правила P_n), а потім знаходяться 'усічені' функції приналежності для c : $C^*_k(z) = \min(M_{Pn}, C_k(z))$C * k (z) = min ($MP_n$, Ck (z)).

В результаті для кожного правила отримуємо своє уявлення для C:



Тобто знаходяться усічені функції приналежності:

$$C_1'(z) = (\alpha_1 \wedge C_1(z)),$$

$$C_2'(z) = (\alpha_2 \wedge C_2(z)).$$

4. Композиція, або об'єднання отриманих усічених функцій, для чого використовується максимальна композиція нечітких множин: MF (z) = max (C * k (z)), де MF (z) - функція належності підсумкового нечіткої множини.

Тобто використанням операції max (позначеної як "v") проводиться об'єднання знайдених усічених функцій, що призводить до отримання підсумкового нечіткого підмножини для змінної виходу з функцією приналежності:

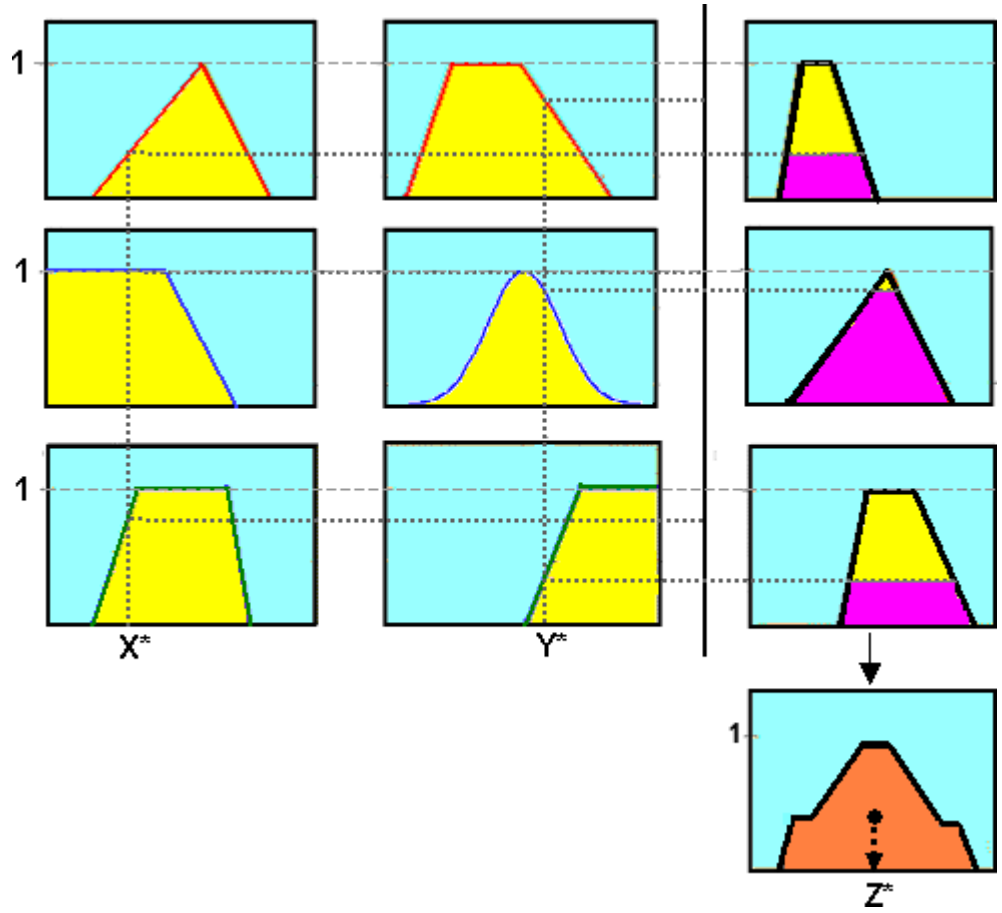
$$\mu_\Sigma(z) = C(z) = C_1'(z) \vee C_2'(z) = (\alpha_1 \wedge C_1(z)) \vee (\alpha_2 \wedge C_2(z)).$$

5. дефазифікації, або приведення до чіткості. Існує кілька методів дефазифікації. Наприклад, метод середнього центру, або центроїдний метод.

$$z_0 = \frac{\int_{\Omega} z\mu_{\Sigma}(z)dz}{\int_{\Omega} \mu_{\Sigma}(z)dz}.$$

У результаті виходить значення Z *:

Нижче на малюнку в графічному вигляді показаний нечіткий логічний висновок.

