В. І. Месюра, Л. М. Ваховська

Основи проектування систем штучного інтелекту. Лабораторний практикум

Частина 1. Способи представлення задач і пошуку розв'язків

Міністерство освіти і науки України
Вінницький національний технічний університет

В. І. Месюра, Л. М. Ваховська

Основи проектування систем штучного інтелекту. Лабораторний практикум
Частина 1. Способи представлення задач і пошуку розв'язків

Затверджено Вченою радою Вінницького національного технічного університету як лабораторний практикум для студентів спеціальностей: "Інтелектуальні системи прийняття рішень" та "Програмне забезпечення автоматизованих систем". Протокол № 7 від 27 грудня 2007 року.

Вінниця ВНТУ 2009

УДК 519.68
М 56

Рецензенти:
А. М. Петух, доктор технічних наук професор
О. М. Роїк, доктор технічних наук професор
В. М. Лисогор, доктор технічних наук професор

Рекомендовано до видання Вченою Радою Вінницького національного технічного університету Міністерства освіти і науки України

Месюра В. І., Ваховська Л. М.
М 56 **Основи проектування систем штучного інтелекту. Лабораторний практикум. Частина 1. Способи представлення задач і пошуку розв'язків.** Лабораторний практикум. – Вінниця: ВНТУ, 2009. - 108 с.

Посібник містить теоретичний матеріал і приклади розв'язування практичних задач з подання та пошуку розв'язків інтелектуальних задач; цілі, практичні завдання, переліки контрольних питань та вимоги до знань студентів, необхідні для виконання лабораторних робіт, що відносяться до першого модуля навчальної дисципліни "Основи проектування систем штучного інтелекту"

Розрахований на студентів комп'ютерних спеціальностей всіх форм навчання.

УДК 519.68

ЗМІСТ

ВСТУП

Штучний інтелект є однією з найважливіших дисциплін в області інформатики, який має глибокі зв'язки з математикою, логікою, психологією, філософією, лінгвістикою і т. ін. Ця дисципліна визнана за самостійну з середини минулого сторіччя і з тих часів на неї витрачено чимало часу і коштів. У широкому сенсі, метою вивчення штучного інтелекту є створення машин для виконання завдань, з якими легко може впоратися людина, але які дуже не просто запрограмувати з використанням традиційного підходу до обчислень. Задачі, з якими легко впорається людина, але які є важкими для програмування, часто виявляються образливо буденними. Наприклад, ходіння кімнатою або підтримка розмови не вимагають від людини таких розумових зусиль, як обчислення добутку кількох чисел. Але створення машин, що виконують такі задачі, виявилося для інженерів проблемою величезної складності. Той факт, що відносно просто побудувати машину, здатну скласти університетський іспит, але дуже важко побудувати машину, здатну виконати якісне прибирання приміщення, кидає виклик традиційним уявленням про те, що значить бути інтелектуалом [1, 2].

У штучному інтелекту під наявністю задачі розуміють ситуацію, коли стан навколишнього світу відрізняється від бажаного (наприклад, людина знаходиться в одному місті, а їй треба потрапити у інше місто; відсутня інформація про „кредитну надійність" особи, але необхідно прийняти конкретне рішення відносно видачі їй кредиту; з наявної поточної ігрової ситуації гравець хоче потрапити у виграшну ситуацію і т. ін.). При цьому метою розв'язання задачі є усунення цієї відмінності за умов існування можливості виконання таких дій, що приведуть до необхідних змін. Але здійснення таких дій „всліпу" не вважається розв'язанням задачі, яке обов'язково має супроводжуватися певним процесом прийняття рішень щодо вибору адекватної дії на основі певного пошукового алгоритму [3].

Задача вважається добре визначеною, якщо для неї можна задати простір можливих рішень (станів), а також спосіб переглядання цього простору з метою пошуку кінцевого (цільового) стану, що відповідає вирішеній задачі. Пошук кінцевого стану задачі полягає у застосуванні до кожного стану певної алгоритмічної процедури з метою перевірки, чи не є він розв'язком задачі, і якщо ні, визначення серед доступних станів того, який скоріше за все є найближчим до цільового.

Людина зазвичай не вирішує задачу в тій формі, в якій вона спочатку формулюється. Вона прагне знайти таке подання задачі, яке полегшує її розуміння та розв'язання за рахунок скорочення простору, в якому необхідно виконувати пошук розв'язку. Етап вибору відповідної форми подання задачі є настільки буденним, що його важливість частіше за все навіть не усвідомлюється. Але, як правило, саме форма або спосіб подання задачі вирішальною мірою визначають успіх її розв'язання. При виборі способу

подання задачі зазвичай враховують дві обставини: подання задачі має достатньо точно моделювати реальність, а спосіб подання має бути зручним з точки зору застосування процедури розв'язання задачі [4].

Оскільки задачі в штучному інтелекті розв'язує комп'ютер, в даному лабораторному практикумі розглядаються способи подання задач, зручні з точки зору їх комп'ютерного розв'язання, найбільш поширеними серед яких є подання задач у просторі станів і подання, що дозволяє звести задачу до підзадач.

Дані подання і відповідні універсальні методи пошуку розв'язків розроблялися переважно на початкових етапах розвитку штучного інтелекту. Пізніше було помічено, що для вирішення багатьох практичних задач одних універсальних стратегій замало. Необхідний також великий об'єм знань і наявність практичного досвіду. Дослідження в області штучного інтелекту зосередилися на поданні і придбанні знань. Проте це не зменшило значущості розроблених стратегій пошуку розв'язків, оскільки вони являють собою певні базові схеми управління механізмами виведення систем, заснованих на знаннях. Способи подання задач і методи пошуку їх рішень, що розглядаються у лабораторному практикумі, відіграють важливу роль в багатьох системах штучного інтелекту, включаючи експертні системи, розуміння природної мови, доведення теорем і навчання.

Перша частина лабораторного практикуму містить необхідний теоретичний матеріал і велику кількість прикладів розв'язування практичних задач з подання та пошуку розв'язків інтелектуальних задач; цілі, практичні завдання, переліки контрольних питань та вимоги до знань студентів, необхідні для виконання лабораторних робіт, що віднесені, згідно з навчальною програмою, до першого модуля дисципліни "Основи проектування систем штучного інтелекту":

- лабораторна робота №1 дозволяє набути досвіду з подання інтелектуальних задач у просторі станів;
- лабораторна робота №2 надає навички пошуку розв'язків у просторі станів з використанням евристичної інформації;
- лабораторна робота №3 знайомить з методами подання та евристичного пошуку розв'язків задач, що розкладаються на підзадачі;
- лабораторна робота №4 висвітлює підходи до оптимізації пошуку виграшних ігрових стратегій.

1 ПОДАННЯ ЗАДАЧ У ПРОСТОРІ СТАНІВ

1.1 Загальне поняття задачі у штучному інтелекті

Істотною відмінністю задач, що розглядаються дисципліною штучного інтелекту, від тих, що розглядаються при вивченні інформатики і математики, є те, що ці задачі як правило не є „повністю визначеними", тобто чітко описаними спеціальною відповідною мовою, наприклад, мовою математики. Задачі штучного інтелекту як правило є наближеними до тих задач, з якими ми стикаємося в реальному житті, і які не є повністю визначеними. Частіше за все вони описуються природною мовою, яка з точки зору нормального опису має такі недоліки, як: неповнота, надмірність, неоднозначність і неточність [4, 5].

Поставити задачу означає перш за все зрозуміти її умову, вилучивши неповноту, надмірність і неоднозначність, або, іншими словами, знайти її відповідне подання. Задачу можна дійсно зрозуміти лише тоді, коли знайдено таке її подання, в якому всі елементи задачі подані без надмірності і багатозначності. При цьому з'являється можливість записати умову задачі у формалізованому вигляді, задача стає водночас і більш абстрактною і більш строгою. В цьому випадку говорять про задачу в замкнутій формі або про замкнуте формулювання задачі [4].

У найбільш загальному вигляді умови задачі з математичної точки зору можна сформулювати у замкненій форми таким чином:

знайти в заданій множині X елементи x,
що задовольняють множину заданих обмежень $K(x)$.

Прикладом такої задачі може служити задача розв'язання рівняння: «вибрати з множини x натуральних чисел такі, що задовольняють рівняння $3x+84=37x$».

Для однієї і тієї ж задачі завжди можна навести кілька різних замкнених формулювань. Не завжди при цьому існує одне найбільш „природне" (не стільки те, що саме напрошується, скільки найбільш просте) формулювання. Навіть якщо існує лише одне таке формулювання, його не завжди просто знайти. Часто саме виявлення такого формулювання стає головною проблемою при розв'язанні задачі [4].

Отже, одержане спочатку замкнене формулювання задачі, в загальному випадку, можна перевести в іншу форму, щоб з урахуванням обмежень $K(x)$ зменшити простір X і, завдяки цьому, поліпшити подання задачі. Процес розв'язання задачі за своєю суттю й є саме такою послідовністю зміни подань, останнє з яких і дає її безпосередній розв'язок.

1.2 Основні форми подання задач у замкненій формі

Існують два основних подання задачі у замкненій формі [6]:

1. Задані початковий S_0 і кінцевий S_1 стани, а також кінцевий перелік операторів *Oab*, що дозволяють перейти від стану S_a до стану S_b. Мова йде про знаходження шляху від S_0 до S_1. Наприклад, для гри "15", оператори *Oab* мають такий порядок: перехід з одного стану до іншого здійснюється послідовним переміщенням пронумерованих фішок на вільне місце. Замкнене формулювання задачі при цьому набуває такого вигляду:

X=(послідовність всіх можливих операторів) .

Розв'язком задачі буде певна послідовність операторів:

$$x=(Oab, Obc, Ocd, \ldots, Ofu),$$

де $S_a=S_1$ та $S_u=S_0$.

Множина обмежень $K(x)$ подає дозволені правила застосування операторів, згідно з якими, зокрема, кінцевий стан для попереднього оператора є початковим станом для наступного.

2. Класичне формулювання задачі в математиці: Одержати $C(x)$, знаючи $H(x)$.

Прикладом такої постановки задачі може бути такий:

показати, що для всіх n, таких що $n \in N$ N має місце $\sum_{i=1}^{n} i^3 = \left(\sum_{i=1}^{n} i\right)^2$.