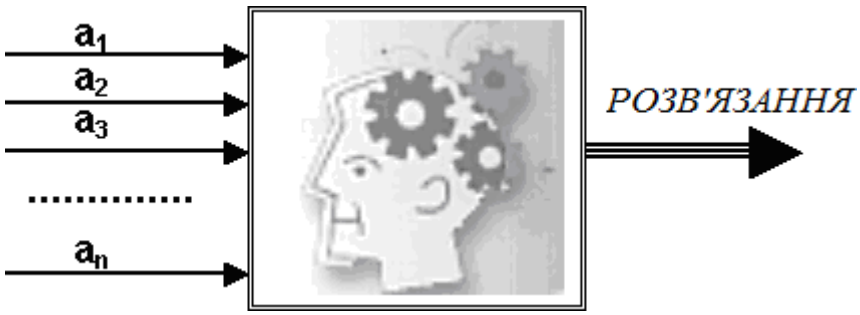


Повертаємося до питання «А чому б не вирішити цю задачу просто - помножити 350 на 3, отримати 1050 - і все!». Поки відповімо на нього так: а якщо прийдуть не 350, а 270 покупців, і хтось візьме 2 кг, а хтось 5? Якщо ми просто помножимо, то акумулюємо у відповіді досвід, укладений в понятті "середнє значення", тобто грунтуючись на середньостатистичних показниках. Так, цей відомий підхід має право на життя, але давно відомо, що він не кращий А в нашому випадку ми акумулюємо досвід, укладений в 3x3x3 правилах, що дає нам надію на більш якісне рішення важко формалізованих завдань.

Задамося питанням: коли застосування даного підходу обгрунтовано?

Ми зупинимося тільки на деяких позитивних моментах.

Уявімо собі систему прийняття рішень як деякий "чорний ящик" з безліччю входів (параметри для прийняття рішення) і одним виходом (результат обробки параметрів, власне рішення):



Якщо в "чорному ящику" використовується адекватна математична модель - дуже добре! Тоді питань немає! Хоча ...

Модель може бути і "не зовсім адекватною". Крім того, вона може бути досить складною. А що якщо рішення потрібно приймати в умовах обмеженості часу - скажімо, кожну мілісекунду (наприклад, в системах управління уприскуванням палива в двигуні внутрішнього згоряння)? А модель представляє із себе систему диференціальних рівнянь в частиних похідних, та такого порядку і складності, що вирішити її один раз потрібно цілу годину машинного часу!

Нарешті, моделі може і не бути. У цьому випадку часто використовують таблиці для перетворення вхідних параметрів у вихідний. Але таблиці дискретні, не можуть описати всі випадки, а якщо і описують (скажімо, з використанням інтервальних значень), то модель в цьому випадку виходить свідомо груба.

Закономірне питання: "А хто сказав, що використання нечіткої логіки виправдано?".

Є така людина. Право на застосування цього апарата нам дає теорема FAT (Fuzzy Approximation Theorem), доведена Бартоломеєм Коско (B. Kosko) в 1993 р Вона свідчить, що будь-яка математична система може бути апроксимована системою, заснованою на нечіткій логіці.

Теорія і практика нечіткої логіки сьогодні застосовується в автомобільній, аерокосмічній і транспортної промисловості, в області виробів побутової техніки, у сфері фінансів, аналізу та прийняття управлінських рішень та багатьох інших. Наприклад, в 1988 році експертна система на основі нечітких правил для прогнозування фінансових індикаторів єдина передбачила біржовий крах.

Гібридизація методів інтелектуальної обробки інформації - девіз, під яким пройшли 90-і роки у західних і американських дослідників. В результаті об'єднання декількох технологій штучного інтелекту з'явився спеціальний термін- 'м'які обчислення' (soft computing). В даний час м'які обчислення об'єднують такі області як: нечітка логіка, штучні нейронні мережі, імовірнісні міркування та еволюційні алгоритми. Вони доповнюють один одного і використовуються в різних комбінаціях для створення гібридних інтелектуальних систем.

Вплив нечіткої логіки виявилося, мабуть, самим великим. Подібно до того, як нечіткі множини розширили рамки класичної математичну теорію множин, нечітка логіка 'вторглася' практично в більшість методів Data Mining, наділивши їх новою функціональністю. Нижче наводяться найбільш цікаві приклади таких об'єднань.

Нечіткі контролери. У 1995-2000 роках системами з використанням нечіткої логіки штатно оснащувалися автомобілі Nissan, Mitsubishi і Honda, а BMW, Hyundai, Mazda, Mercedes і Peugeot планували впровадити такі системи. Нечіткі контролери застосовуються в АБС, системах управління двигуном внутрішнього згоряння (ДВЗ).

Вони знайшли масове застосування в побутовій апаратурі - відеокамери, пральні машини, мікрохвильові печі тощо

Нечіткі нейронні мережі. Нечіткі нейронні мережі (fuzzy-neural networks) здійснюють висновки на основі апарату нечіткої логіки, проте параметри функцій приналежності налаштовуються з використанням алгоритмів навчання НМ. Тому для підбору параметрів таких мереж застосуємо метод зворотного поширення помилки, спочатку запропонований для навчання багатошарового персептрона. Для цього модуль нечіткого управління представляється у формі багатошарової мережі. Нечітка нейронна мережа як правило складається з чотирьох шарів: шару фазифікація вхідних змінних, шару агрегування значень активації умови, шару агрегування нечітких правил і вихідного шару.

Найбільшого поширення в даний час отримали архітектури нечіткої НМ виду ANFIS і TSK. Доведено, що такі мережі є універсальними аппроксіматорамі.

Швидкі алгоритми навчання і интерпретируемость накопичених знань - ці фактори зробили сьогодні нечіткі нейронні мережі одним з найперспективніших і ефективних інструментів м'яких обчислень.

Адаптивні нечіткі системи. Класичні нечіткі системи володіють тим недоліком, що для формулювання правил і функцій приналежності необхідно залучати експертів тієї чи іншої предметної області, що не завжди вдається забезпечити. Адаптивні нечіткі системи (adaptive fuzzy systems) вирішують цю проблему. У таких системах підбір параметрів нечіткої системи виробляється в процесі навчання на експериментальних даних. Алгоритми навчання адаптивних нечітких систем щодо трудомісткі і складні в порівнянні з алгоритмами навчання нейронних мереж, і, як правило, складаються з двох стадій:

1. Генерація лінгвістичних правил;
2. Коригування функцій приналежності.

Перша задача відноситься до задачі переборного типу, друга - до оптимізації в безперервних просторах. При цьому виникає певне протиріччя: для генерації нечітких правил необхідні функції приналежності, а для проведення нечіткого виводу - правила. Крім того, при автоматичній генерації нечітких правил необхідно забезпечити їх повноту і несуперечність.

Значна частина методів навчання нечітких систем використовує генетичні алгоритми. В англомовній літературі цьому відповідає спеціальний термін - Genetic Fuzzy Systems.

Значний внесок у розвиток теорії і практики нечітких систем з еволюційної адаптацією внесла група іспанських дослідників на чолі з Ф. Херрера (F. Herrera).