Цей варіант можна звести до попереднього, якщо припустити $S_0=C(x)$ та $S_1=H(x)$. Однак, істотна відмінність полягає в тому, що в даному випадку не задані оператори переходу від одного стану до іншого. Мистецтво математика й полягає у знаходженні корисних для заданих умов задачі операторів. Можливі й такі випадки, коли кінцевий стан $C(x)$ не задано взагалі. При цьому задача може мати такий вигляд: "Розрахувати $\sum_{i=1}^{n} i^3$", внаслідок чого вона стає зовсім не визначеною, оскільки критерій зупинки процесу розв'язання задачі стає суб'єктивним. В даному випадку намагаються одержати в результаті розв'язання задачі вираз "більш простий" ніж початковий, хоча простота не є жорстко визначеним математичним поняттям.

1.3 Подання задачі у просторі станів

Одним з найбільш загальних підходів до формалізації задач при пошуку розв'язків є підхід, оснований на поданні задачі у просторі станів, згідно з яким задача описується множиною можливих станів (ситуацій, властивих даній проблемній області) та множиною можливих дій (операторів переходів між станами), які спричиняють перетворення однієї ситуації на іншу.

Таке подання є цілком природним, оскільки про існування задачі у науковому напрямку „штучний інтелект" йдеться у тому випадку, коли

стан навколишнього середовища відрізняється від бажаного. При цьому розв'язок задачі полягає в усунені існуючої відмінності шляхом виконання дій, які приводять до поступового перетворення початкової ситуації у цільову.

Для подання задачі у просторі станів необхідно виконати такі основні дії:

а) визначити сукупність ознак, за допомогою яких будуть описуватися стани системи;

б) вибрати форму опису станів;

в) навести опис початкового й поточних станів та визначити заборонені стани;

г) задати ознаки множини цільових станів, яких в загальному випадку може бути достатньо багато;

д) визначити оператори переходу з одного стану до іншого (множину можливих дій у даній проблемній області) і обмеження на застосування операторів (виконання дій).

Розглянемо, наприклад, задачу управління рухом автомобіля. Головний процес будь-якої задачі управління подається множиною ознак, що мають встановлюватись у певні значення шляхом відповідного управління (згідно з деяким набором правил), яке забезпечується певною множиною управляючих змінних. При цьому ознаки (розрізнювальні характеристики) стану задачі, повинні задовольняти такі вимоги:

- набувати конкретних значень (з певної множини) при досягненні цілі, тобто забезпечувати можливість оцінювання ступеня відповідності поточного стану цільовому;
- змінюватися в часі.

Не варто використовувати за ознаки стану значення характеристик, що не змінюються в часі, оскільки їх можна зберігати в окремій області пам'яті, не витрачаючи додаткові ресурси (пам'ять, час) на їхню обробку при кожному виконанні перетворення станів.

У задачі управління рухом автомобіля, головними вимогами є підтримка в заданих межах швидкості автомобіля і забезпечення безпеки руху, яке полягає в запобіганні від несанкціонованого виїзду на узбіччя та зіткнення з іншими об'єктами, присутніми на дорозі. Отож, у нашому випадку стани задачі можна, спрощено, оцінювати за значеннями таких основних ознак, як: «Швидкість руху», «Відстань до узбіччя» та «Відстань до перешкоди» (рис.1.1), хоча існує дуже велика кількість інших важливих ознак, наприклад, таких як «Технічний стан автомобіля», «Стан дорожнього покриття» і т. ін.

Природною формою опису стану $s_i \in S$ задачі буде, в нашому випадку, вектор (кортеж) з трьох координат, кожен з яких може мати певні обмеження на свої значення:

$$S = (V, D, U),$$

де V - швидкість руху: $0 < v < 120$ (км/год.);
D - відстань до перешкоди: $0{,}5 < d$ (м) ;
U - відстань до узбіччя: $0 < u < 10$ (м).

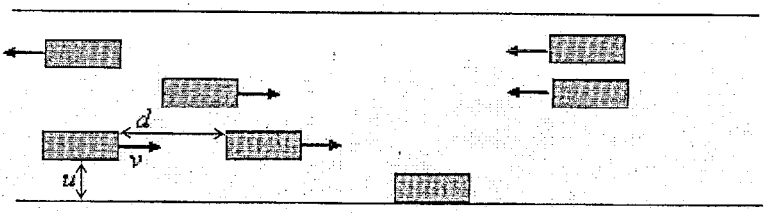


Рисунок 1.1 – Задача управління рухом автомобіля

Забороненою є множина $\bar{S}$ станів, в яких значення координат, або значення певних наборів значень координат вектора, виходять за встановлені межи. Наприклад, забороненою (аварійною) буде ситуація $\bar{s}_i = (v_i, 0, u_i)$, яка відповідає дотику автомобіля до стороннього об'єкта, або відображена на рис.1.1 ситуація подвійного обгону, оскільки вона порушує правила дорожнього руху.

Початковим станом задачі може бути стан $s_0 = (0, d_0, 0)$, або довільний не заборонений стан $s_i = (v_i, d_i, u_i)$, в якому вмикається система автоматичного управління.

Ознакою цільового стану в даному випадку будуть значення координат вектора, максимально наближені до оптимальних, з урахуванням поточної ситуації і набору правил управління (наприклад, за оптимальні значення можуть бути визначені такі: v=90, s=20, u=1).

Операторів переходу між станами (можливих дій управління) у нашому прикладі буде дев'ять, по три для кожної ознаки: *Ix* – збільшити значення ознаки *x*, *Zx* – не змінювати (підтримувати постійним) значення ознаки *x*, *Dx* – зменшити значення ознаки *x*.

Обмеження на застосування операторів визначаються правилами дорожнього руху та набором правил управління автомобілем, складених з урахуванням системи переваг особи, що приймає рішення. Наприклад, у стані $s_i = (100, 5, 2)$, що передує обгону, забороненим буде оператор *Du* зменшити відстань до узбіччя, виконання якого може призвести до зіткнення з перешкодою.

Одним з найбільш поширених засобів подання простору станів є графи [3]. При цьому вершини графа подають можливі стани проблемної області задачі, а дуги – окремі дії з розв'язування задачі, що приводять до переходу системи з одного стану до іншого. Один або декілька початкових

станів, що відповідають початковій інформації поставленої задачі, утворюють корінь дерева. Граф також містить одну або кілька цільових умов, що задовольняють розв'язок задачі. При цьому розв'язування задачі зводиться до процесу знаходження в графі шляху розв'язання (ланцюга, що веде до розв'язку), від початкового стану до цільового.

Отже, простір станів задачі можна подати четвіркою [1]:

$$PS = (N, A, S, G),$$

де N – множина можливих станів проблемної області задачі (вершин графа);

A – множина дуг між вершинами, що відповідає діям з розв'язання задачі;

$S_0 \subset N$– непуста множина початкових станів задачі;

$G \subset N$ – непуста множина, що складається з цільових станів, які можуть описуватися:

- значеннями ознак станів, що можна вимірювати в процесі пошуку;
- властивостями шляхів, що виникають в процесі пошуку, наприклад вартістю переміщення по дугах.

Розв'язком є шлях з вершини множини S_0 до вершини з множини G. Дуги у просторі станів відповідають діям процесу розв'язування, а шляхи подають розв'язки на різних стадіях завершення (часткові розв'язки). Шлях є метою пошуку. Він починається з початкового стану і продовжується доти, доки не буде виконано цільову умову. Породження нових станів уздовж шляху забезпечується операторами, що відповідають припустимим діям або правилам виведення.

Задача алгоритму пошуку полягає в знаходженні припустимого шляху у просторі станів. Алгоритми пошуку мають спрямовувати шляхи від початкової вершини до цільової, оскільки саме вони містять ланцюжок операцій, що веде до розв'язку задачі.

Одна із загальних особливостей пошуку на графах полягає у тому, що стани можуть бути досягнуті різними шляхами. Тому важливо обирати оптимальні шляхи. Крім того, множинні шляхи приводять до циклів і петель, яких треба уникати. Отож, алгоритми пошуку на графах повинні виявляти і усувати петлі на припустимих шляхах.

1.4 Графові подання у розв'язуванні задач

Граф складається з множини (необов'язково кінцевої) вершин, деякі пари з яких поєднуються за допомогою дуг. Якщо дуга направлена від вершини n_i до вершини n_j, то кажуть, що вершина n_i є батьківською для n_j, а вершина n_j є дочірньою для n_i. Якщо дві вершини є дочірніми одна до одної, то таку пару вершин називають іноді ребром графа. При поданні простору станів вершини графа описують стани, а дуги - оператори. Послідовність з k вершин, в якій кожна наступна вершина є дочірньою для попередньої, називається шляхом довжини k. Якщо існує шлях від n_i до n_j, то n_j

називають нащадком вершини n_i, чи вершиною, яка досяжна з n_i. Часто дугам графа приписують вартість, яка відображає вартість застосування відповідного оператора. В задачах оптимізації виникає необхідність відшукати шлях з мінімальною вартістю.

Граф може бути задано як у явному, так і у неявному вигляді. При явному заданні його вершини і дуги перераховуються явним чином (наприклад, у вигляді таблиці чи матриці). Така таблиця (матриця) може містити й відомості про вартість кожної з дуг. Очевидно, що для великих графів таке подання на практиці може бути неприйнятним, а для нескінчених - неможливим.

При неявному способі задання визначається деяка кінцева множина $\{s_i\}$ початкових вершин, а також задається оператор Γ, застосування якого до будь-якої вершини дозволяє отримати всі її дочірні вершини і вартості відповідних дуг. Послідовне застосування оператора Γ до елементів множини $\{s_i\}$ і їх дочірніх елементів дає подання для графа, який заданий неявно через Γ та $\{s_i\}$. При цьому процес пошуку в просторі станів послідовності операторів, що розв'язують задачу, відповідає перетворенню в явну форму деякої частини неявно заданого графа, такої, щоб до неї входила вершина, що відповідає меті.

Простір станів може бути поданий також за допомогою недетермінованих алгоритмів. Наприклад, процес породження простору станів можна подати блок-схемою, наведеною на рис.1.2 [3].