Нечіткі запити. Нечіткі запити до баз даних (fuzzy queries) - перспективний напрямок в сучасних системах обробки інформації. Даний інструмент дає можливість формулювати запити на природній мові, наприклад: 'Вивести список недорогих пропозицій про винаймання житла близько до центру міста', що неможливо при використанні стандартного механізму запитів. Для цієї мети розроблено нечітку реляційна алгебра і спеціальні розширення мов SQL для нечітких запитів. Велика частина досліджень у цій області належить західноєвропейським вченим Д. Дюбуа і Г. Праде.

Нечіткі асоціативні правила. Нечіткі асоціативні правила (fuzzy associative rules) - інструмент для вилучення з баз даних закономірностей, які формулюються у вигляді лінгвістичних висловлювань. Тут введені спеціальні поняття нечіткої транзакції, підтримки та достовірності нечіткого асоціативного правила.

Нечіткі когнітивні карти. Когнітивна карта (від лат. Cognitio - знання, пізнання) - образ знайомого просторового оточення, або - ментальні репрезентації (подання) людини про просторову організацію навколишнього середовища.

Нечіткі когнітивні карти (fuzzy cognitive maps) були запропоновані Б. Коско в 1986 р і використовуються для моделювання причинних взаємозв'язків, виявлених між концептами деякій області. На відміну від простих когнітивних карт, нечіткі когнітивні карти являють собою нечіткий орієнтований граф, вузли якого є нечіткими множинами. Спрямовані ребра графа не тільки відображають причинно-наслідкові зв'язки між концептами, а й визначають ступінь впливу (вага) пов'язуються концептів. Активне використання нечітких когнітивних карт в якості засобу моделювання систем обумовлено можливістю наочного уявлення аналізованої системи і легкістю інтерпретації причинно- наслідкових зв'язків між концептами. Основні проблеми пов'язані з процесом побудови когнітивної карти, який не піддається формалізації. Крім того, необхідно довести, що побудована когнітивна карта адекватна реальній моделюється системі. Для вирішення даних проблем розроблені алгоритми автоматичного побудови когнітивних карт на основі вибірки даних.

Нечітка кластеризація. Нечіткі методи кластеризації, на відміну від чітких методів (наприклад, нейронні мережі Кохонена), дозволяють одному і тому ж об'єкту належати одночасно кільком кластерам, але з різним ступенем. Нечітка кластеризація в багатьох ситуаціях більш 'природна', ніж чітка, наприклад, для об'єктів, розташованих на кордоні кластерів. Найбільш

поширені: алгоритм нечіткої самоорганізації c-means і його узагальнення у вигляді алгоритму Густафсона-Кесселя.

СПИСОК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

1. **Керницький А., Лобур М., Врубель Є.** Штучний інтелект та еволюція інженерних баз даних // *Вісник Національного університету «Львівська політехніка»*. 2000. № 413. С. 161–164. URL: https://ena.lpnu.ua/handle/ntb/54795.
2. **Єфремов М. Ф., Єфремов Ю. М.** Штучний інтелект, історія та перспективи розвитку // *Вісник ЖДТУ. Серія «Технічні науки»*. 2008. № 2(45). С. 123–126. DOI: 10.26642/tn-2008-2(45)-123-126.
3. **Пальчевський Б. О., Шаповал О. М., Великий О. А.** Оптимізаційний синтез функціонально-модульної структури пакувального устаткування: монографія. Луцьк: РВВ Луцького НТУ, 2013. 165 с.
4. **Пальчевський Б. О., Крестьянполь О. А., Крестьянполь Л. Ю.** Інформаційні технології в проектуванні системи захисту пакованої продукції: монографія. Луцьк: Вежа-Друк, 2015. 160 с.
5. **Зайченко Ю. П., Гончар М. А.** Нечіткі методи кластерного аналізу в задачах автоматичної класифікації в економіці // *Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка*. 2007. № 47. С. 197–204.
6. **Зайченко Ю. П., Сидорук І. А.** Аналіз багатокритеріальної задачі оптимізації інвестиційного портфеля на основі прогнозування прибутковості акцій // *Вісник НТУУ «КПІ». Інформатика, управління та обчислювальна техніка*. 2015. Вип. 62. С. 79–88.
7. **Мова В. І., інші автори.** Інтелектуалізація комп'ютерів — проблеми та можливості // *Штучний інтелект*. 2006. № 3.
8. **Мова В. І., інші автори.** Знаково-орієнтовані робочі станції Інпарком // *Штучний інтелект*. 2009. № 1.
9. **Мова В. І., інші автори.** Інтелектуальний персональний комп'ютер гібридної архітектури // *Штучний інтелект*. 2012. № 3.