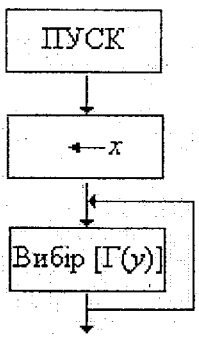


Рисунок 1.2 – Блок схема породження простору станів

На рис.1.2 через x та y позначені довільні сукупності даних. На початку роботи алгоритму змінна y вважається рівною вхідній структурі даних x, що описує початковий стан. Оператор присвоювання *ВИБІР* (встановити y таким, що дорівнює деякому члену множини $\Gamma(y)$ елементів, які безносередньо йдуть за y), є недетермінованим в тому розумінні, що при його виконанні може бути обрано будь-який член

множини $I(y)$. При цьому множина всіх можливих способів виконання програми (можливо нескінченна), охоплює весь простір станів. Програми, що використовують недетерміновані оператори, мають назву недетермінованих програм.

Поширене з урахуванням недетермінованості поняття оператора розгалуження, який позначається V, використовується для позначення розгалуження на n напрямків, яке подається n предикатами, $p_1(x,y)$ ... $p_n(x,y)$, що набувають значення "помилка" або "істина" в сумісній області зміни x і y, причому хоча б один з предикатів повинен мати значення "істина". Кожен предикат відповідає одній гілці. Оператор V обирає будь-яку одну гілку із значенням "істина". Очевидно, що звичайні присвоювання і розгалуження є окремими випадками недетермінованих.

1.5 Технологія подання задач у просторі станів

Розглянемо технологію подання задач у просторі станів на кількох прикладах, що відображають різноманітні проблемні області [4, 5].

Задача спрощення (перетворення) алгебраїчного виразу.

Спростити алгебраїчний вираз *(AB+CD)/BC*.

а) *вибір ознак для опису стану*. Очевидно, що опис стану має містити ідентифікатори змінних (констант) та алгебраїчних операцій, що входять до складу алгебраїчного виразу;

б) *форма опису стану*. Стан задачі можна подати, наприклад, двійковим деревом, некінцеві вершини якого відображають арифметичні знаки (+, -, x, /), а кінцеві - змінні, чи константні символи виразу. Перевагою такого опису є його наочність. Іншою часто використовуваною формою опису алгебраїчних виразів є лінійний рядок з префіксними операторами (наприклад, зворотний польський запис), перевагою якого є реалізація складних обчислень за один прохід програми;

в) *опис станів*. Приклад початкового стану у разі подання задачі перетворення алгебраїчного виразу двійковим деревом наведено на рис.1.3.

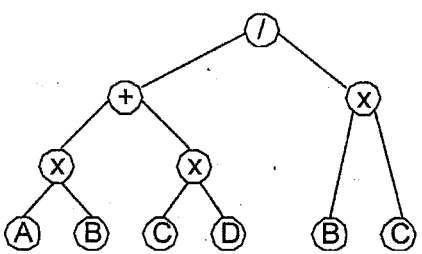


Рисунок 1.3 – Подання алгебраїчного виразу двійковим деревом

У разі подання даного виразу лінійним рядком з префіксними операторами, початковий стан набуде вигляду:

$$/+\times AB\times CD\times BC.$$

Забороненими є стани, які являють собою алгебраїчні вирази, записані з порушеннями загальноприйнятої нотації.

г) *опис цільового стану*. Ознакою цільового стану є будь яке дерево або рядок, що мають якнайменшу кількість компонентів алгебраїчного виразу (символів змінних, констант та алгебраїчних операцій), але не більшу за кількість компонентів початкового стану. Приклад двійкового дерева, що подає один з можливих цільових станів, наведено на рис.1.4.

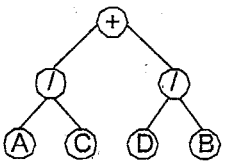


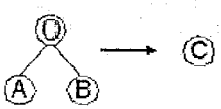
Рисунок 1.4 – Подання двійковим деревом цільового стану

Відповідний лінійний рядок набуде вигляду:

$$+/AC/DB.$$

д) *оператори переходів*. Являють собою множину алгебраїчних операцій. Обмеженнями на застосування операторів є, наприклад, невідповідність кількості аргументів даній алгебраїчній операції або вихід, внаслідок виконання операції, значень параметрів за встановлені межі і т. ін.

Загальне правило застосування оператора за умов подання станів двійковим деревом може бути записано так, як наведено на рис 1.5.



O – алгебраїчна операція (+, -, /, x, ...)
a, *в*, *c* – значення змінних або констант
$c = a$ O $в$ – результат операції над значеннями *a* і *в*

Рисунок 1.5 – Правило переходу між станами при поданні стану двійковим деревом

При описі станів у вигляді рядків найбільш зручним є спосіб подання операторних обчислень на основі правил переписування, відомих також як продукційні правила [2].

Множина правил переписування визначає можливі способи перетворення одного рядка в інший у вигляді $S_i \Rightarrow S_j$. Прикладом правила переписування може бути таке:

$$A\$ \Rightarrow B\$,$$

яке означає, що при появі символу A в першій позиції деякого рядка, його можна замінити на символ B. Знак \$ означає довільний підрядок, у тому числі й пустий.

Інші правила можуть мати, наприклад, такий вигляд:

1) $\$_1\$_2\$_3 \Rightarrow \$_1\$_2\$_2\$_3$ (будь-який підрядок може бути повторений);

2) $\$_1\$_2\$_2\$_3 \Rightarrow \$_1\$_2\$_3$ (один з розташованих поруч однакових підрядків може бути вилучений).

Наприклад, відповідно до першого правила, рядок ABC можна перетворити на рядок ABCABC (якщо, $\$_1$=Ø, $\$_2$=ABC, $\$_3$=Ø), ABCBC (якщо, $\$_1$=A, $\$_2$=BC, $\$_3$=Ø), ABABC (якщо, $\$_1$=Ø, $\$_2$=AB, $\$_3$=C) і т. ін. Тобто, до одного і того ж рядка правило переписування може застосовуватись декількома різними способами.

У нашому прикладі, множина операторів може бути задана таким чином:

$$\$_1 Oab\$_2 \Rightarrow \$_1 c\$_2,$$
$$\$_1 O_1 O_2\$_2 \Rightarrow \$_1 c^*\$_2,$$

де O – алгебраїчна операція (+, -, /, x, ...);

a, $в$, c – значення змінних або констант виразу;

$c = a\ O\ в$ – результат виконання операції O над значеннями a і $в$;

$c^* = a^*\ O\ в^*$ – результат виконання операції O над значеннями a^* і $в^*$ взятих зі стеку результатів попередніх операцій .

Задача про чотирьох шахових коней

У кутках шахової дошки розміром 3 × 3 розміщені чотири шахових коня, два з яких білі (БК), а два - чорні (ЧК), як показано на рисунку 1.6.

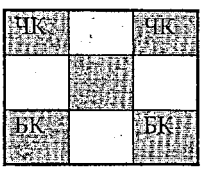


Рисунок 1.6 – Початкове розташування фігур на дошці у задачі про чотирьох шахових коней

Треба визначити мінімальну послідовність ходів, які дозволять білим коням зайняти місця чорних, і навпаки.

а) *вибір ознак для опису стану.* Очевидно, що опис стану має містити поточні місця розташування коней.

б) *форма опису стану.*

При першому знайомстві із задачею виникає бажання

промоделювати її шляхом переміщення коней на справжній шаховій дошці. Другою ідеєю стає, як правило, спроба формалізувати її в декартовій системі координат, що в даному випадку теж є не простою задачею (рис.1.7, а). Але якщо зрозуміти, що в задачі не є важливим моделювання фізичного розміщення коней на дошці, а головним є лише врахування взаємозв'язків між ходами при переході від однієї позиції до іншої, то задача значно спрощується.

Тобто, не важливо, як саме розташовані коні один відносно одного в кожній позиції, а важливо лише на які поля вони можуть потрапити за один хід. Такі поля "з точки зору" коня є для нього сусідніми. Якщо тепер пронумерувати поля дошки цифрами від 1 до 9, то сусідніми , наприклад, для поля 1, стануть поля 6 і 8 (рис.1.7, в). Розмірковуючи таким чином прийдемо до нового подання задачі, в якому дошка має форму кола і коні ходять по колу, пересуваючись за один хід на сусіднє поле (рис.1.7, в). Отже, поданням задачі є восьмирозрядний вектор, кожен розряд якого відображає відповідне поле дошки у порядка 1, 6, 7, 2, 9, 4, 3, 8, зі значеннями координат з множини M:

$$M = \{0, b, w\},$$

де 0 – відображає не зайняте поле;

b – поле зайняте білим конем;

w – поле зайняте чорним конем.

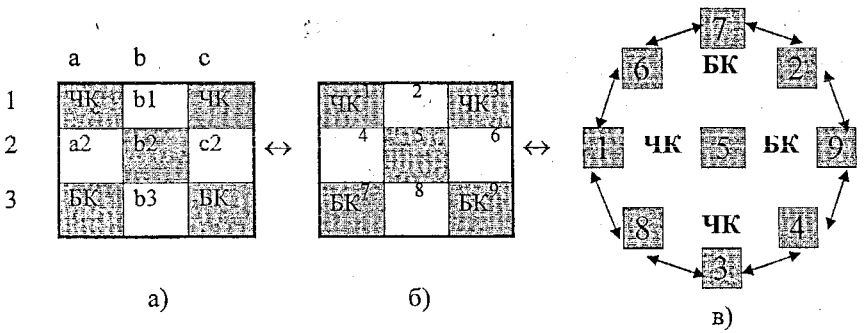


Рисунок 1.7 – Зміни подання задачі про чотирьох коней

в) *опис станів*. Початковий стан подається вектором

$$s_0 = (b, 0, w, 0, w, 0, b, 0).$$

За умов виконання правил гри, заборонені стани в даному описі відсутні.

г) *опис цільового стану*. Цільовий стан подається вектором

$$s_g = (w, 0, b, 0, b, 0, w, 0).$$

д) *оператори переходів*. Відповідають переходам коней у сусідні позиції, з урахуванням можливості переходів між першим і останнім розрядом вектора:

$$\$_1 h 0 \$_2 \Rightarrow \$_1 0 h \$_2;$$
$$\$_1 0 h \$_2 \Rightarrow \$_1 h 0 \$_2,$$

де h набуває значення з множини $\{b, w\}$.

Будь-які обмеження на застосування операторів у вибраному поданні відсутні.

Зауважимо, що розв'язання задачі при такому поданні стає очевидним, необхідно лише повернути коло, яке графічно подає початковий простір станів, на 180° (циклічно зсунути початковий вектор на чотири розряди вправо). Таке повертання відповідає чотирьом ходам кожного коня, тобто мінімальна кількість ходів необхідна для розв'язання задачі складає шістнадцять.

Задача комівояжера

Класична комбінаторна задача. Необхідно прокласти маршрут таким чином, щоб побувавши в кожному місті один раз повернутись до вихідного пункту. Бажано, щоб маршрут мав мінімальну довжину (задача має ефективне вирішення для 50 міст, добре - для 200, лише наближене – для більшої кількості). Комівояжеру треба відвідати кожне з п'яти міст, зображених на рис.1.8.

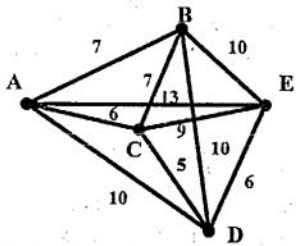


Рисунок 1.8 – Графічне подання задачі комівояжера

Між кожною парою міст існує шлях, довжину якого вказано на рисунку. Необхідно знайти найкоротший шлях, який дозволить комівояжеру, виїхавши з міста А, по одному разу проїхати через кожне місто і повернутися до пункту відправлення А.

а) *вибір ознак для опису станів*. Як ознаки необхідно вибрати такі характеристики задачі, які у кожну мить часу надають нам її „моментальне фото" (поточний стан її частинного розв'язку), що дозволяє судити про те, яку частину задачі на дану мить вже розв'язано, і якими можуть бути шляхи продовження розв'язування задачі. Оскільки задачею комівояжера є відвідання певної кількості міст, то у кожну мить часу нам треба мати інформацію про те, які саме міста ним вже відвідані.

Отже, ознаками поточного стану розв'язку задачі має стати список міст, в яких вже побував комівояжер.

б) *форма опису станів.* Список відвіданих міст можна природно подавати відповідним рядком $ символів (ідентифікаторів міст), наприклад $=AC, $=ABE і т. ін, або вектором (a,b,c,d,e), кожен розряд якого подає одне з міст, причому значення розряду 0 позначає місто, до якого комівояжеру ще слід прибути, а значення 1 – місто, вже відвідане на дану мить комівояжером. У першому випадку кожен крок розв'язання задачі буде супроводжуватися додаванням до рядка нового символу, а у другому - встановленням значення 1 у відповідному розряді вектора.

Зауважимо, що перший варіант є більш інформативним, оскільки містить у собі інформацію не тільки про відвідані міста, але й про використані при цьому шляхи, що надасть в подальшому можливість обчислити у просторі станів оптимальний шлях на основі таблиць відстаней між містами.

Для забезпечення можливості врахування довжини подоланих шляхів у векторному поданні, здійснимо його модифікацію: до кожного з розрядів вектора будемо замість одиниці заносити значення довжини шляху, який з'єднує відповідне даному розряду місто з тим містом, з якого до нього прибув комівояжер. При цьому для обчислення загальної довжини шляхів кожної з мандрівок комівояжера, достатньо буде підсумувати значення усіх розрядів відповідного вектора.

в) *опис станів.* При поданні станів символьними рядками початковий стан набуде вигляду рядка з одного символу – ідентифікатора міста початку подорожі, у нашому прикладі:

$$s_0 = A.$$

У разі векторного подання, початковий стан відобразиться вектором:

$$s_0 = (0, 0, 0, 0, 0).$$

Зауважимо, що початковий вектор завжди містить лише нулі, незалежно від міста початку подорожі, оскільки ми відмічаємо значущими цифрами не ті міста, які вже відвідав комівояжер, а ті – до яких йому ще належить прибути.

До заборонених станів будуть віднесені рядки, в яких будь-який символ повторюється більше одного разу, за винятком того, що останнім в списку може знов бути символ А.

г) *опис цільового стану.* Будь-який рядок, що розпочинається і закінчується символом А і містить у собі усі інші символи, що зустрічаються рівно по одному разу, є цільовим станом.

Вектор (1,1,1,1,1) є „передцільовим" станом і потребує виконання оператору переходу до початкового міста *A*.

д) *оператори переходів.* Оператор відповідає переїзду комівояжера з міста *X* до наступного міста *Y*:

$$\$_1 X \xrightarrow{Y} \$_1 XY.$$

Оператор є забороненим, якщо не існує прямого шляху між містами X і Y, або якщо він призводить до неприпустимого опису стану. Наприклад, при початку подорожі комівояжера з міста A, оператор

$$\$_1 X \xrightarrow{Y} \$_1 XA$$

не може бути застосований до жодного рядка, що не містить у собі ідентифікатори усіх міст, або до вектора з більш ніж одним нульовим розрядом.

На рис. 1.9 наведено графічне відображення запропонованого подання простору станів для задачі комівояжера у вигляді графа:

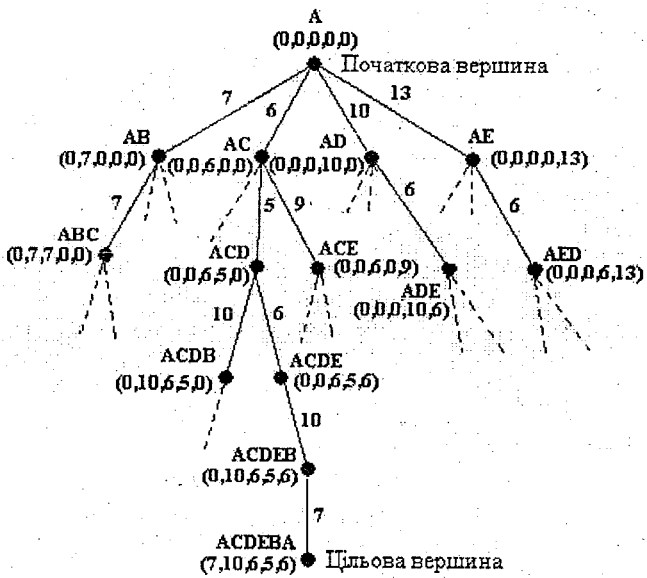


Рисунок 1.9 – Простір станів задачі комівояжера

1.6 Використання змінних для опису станів

Вибір певного подання задачі в просторі станів суттєво впливає на пошукові зусилля, необхідні для її розв'язання. Очевидно, що більш бажаними є подання, які характеризуються простором станів з меншою потужністю (графи яких мають менш кількість вершин). Існує багато задач, які здаються дуже важкими, але при правильному подані яких, потужності відповідних просторів станів стають дуже малими. Іноді, шляхом відкиданням одних операторів за непотрібністю та об'єднанням

інших операторів до більш потужних, простір станів вдається «ущільнити» до точки. Якщо ж такі прості перетворення виявляються не здійснюваними, менший простір може забезпечити повне переформулювання задачі, що змінює саме поняття стану.

Надзвичайно важливою ідеєю в області подання задач є використання в описах станів змінних, які можуть описувати цілу множину станів замість одного. Описи конкретних станів отримують при цьому підстановкою на місця таких змінних конкретних значень. Вираз, що містить змінні, використовувані для опису множини станів у вказаному значенні, називається схемою для опису станів. Проілюструємо використання схем для опису станів на прикладі задачі про мавпу і банани [7].

Задача про мавпу і банани

В кімнаті, де знаходиться мавпа, є ящик і низка бананів, які закріплені до стелі так високо, що мавпа не може дістати їх, знаходячись на підлозі. Треба визначити послідовність дій, яка допоможе мавпі дістати банани.

а) *вибір ознак для опису станів*. Очевидно, що в описі станів задачі мають з'явитися такі їх ознаки, як: координати мавпи у кімнаті (вертикальні та горизонтальні); координати ящика та наявність бананів у мавпи. Координати бананів не варто включати до опису станів, оскільки вони не змінюються у часі і можуть зберігатися у пам'яті комп'ютера як константне значення.

б) *форма опису станів*. Очевидно, в описі станів повинні з'явитися такі елементи, як: координати мавпи у кімнаті (вертикальні та горизонтальні); координати ящика та наявність бананів у мавпи. У зв'язку з необхідністю подання значень кінцевої кількості ознак (постійної протягом розв'язання задачі), доцільно скористатись векторною формою опису станів.

в) *опис станів*. Подамо опис станів чотирирозрядним вектором:

$$(w, x, y, z),$$

де w - двовимірний вектор координат мавпи на горизонтальній площині;

$x = \{0,1\}$ – вертикальні координати мавпи: 0 – розташування мавпи на підлозі, 1 – розташування мавпи на ящику;

y – двовимірні координати ящику;

$z = \{0,1\}$ наявність/відсутність у мавпи банана.

Якби кожне значення кортежу (w, x, y, z) описувало один стан, то їх було б нескінченно багато. Навіть якщо накласти деяку сітку на підлогу (зробивши можливим знаходження предметів тільки у кінцевій кількості точок), прийшлось би мати справу з дуже великим простором станів. Використання схеми опису станів допомагає подолати цю проблему.

г) *опис цільового стану*. Будь-який вектор з одиничним значенням

останнього розряду є цільовим станом.

д) *оператори переходів*. Відповідають у задачі чотирьом можливим діям мавпи:

підійти (*u*) - мавпа пересувається до точки *u* у площині підлоги;

пересунути (*v*) - мавпа пересуває ящик до точки *v* підлоги кімнати;

залізти (на ящик);

схопити (банани).

У зв'язку з присутністю змінних, оператори *підійти* і *пересунути* фактично є схемами. Умови застосування і дії цих операторів, задаються такими правилами переписування:

$$(w, 0, y, z) \xrightarrow{підійти(u)} (u, 0, y, z)$$

$$(w, 0, w, z) \xrightarrow{пересунути(v)} (v, 0, v, z)$$

$$(w, 0, w, z) \xrightarrow{залізти} (w, 1, w, z)$$

$$(c, 1, c, 0) \xrightarrow{схопити} (c, 1, c, 1),$$

де c - координата точки підлоги, яка розташована безпосередньо під бананами.