10. **Кизим М. О., Матюшенко І. Ю., Шостак І. В.** Перспективи розвитку інформаційно-комунікаційних технологій і штучного інтелекту в економіках країн світу та України. Харків: ІНЖЕК, 2012. 489 с.
11. **Глибовець М.М., Олецький О.В.** Системи штучного інтелекту. — К.: КМ Академія, 2002. — 366 с.
12. **Руденко О.Г., Бодянський Є.В.** Штучні нейронні мережі: Навчальний посібник. — Харків: ТОВ «Компанія СМІТ», 2006. — 404 с.
13. **Ситник В.Ф.** Системи підтримки прийняття рішень: Навч. посібник. – К.: КНЕУ, 2003.
14. **Ситник В.Ф., Краснюк М.Т.** Інтелектуальний аналіз даних (дейтамайнінг): Навч. посібник. – К.: КНЕУ, 2007. – 376 с.
15. **Субботін С.О.** Подання й обробка знань у системах штучного інтелекту та підтримки прийняття рішень. – Запоріжжя: ЗНТУ, 2008. — 341 с.
16. **Fox, J., South, M., Khan, O., Kennedy, C., Ashby, P., & Bechtel, J.** (2020). OpenClinical.net: Artificial intelligence and knowledge engineering at the point of care. *BMJ Health Care Inform, 27*(2), e100141. https://doi.org/10.1136/bmjhci-2020-100141.
17. **Ji, S., Pan, S., Cambria, E., Marttinen, P., & Yu, P. S.** (2020). A Survey on Knowledge Graphs: Representation, Acquisition and Applications. arXiv:2002.00388. Retrieved from https://arxiv.org/abs/2002.00388.
18. **Kitchin, D., & Vallati, M.** (Eds.). (2020). *Knowledge Engineering Tools and Techniques for AI Planning*. Springer. https://doi.org/10.1007/978-3-030-38561-3.
19. **Mohseni, S., Zarei, N., & Ragan, E. D.** (2018). A Multidisciplinary Survey and Framework for Design and Evaluation of Explainable AI Systems. arXiv:1811.11839. Retrieved from https://arxiv.org/abs/1811.11839.
20. **Salem, A. B. M. M., & Parusheva, S.** (2018). Exploiting the knowledge engineering paradigms for designing smart learning systems. *Eastern-*

European Journal of Enterprise Technologies, *2*(92), 38–44. https://doi.org/10.15587/1729-4061.2018.128410.

21. **Willard, J., Jia, X., Xu, S., Steinbach, M., & Kumar, V.** (2020). Integrating Scientific Knowledge with Machine Learning for Engineering and Environmental Systems. arXiv:2003.04919. Retrieved from https://arxiv.org/abs/2003.04919.
22. **Zhang, H., Khashabi, D., Song, Y., & Roth, D.** (2020). TransOMCS: From Linguistic Graphs to Commonsense Knowledge. arXiv:2005.00206. Retrieved from https://arxiv.org/abs/2005.00206.

Навчальний посібник

Методи та засоби інженерії знань для штучного інтелекту

Відповідальний за випуск : Г.О. Шліхта
Комп'ютерне верстання Г.О. Шліхта

Підп. до друку 29.09.2022р

Формат 60х84/16.
Ум. друк. арк. _____.
Обл.-вид. арк. _.
Тираж 20 пр.
Зам. №
Собівартість вид. грн к.

Видавець і виготовлювач : Рівненський державний гуманітарний університет