Побудуємо простір станів. Нехай спочатку мавпа знаходиться в точці *a*, а ящик в точці *b*. Тоді початковим станом буде (a,0,b,0). Єдиним оператором, який можна застосувати у цій ситуації (значення першої і третьої координат вектора не збігаються між собою) є *підійти*(*u*), що приводить до схеми (u,0,b,0). Тепер можна застосувати три оператори. Якщо $u = b$, то мавпа може чи то залізти на ящик, чи то пересунути його. Крім того, незалежно від значення *u* мавпа може перейти в будь-яку іншу точку підлоги. Якщо мавпа вилізе на ящик, це приведе до стану (b,1,b,0), пересування ящика в точку *v* - до схеми (v,0,v,0), а перехід у інше місце, що описується новою змінною, не змінює опису. Отриманий граф простору станів (рис.1.10) є невеликим і в ньому не важко відшукати шлях розв'язання задачі. Підстановкою на місця змінних відповідних окремих значень, отримаємо шукану послідовність операторів: *підійти (b)*, *пересунути (c)*, *залізти*, *схопити*.

1.7 Програмна реалізація продукційної системи для побудови простору станів

Продукцією називають правило, що складається з двох частин, одна з яких пов'язана з розпізнаванням ситуації, а друга - з певною дією. Тобто, продукція це пара "ситуація-дія", ліва частина якої містить сукупність ознак, які слід шукати, а права - список того, що треба зробити.

При використанні продукцій в дедуктивних системах ситуації, які впливають на вибір продукції, є комбінаціями певних фактів. При цьому

дії подають ствердженням відносно цих фактів, і виводяться безпосередньо з самих цих комбінацій фактів. Отже, в такому випадку продукції подають не пари "ситуація-дія", а пари "передумова - висновок" і подаються логічною операцією імплікації [5, 6].

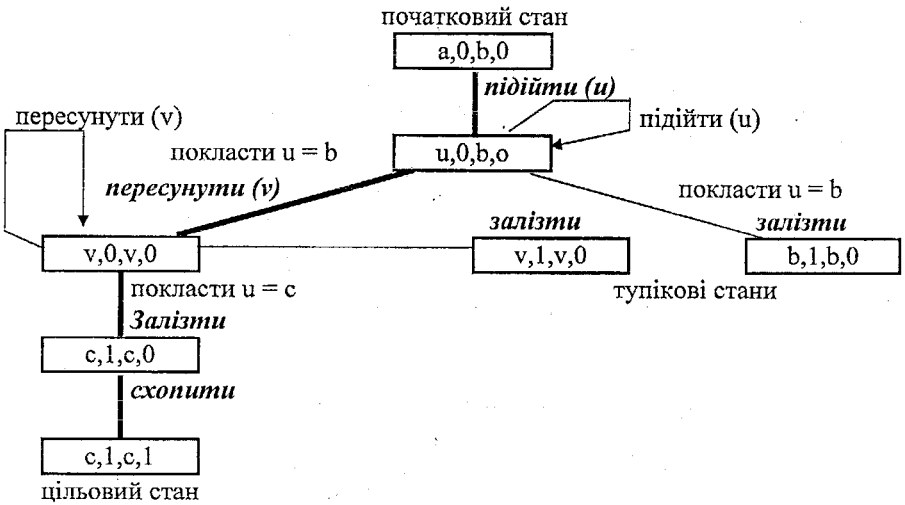


Рисунок 1.10 – Простір станів для задачі про мавпу і банани

На понятті "формальна система продукцій" базується велика кількість систем штучного інтелекту. Вперше продукційні системи були запропоновані у 1943 р. Емілем Постом, який довів, що вони є логічними системами, еквівалентними машині Т'юрінга, тобто, що вони є універсальними. Це означає, що будь-яка система, яка оперує символами, може бути реалізована в одній з продукційних систем Поста.

Система продукцій Поста задається за допомогою свого алфавіту

$$C = \{c_1, ..., c_p\}$$

і системи базисних продукцій

$$x_i W \rightarrow W y_i \ (i = 1, ..., l),$$

де x_i, y_i - слова в алфавіті С.

Нехай деяке слово С розпочинається з підрядка x_i. Застосувати до C продукцію $x_i W \rightarrow W y_i$ означає викреслити з C початковий підрядок x_i, і приписати до слова, яке залишилося, підрядок y_i. Наприклад, застосувавши до слова *aba* продукцію $abW \rightarrow Wc$, ми отримаємо слово *ac*.

Кожна система продукцій може розглядатись як формальна система з правилами виведення p_i $(i = 1, .., l)$, де $p(D,C)$ вважається істинним

(придатним) якщо слово C буде одержано з D за допомогою продукції $x_iW \rightarrow Wy_i$.

Якщо на набір впорядкованих продукцій накласти неявну структуру управління, то буде здійснено перехід до поняття нормального алгоритму Маркова, в якому впорядковані продукції (формули підстановок) застосовуються до деякого заданого слова. Продукції перевіряються на можливість застосування до заданого слова одна за одною, згідно із заданим порядком. При виявленні першої ж придатної продукції вона застосовується до слова і змінює його згідно з відповідним правилом виведення. Далі процес перевірки можливості застосування продукцій продовжується, розпочинаючи з продукції, що має найвищий пріоритет. Такий цикл "перевірка - виконання" продовжується до того часу, доки не буде знайдено жодної продукції, яку можна було б застосувати до відповідного слова, або не буде застосовано таку продукцію, яка відмічена як завершальна.

Психологічні дослідження процесів прийняття рішень людиною, проведені на початку семидесятих років XX сторіччя американськими вченими Ньюелом і Саймоном показали, що людина в своїх міркуваннях використовує правила подібні до продукцій, тобто правила вигляду "умова - дія". Ньюел запропонував використовувати продукційні системи для комп'ютерного моделювання процесів прийняття рішень [3].

Згідно з пропозиціями Ньюела, застосування продукційної системи забезпечує чітке розмежування між такими стандартними обчислювальними компонентами як дані, операції та управління. При цьому продукційну систему можна визначити трійкою:

$$P = < M, K, I >,$$

де M – така центральна структура, як глобальна база даних (робоча пам'ять);

K - база знань (множина чітко визначених правил вигляду "умова - дія");

I - інтерпретатор, який реалізує процедури виведення за допомогою деякої глобальної стратегії управління.

Головна відмінність продукційних систем від традиційних полягає в доступності глобальної бази даних усім правилам продукцій. Одні правила не викликають інші, як це робиться у традиційних програмних системах, а зв'язок між ними здійснюється виключно через глобальну базу даних. Якщо доповнення чи зміни в базі знань традиційної програми можуть вимагати внесення великих змін у програмі та вже існуючих структурах даних, то продукційні системи є значно більш модульними і допускають відносно незалежне внесення змін до бази даних, системи управління або правил.

У загальному випадку, для розв'язання задачі за допомогою системи продукцій треба визначити глобальну базу даних, набір правил та задати

стратегію управління, тобто вирішити проблему подання задачі. Наприклад, для гри у шахи ними будуть, відповідно: стани, ходи та мета задачі. Кожна конфігурація фігур у грі є станом задачі. Множина всіх можливих конфігурацій утворює простір станів, який для гри у шахи є надзвичайно великим і складає близько 10^{120}, перевищуючи кількість молекул у Всесвіті [8]. Для опису станів можна обрати матрицю розміром 8×8. Початкова глобальна база даних буде описом початкового стану.

Кожний хід перетворює один стан на інший і визначається правилом продукції, яке відповідним чином застосовується до описів станів. Кожне правило має попередню умову, яку повинен задовольняти опис стану для того, щоб правило можна було застосувати до нього. Наприклад, попередньою умовою для правила, яке відповідає за переміщення короля у певне поле шахової дошки, є вимога, щоб це поле не знаходилося під „обстрілом" ворожої фігури.

Узагальнений алгоритм управління системою продукцій для побудови простору станів задачі можна записати у такому вигляді [3]:

```
Procedure Production
Begin
    Занести до глобальної бази даних DATA опис початкового стану;
    Until DATA задовольняє термінальну умову, Do
    Begin
        Обрати деяке правило П з множини правил, які можна
        застосувати до глобальної бази даних DATA;
        DATA:= результат застосування П до DATA;
    End
End.
```

1.8 Контрольні питання

1. Наведіть Ваше особисте означення поняття „інтелект".
2. Що розуміється під задачею в штучному інтелекті?
3. У чому, на Ваш погляд, полягають основні відмінності між навчальними задачами та задачами, які доводиться розв'язувати у реальному житті?
4. Наведіть п'ять власних прикладів задач, які важко формалізувати.
5. Що означає термін „поставити задачу"?
6. Що розуміється під поданням задачі?
7. Наведіть власний приклад задачі, ізоморфне перетворення якої значно полегшує її розв'язання.
8. Поясніть поняття формулювання замкненої задачі.
9. Які основні форми подання задач Ви знаєте?
10. Наведіть приклади різних форм подання опису станів.
11. На власному прикладі поясніть поняття інфіксної та префіксної форм

запису алгебраїчних виразів.

12. Що необхідно зробити для подання задачі у просторі станів?
13. Поясніть на прикладі, як саме Ви будете обирати ознаки для подання задачі у просторі станів.
14. Поясніть відмінності та наведіть приклади доцільності використання векторного та рядкового опису станів.
15. Наведіть конкретний приклад формального опису оператора переходу між станами.
16. Дайте означення поняття „дерево" через поняття „граф", та наведіть приклад подання деревом простору станів будь-якої задачі.
17. Що таке „недетермінований алгоритм"? Нарисуйте схему недетермінованого алгоритму для гри „8".
18. Наведіть основні терміни, переваги та недоліки подання задач за допомогою графів.
19. Що таке схема опису станів?
20. Наведіть власні приклади задач, які недоцільно або неможливо подати у просторі станів. Поясніть чому саме.
21. Поясніть, що означає термін „продукційна система".

ЛАБОРАТОРНА РОБОТА №1
ПОДАННЯ ЗАДАЧ У ПРОСТОРІ СТАНІВ

Мета роботи

Закріплення теоретичного матеріалу та надбання практичних навичок з формалізації подання інтелектуальних задач у просторі станів і створення відповідного програмного забезпечення.

У результаті виконання лабораторної роботи студент повинен вміти:

- визначати найбільш інформативні ознаки об'єкта, явища, процесу для подання задачі;
- здійснювати обґрунтований вибір форми подання задачі;
- будувати оператори переходів між станами з урахуванням обмежень на їх застосування;
- визначати ознаки цільової ситуації;
- створювати програмні моделі продукційних систем.

Завдання на підготовку

Студент повинен знати:

- основні поняття штучного інтелекту;
- поняття та особливості інтелектуальної задачі;
- форми та методи подання інтелектуальних задач;
- методологію подання задачі у просторі станів;
- мови програмування LISP, PROLOG, Pascal, C++.

Студент повинен вміти:

- створювати програми мовами LISP, PROLOG, Pascal, C++.

Для допуску до виконання роботи необхідно:

- вміти відповісти на теоретичні питання за ходом виконання роботи;
- показати викладачу заготівку звіту про лабораторну роботу, яка повинна містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1. Здійснити подання у просторі станів узгодженої з викладачем інтелектуальної задачі.

Перевага надається самостійному формулюванню студентом інтелектуальної задачі, що відповідає тематиці його наукової діяльності, курсової або бакалаврської роботи, області інтересів студента і. т. ін. У разі неспроможності самостійного формулювання, задача вибирається з наведеного переліку завдань до лабораторної роботи.

2. Написати програму автоматичної побудови простору стану, узгодженою з викладачем мовою програмування.

Порядок виконання лабораторної роботи

1. Для вибраної задачі:
- визначити найбільш інформативні ознаки, які будуть використовуватися у поданні станів;
- обгрунтувати вибір форми подання станів;
- подати у вибраній формі початковий стан задачі;
- визначити перелік операторів переходів між станами:
- для кожного оператора побудувати:
  - обмеження на його застосування;
  - шаблон частини ЯКЩО оператора переходу;
  - шаблон частини ТО оператора переходу;
- сформулювати ознаки цільових станів.

2. Графічно відобразити подання задачі у просторі станів у вигляді відповідного графа з відображенням формального подання станів і ідентифікаторів операторів переходів (припускається відображення фрагмента простору станів, в межах якого повинні бути відображені приклади застосування кожного з операторів переходу між станами).

3. Написати програму для моделювання обраної форми подання задачі у просторі станів, яка повинна забезпечувати можливості:
- перегляду бази знань (операторів переходів між станами)
- покрокового виведення на екран кожного наступного стану в процесі формування простору станів з відображенням ідентифікатора застосованого оператора (продукційного правила);
- автоматичного формування простору станів.

Зміст звіту

Теоретична частина

1. Опис умов задачі.
2. Обгрунтування вибору значущих параметрів.
3. Обгрунтування вибору форми подання станів.
4. Формальний опис початкового стану.
5. Формальний опис ознак цільової ситуації.
6. Формальний опис операторів переходів між станами.
7. Фрагмент простору станів з використанням кожного оператора переходів (правила з бази знань).
8. Структурна схема продукційної системи для побудови простору станів.

Практична частина

1. Схема алгоритму програми побудови простору станів.

2. Приклад роботи програми, проілюстрований скріншотами.
3. Інструкція користувача для роботи з програмою.
4. Лістінг програми з коментарями.

Приклади завдань на лабораторну роботу

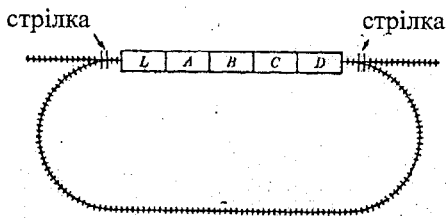
Подайте у просторі станів такі задачі.

1. Задано граматику, реченнями якої є рядки одного з таких видів:
- за символом *a* наступним є символ *b*;
- за символом *a* наступним є деяке речення;
- за деяким реченням наступним є символ *b*;
- за деяким реченням наступним є інше речення.

При такому визначенні формальної мови прикладами її речень можуть бути : *aab*, *abaabab*, *aaaaab* і т. ін. Не є реченнями граматики рядки: *aaa*, *aba*, *abaa*, і т. ін.

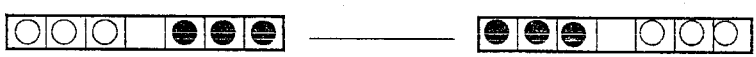
Необхідно визначити, чи є заданий на вході рядок реченням даної граматики.

2. Локомотив *L* і вагони стоять на залізничному шляху, наведеному нижче, у порядка (зліва направо). Локомотив можна довільно відчіпляти і зчіпляти з окремими вагонами, стрілки можуть набувати довільного положення і локомотив може тягнути або штовхати вагон, до якого він причеплений. Необхідно визначити усі можливі розташування вагонів і локомотива на прямому відрізку залізничного шляху.



3. Подайте у просторі станів задачу про лицарів та зброєносців. Три лицарі зі зброєносцями мають переправитися через річку на човні, який бере не більше двох осіб. Треба здійснити переправу так, щоб на кожному з берегів і в човні ніхто зі зброєносців не знаходився разом з чужими лицарями без свого хазяїна.

4. Задано і цільову позиції у грі, що має такі правила:

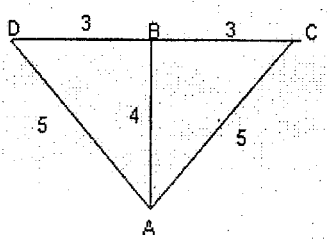


- шашку можна пересувати на сусіднє поле;
- дозволяється переступати через сусідню шашку, якщо за нею є вільне

поле; - білі і чорні шашки повинні просуватись назустріч без поверненъ назад.

Необхідно перетворити початкову позицію на цільову.

5. Дано два джерела води A і B. З A витікає 1000 літрів води за хвилину, з B - 500 літрів. Джерела повинні забезпечити водою два басейни C і D, - потреба кожного з яких складає 750 літрів на хвилину. Вода може подаватися по трубах з максимальною пропускною здатністю 750 літрів за хвилину. Джерела і басейни розташовані так, як показано на рисунку. Як треба з'єднати труби, щоб їх загальна довжина була найменшою?



6. Загін солдат підходить до ріки, через яку необхідно переправитись. Міст зламано, але на човні вздовж берега катаються двоє хлопчаків. В човні може переправитися лише один солдат або два хлопчики і не більше. Необхідно забезпечити переправу всіх солдат через ріку.

7. В квадраті, який має 16 комірок, треба розмістити 16 літер (4 літери A, 4 літери B, 4 літери C, 4 літери E) так, щоб на кожній вертикалі і на кожній горизонталі будь-яка літера зустрічалась тільки один раз.

8. Дано два глечики: перший - з водою ($V = 5$л), другий - пустий ($V = 2$л). Необхідно отримати рівно 1л в глечику ємністю 2л. Воду можна виливати і переливати з одного глечику до іншого.

9. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів.

10. Три лицарі і три зброєносці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі або у човні зброєносець зали-

шиться без свого лицаря, його буде вбито.

11. Фермер, вовк, козел і капуста знаходяться на одному березі річки. Їм треба переправитись на інший берег, але вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, не можна залишати на одному березі сам на сам козу і капусту та козу і вовка.

12. Необхідно розташувати на шаховій дошці вісім ферзів так, щоб ні один з них не знаходився під ударом іншого.

13. Є три стержні на одному з яких знаходяться три диски в порядку зменшення їх розмірів знизу до верху. Необхідно перекласти диски з першого стержня на третій враховуючи такі правила:

- на кожному ході один з дисків повинен переміщуватись з одного стержня на інший;
- диск більшого діаметра заборонено класти на диск меншого діаметра.

14. По колу розташовані 5 лунок. На початку гри одна лунка, яка має назву "Рума" є пустою, а кожна з інших лунок містить по 2 кульки. На кожному ході всі кульки, що знаходяться в деякій лунці А, розкладаються по одній в сусідні лунки за ходом годинникової стрілки. Перший хід робиться з будь-якої лунки. Якщо при черговому ході остання кулька потрапляє до руми, то наступний хід можна зробити з будь-якої лунки крім руми, а якщо не потрапляє - то наступним ходом розподіляються кульки з тієї лунки, до якої потрапила остання куля на попередньому ході. Якщо остання кулька потрапляє в пусту лунку гра вважається програною. Зберіть усі кульки до руми.

15. Визначте , чи є рядок

$$(((\,),(\,)),(\,),((\,),(\,)))$$

реченням S у граматиці, що визначається такими правилами переписування:

а) $S \leftarrow (\,)$;
б) $A \leftarrow S$;
в) $A \leftarrow A, A$;
г) $S \leftarrow (A)$.

Вважається, що символ розташований ліворуч від стрілки, може бути підставлений на місце підрядка символів, розташованих праворуч від стрілки, у будь якому місці рядка, у якому зустрівся цей підрядок.

16. Дуга між вершиною n_i та наступною за нею вершиною n_j називається неповерненою, якщо n_i недосяжна з n_j. Наведіть два приклади задач, для яких графи у просторі станів містять неповернені вершини. Запропонуйте подання цих задач у просторі станів.

17. Запишіть у інфіксній формі та подайте у вигляді двійкового дерева такий вираз: [(A + B)×C + (A - D/E)]×F.

18. Запишіть у інфіксній формі та подайте у вигляді двійкового дерева такий вираз: (A/G + B×F) ×C - (A - D/E) ×H.

19. Наведіть формальний опис (використовуючи знак $, який позначає довільний підрядок, в тому числі і пустий), для таких правил перетворювання рядків:

- два ідентичних підрядки, розділені символом A можуть бути замінені символом B;
- підрядок, що розташований між двома символами B, може бути добавлений в кінець рядка.

20. Задані правила перетворення рядків:

$1$2$3 -> $1$2$2$3;

$1$2$2$3 -> $1$2$3,

де $i - довільний підрядок, в тому числі і пустий.

Доведіть, чи достатньо цих правил для перетворення рядка ABCBABC в рядок ABCAB. При позитивній відповіді наведіть розв'язання задачі.

2 ПОШУК РОЗВ'ЯЗКІВ У ПРОСТОРІ СТАНІВ

2.1 Загальна постановка задачі пошуку розв'язків у просторі станів

В загальному випадку задача пошуку розв'язків у просторі станів може бути подана четвіркою [1, 3]:

$$(S, S_0, F, G),$$

де S - множина станів;

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

F – множина операторів переходів між станами, що перетворюють одні стани на інші;

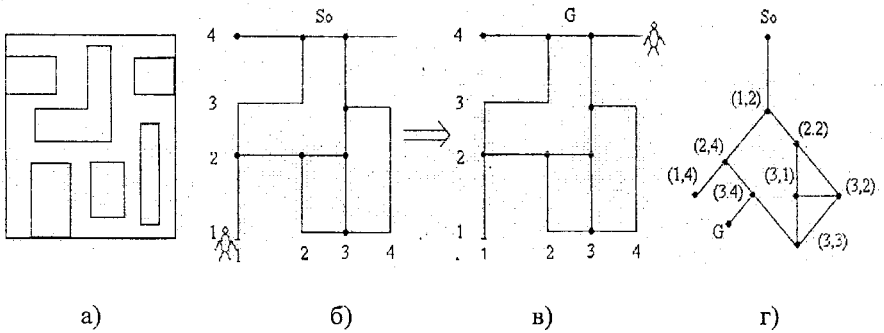
G – множина цільових станів, $G \in S$.

Кожний оператор $f \in F$ є функцією, що відображає один стан в інший – $s_j = f(s_j)$, де $(s_j, s_j) \in S$. Розв'язком задачі є послідовність операторів $f_i \in F$, що перетворює початкові стани на кінцеві, такі, що

$$f_n(f_{n-1}(\ldots(f_2(f_1(S_0)))\ldots)) \in G.$$

Якщо існує більше ніж одна така послідовність, і заданий критерій оптимальності, то виникає задача пошуку оптимальної послідовності.

Проілюструємо сказане прикладом пошуку шляху в лабіринті (рис. 2.1а) [9]. Початковий і кінцевий стани задачі можна задати за допомогою схем, зображених на рис.2.1б і рис.2.1в.



а) б) в) г)

Рисунок 2.1 – Задача пошуку шляху в лабіринті:
а) схема лабіринту; б) початковий стан; в) цільовий стан;
г) графове подання простору станів задачі

Хоча мандрівник може знаходитися в будь-якій точці лабіринту, нас цікавлять тільки ті точки простору, в яких шляхи перетинаються, і які потребують від мандрівника прийняти рішення щодо подальшого напрямку руху. Тому можливі стани задачі відображають присутність мандрівника на перехрестях шляхів. Переміщення мандрівника від одного перехрестя до іншого, відповідає переходам задачі з одного стану до

іншого. Отже, множина операторів задачі подається лише одним оператором пересування, а розв'язування задачі зводиться до знаходження послідовності операторів, що перетворюють стан задачі з початкового S_0 до кінцевого G.

Простір станів цієї задачі можна подати у вигляді графа (рис.2.1г). Вершини графа, позначені координатами перехресть, відповідають станам задачі, а дуги - операторам пересування. Оскільки при розв'язанні цієї задачі використовується лише один оператор, дуги не мають міток. Розв'язування задачі полягає у визначенні шляху від початкової вершини S_0 до цільової вершини G.

Неважко побачити, що в даній задачі до цільової вершини ведуть аж п'ять різних шляхів. Якщо кількість елементарних відрізків між точками перетинів використовувати як оцінку вартості шляху, то сумарні вартості кожного з п'яти шляхів складуть: 4, 6, 6, 7, 7. За оптимальний можна вибрати один з двох шляхів мінімальної вартості.

Іноді розглядають пошук з обмеженнями. Обмеження є деякими умовами, що мають виконуватися при досягненні цільового стану. Прикладом такої умови при пошуку шляху в лабіринті може бути вимога, щоб довжина пройденого шляху не перевищувала певного значення L. Тоді стан задачі визначатиметься позицією мандрівника в лабіринті і пройденою відстанню.

Граф станів задачі може задаватися неявно. В цьому випадку задається множина початкових станів S_0 і множина операторів F, які, при застосуванні до вершини графа, будують всі його дочірні вершини. Процес застосування операторів до деякої вершини з метою отримання всіх її дочірніх вершин називається розкриттям вершини. При цьому можуть задаватися умови можливості застосування оператора до вершини (стану). Наприклад, оператор пересування в задачі пошуку шляху в лабіринті може бути виконаний лише за умови існування шляху між відповідними вершинами. Якщо розглядається декілька операторів, то кожний з них може мати власні умови застосування.

Методи пошуку в просторі станів розділяються на дві групи: методи сліпого і впорядкованого (евристичного) пошуку. Перевагою методів сліпого пошуку є простота алгоритмічної реалізації і гарантія отримання розв'язку, якщо він існує. Методи сліпого пошуку називають також неінформованими методами, оскільки в процесі пошуку шляху на графі не враховується інформація про ступінь близькості поточної та цільової вершини. Внаслідок цього в методах сліпого пошуку виконується повний перегляд всього простору станів. Для нетривіальних задач такий перегляд є неможливим через надто великий розмір простору станів. В цьому випадку виникає проблема комбінаторного вибуху, коли розмір простору рішень зростає надзвичайно швидко із зростанням кількості можливих альтернатив. Ефективним засобом боротьби з комбінаторною складністю є

методи евристичного пошуку, що дозволяють істотно зменшити обсяг пошуку у великих просторах рішень.

Загальна процедура пошуку на графах базується на використанні двох списків: списку OPEN, що містить ідентифікатори (імена) вершин, які ще підлягають розкриттю у майбутньому, і списку CLOSED, де знаходяться імена вершин, вже розкритих на дану мить часу, і ім'я вершини, яка буде розкрита на поточному кроці роботи алгоритму [3, 5].

2.2 Неінформовані процедури пошуку у просторі станів

2.2.1 Випадковий пошук. При випадковому пошуку інформованість алгоритму обмежується топологією графа простору станів. Стратегія пошуку полягає в тому, що, починаючи з початкового стану, оператор переходу до наступного стану задачі вибирають цілком випадково. Пошук продовжується доти, доки не буде виконаний перехід до цільового стану.

Дана процедура може бути описана таким чином:

```
Procedure Random_Search;
Begin
  n := „початкова вершина";
  While n <> „цільова вершина" Do
      Begin
        Вибрати випадково оператор, який можна застосувати до „n";
        Застосувати оператор до „n" і отримати нову вершину „n_New";
        n := n_New;
End.
```

Випадковий вибір оператора не гарантує збіжності алгоритму. Проте, такий метод пошуку забезпечує розв'язання простих задач з обмеженим простором пошуку, за наявності шляху між початковою і цільовою вершиною.

Оскільки граф станів обстежується випадковим чином, без запам'ятовування пройденого маршруту, можливий багатократний вибір одних і тих самих станів. Покращити процедуру пошуку можна введенням певної систематичності вибору переходів між станами.

2.2.2 Пошук в глибину і в ширину. Дані методи виключають повторний вибір одних й тих самих вершин. Для цього за допомогою списків CLOSED і OPEN ведеться облік, відповідно, вже розкритих вершин і вершин, що підлягають розкриттю. Крім того, задається певний порядок (систематичність) обстеження графа станів, що дозволяє послідовно пройти всі маршрути по одному разу, без пропусків і повторень [1, 3, 6]. Розглянемо процедуру пошуку в глибину:

```
Procedure Depth_First;
Begin
    Помістити початкову вершину до списку OPEN;
    CLOSED := „пустий список";
```

```
While OPEN <> „пустий список" Do
    Begin
      n := first (OPEN);
      if n = „цільова вершина" Then Вихід (УСПІХ);
      Перемістити n зі списку OPEN до списку CLOSED;
      Розкрити вершину n і помістити всі її дочірні вершини,
відсутні
      в списках OPEN і CLOSED, на початок списку OPEN,
      зв'язавши з кожною дочірньою вершиною показчик на n;
    End;
  Вихід (НЕВДАЧА);
End.
```

На початку пошуку список CLOSED порожній, а OPEN містить тільки початкову вершину. На кожному кроці роботи алгоритму з голови списку OPEN вибирається для розкриття перша вершина (виклик функції first(OPEN)). Розкрита вершина переміщається в список CLOSED, а її дочірні вершини поміщаються на початок списку OPEN. Таким чином, на наступному кроці розкриватимуться дочірні вершини поточної вершини n. Принцип формування списку OPEN при цьому відповідає стеку, тобто вершина, додана до списку останньою, обробляється першою (LIFO - Last In First Out). Якщо прослідити напрям пошуку по дереву станів, то можна побачити, що фронт пошуку росте в глибину. Зауважимо, що всі дочірні вершини у процесі пошуку забезпечуються покажчиками на відповідні батьківські вершини для можливості простеження зворотного шляху (з цільової вершини до початкової), який подає розв'язок задачі .

Процедура пошуку в ширину відрізняється від розглянутої процедури пошуку в глибину лише тим, що дочірні вершини, одержувані при розкритті вершини n, поміщаються в кінець списку OPEN. Отже, при пошуку в ширину принцип формування списку OPEN відповідає черзі, тобто вершина, внесена до списку першою, обробляється першою (FIFO - First In First Out). Завдяки цьому фронт пошуку росте в ширину, що гарантує знаходження шляху з мінімальною кількістю дуг (ребер) між початковою і цільовою вершинами.

2.2.3 Алгоритм рівних цін. Порівняно з пошуком у глибину або в ширину, даний алгоритм потребує більшої інформованості про задачу. Кожному оператору, що перетворює стан n_i на стан n_j, ставиться у відповідність деяка функція вартості $c\{n_i,n_j\}$, що характеризується позитивними значеннями. В ході пошуку шляху на графі враховуються сумарні витрати на досягнення вершини, тобто вартість шляху до цієї вершини. Вартість шляху до поточної вершини n_j визначають за такою формулою:

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

де $g(n_i)$ - вартість шляху з початкової вершини до поточної вершини n_i.

У процесі пошуку, вершини в списку OPEN перевпорядковуються на кожному кроці алгоритму в порядка збільшення вартості, і кожного разу для розкриття вибирається вершина з найменшою вартістю. Це дозволяє знайти шлях мінімальної вартості між початковою та цільовою вершинами. Зауважимо, що у випадку виявлення в процесі пошуку більш дешевого шляху ніж поточний, вартість шляху до поточної вершини може змінюватися. Тому будемо позначати вартість оптимального шляху з початкової вершини до вершини n як $g(n)$, а вартість поточного шляху, знайденого алгоритмом, через $\hat{g}(n)$, тобто $\hat{g}(n)$ є наявною на дану мить часу оцінкою істинної вартості оптимального шляху $g(n)$. Наприкінці пошуку ці значення прирівнюються, тобто виконується $\hat{g}(n)=g(n)$.

Процедура пошуку оптимального шляху (мінімальної вартості) може бути описана таким чином:

```
Procedure Optimal_Search;
Begin
   Помістити початкову вершину до списку OPEN;
   CLOSED := „пустий список";
   While OPEN <> "пустий список" Do
        n := first(OPEN);
        if n = "цільова вершина" Then Вихід (УСПІХ);
        Перемістити вершину n зі списку OPEN до списку CLOSED;
        Розкрити вершину n і для кожної дочірньої вершини j обчислити
        вартість ĝ(n,nj);
        Помістити дочірні вершини, відсутні в списках OPEN і CLOSED,
до
        списку OPEN, зв'язавши з кожною дочірньою вершиною
покажчик
        на вершину n і покласти ĝ(nj)=ĝ(n,nj);
        Для кожної з дочірніх вершин, які вже містяться в списку OPEN,
        порівняти поточну вартість ĝ(n,nj) з раніше обчисленим
значенням
        вартості ĝ(nj), що зберігається в списку OPEN, якщо ĝ(n,nj) <
ĝ(nj),
        то встановити ĝ(nj) = ĝ(n,nj). Забезпечити вказані дочірні
вершини
        покажчиками на вершину n;
        Упорядкувати список OPEN за збільшенням вартості;
   End;
   Вихід(НЕВДАЧА);
End.
```

При розкритті вершини n, оптимальний шлях до неї вже знайдений, тобто $\hat{g}(n)=g(n)$ [2, 8]. Іншими словами, якщо вершина вже знаходиться на початку списку OPEN, то для неї неможливо знайти більш дешевого шляху. Отже, переміщення вершини в список CLOSED є остаточним.

Розглянута процедура пошуку гарантує знаходження шляху мінімальної вартості, якщо граф кінцевий і існує шлях з початкової вершини до цільової вершини. Проте сам процес пошуку не є оптимальним, оскільки пошук виконується без урахування відстаней від поточних вершин до цільових.

2.3 Евристичний пошук

В зв'язку з існуючими обмеженнями на час обчислень та доступні обсяги пам'яті, методи сліпого перебору частіше за все неможливо використовувати на практиці. Разом з тим часто можна сформулювати деякі евристичні правила, що дозволяють значно скоротити обсяги перебору.

У [10] евристика визначається як „вивчення методів і правил відкриттів та винаходів". Це слово має грецьке коріння, глагол eurisko означає „досліджувати".

У просторі станів пошуку евристика визначається як набір правил, для вибору тих шляхів з простору станів, які з найбільшою ймовірністю приведуть до задовільного розв'язання проблеми.

У штучному інтелекті евристику використовують у двох основних ситуаціях.

1. Задача може не мати точного розв'язку в зв'язку з невизначеністю у її постановці І-АБО у початкових даних. Прикладом такої задачі є медична діагностика. Певний набір ознак може мати кілька причин; тому лікарі використовують евристику, щоб поставити найбільш точний діагноз і підібрати відповідні методи лікування. Як інший приклад задачі з невизначеністю можна навести систему технічного зору. Візуальні сцени часто є неоднозначними і викликають різні (часто не пов'язані одне з одним) припущення щодо протяжності та орієнтації об'єктів. Гарною ілюстрацією таких неоднозначностей є оптична омана. Системи технічного зору часто вимушені використовувати евристику для вибору найбільш імовірної інтерпретації з кількох можливих.

2. Задача може мати точний розв'язок, але вартість його пошуку може бути надто високою. У багатьох задачах (наприклад у грі в шахи), простір станів зі збільшенням глибини пошуку зростає експоненціально і прямий пошук в глибину і в ширину стає практично неможливим. Евристика дозволяє уникнути такої складності і вести пошук на найбільш „перспективному" шляху, вилучаючи з перегляду неперспективні стани разом з їх нащадками.

Евристика є лише припущенням наступного кроку, який слід зробити на шляху розв'язування. Вона часто ґрунтується на досвіді або інтуїції і, на жаль, може помилятися. Оскільки евристика використовує лише таку обмежену інформацію, як опис поточних станів у списку OPEN, вона рідко здатна передбачити подальшу поведінку простору станів. Вона може привести алгоритм пошуку до субоптимального розв'язку або взагалі не знайти розв'язку задачі. Це обмеження закладено у природу евристичного пошуку і не може бути подолане „кращою" евристикою, або більш ефективним алгоритмом пошуку. Проте у багатьох випадках евристичний пошук є єдиним методом розв'язання задач, в яких неможливо перебрати усі можливі стани.

Евристичні алгоритми складаються з двох частин,: евристичної міри (оцінки) і алгоритму, який використовує її у просторі станів. Список відкритих вершин упорядковується в них за збільшенням значень деякої оціночної функції, яка формується на основі евристичних правил. Оціночна функція може включати дві складові, одна з яких називається евристичною і є оцінкою близькості поточної і цільової вершин. Чим меншим є значення евристичної складової оціночної функції, тим ближчою є поточна вершина до цільової.

Залежно від способу формування оціночної функції виділяють такі алгоритми евристичного пошуку, як: алгоритм "підйому на гору", алгоритм глобального врахування відповідності цілі, алгоритм A*.

2.3.1 Алгоритм підйому на гору. Алгоритм здійснює цілеспрямований пошук у напрямку найбільшого зменшення евристичної оціночної функції $\hat{h}(n)$, яка забезпечує оцінку (прогноз) вартості найкоротшого шляху від поточної вершини n до найближчої цільової вершини, тобто є мірою вартості шляху, що залишився. Чим меншим є значення $\hat{h}(n)$, тим більш перспективним є шлях, на якому знаходиться вершина n.

Подібний алгоритм використовується при пошуку екстремумів функцій. Відомо, що положення екстремуму функції багатьох змінних визначається напрямком вектора градієнта функції. Тому пошук екстремуму здійснюють у напрямку найбільшого зростання (убування) функції, тобто в напрямку, який збігається (або протилежний) з напрямком градієнта. Пошук максимуму функції в цьому випадку нагадує сходження на пік по найкрутішому маршруту. Тому даний алгоритм і називають алгоритмом „підйому на гору" (hill-climbing).

На кожному кроці алгоритму для кожної з дочірніх вершин $n_1, n_2..., n_m$ вершини n обчислюється значення оціночної функції $\hat{h}(n_i)$ і для продовження шляху вибирається вершина з найменшим значенням $\hat{h}(n_i)$. Процедура пошуку терпить невдачу, якщо для всіх дочірніх вершин $\hat{h}(n_i) \geq \hat{h}(n)$:

```
Procedure Hill_Climbing;
Begin
    N := „початкова вершина” ;
    While n <> „цільова вершина” Do
    Begin
        Розкрити вершину n і для усіх дочірних вершин n_i, обчислити
        оцінки ĥ(n_i);
        Вибрати дочірню вершину n_i, з мінімальним значенням ĥ(n_i);
        if ĥ(n_i) ≥ ĥ(n)Then Вихід(НЕВДАЧА);
        n:=n_i;
    End;
    Вихід(УСПІХ);
End.
```

Існують різні способи побудови евристичних оціночних функцій, проте готові рецепти відсутні. При розв'язуванні кожної конкретної задачі використовують раніше накопичений досвід розв'язання подібних задач. Наприклад, приклад для пошуку шляху в лабіринті (рис. 2.1) оцінка $\hat{h}(n_i)$ може обчислюватися за такою формулою [13]:

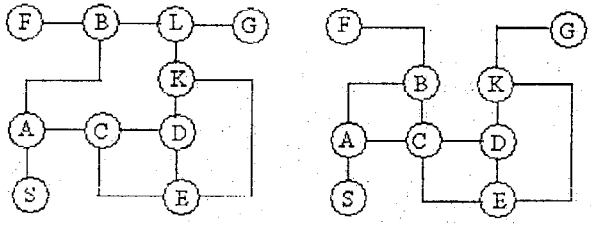
$$\hat{h}(n_i) = |x_g - x_n| + |y_g - y_n|,$$

де x_g, y_g - координати цільової вершини;

x_n, y_n - координати поточної вершини.

Дана оцінка відповідає відстані від вершини n до цільової вершини. При пошуку шляху в лабіринті, зображеному на рис. 2.2, а, з використанням запропонованої оціночної функції, вершини будуть розкриватися у такому порядка: A, B, L, G. Але пошук шляху в лабіринті, наведеному на рис. 2.2, б, завершиться невдачею. При цьому послідовність розкритих вершин буде мати такий вигляд: A, B. Оціночна функція $\hat{h}(n_i)$ у вершині B отримає значення 3. Подальші кроки виявляються безуспішними, оскільки $\hat{h}(F)$=3 і $\hat{h}$(C)=4, а для продовження пошуку необхідно, щоб виконувалася умова $\hat{h}(n_i) < \hat{h}(n)$.

Таким чином, хоча даний алгоритм забезпечує прискорення пошуку, він не гарантує досягнення цільової вершини. Передчасне завершення алгоритму наступає у випадках, коли в процесі пошуку зустрічаються локальні мінімуми оціночної функції $\hat{h}(n)$, або коли для групи сусідніх вершин оцінки $\hat{h}(n)$ виявляються рівними між собою (проблема „плато”).



а) б)

Рисунок 2.2 – Графи станів для задачі пошуку „підйомом на гору”: а) приклад вдалого пошуку; б) приклад невдалого пошуку

2.3.2 Глобальне врахування відповідності цілі. Цей алгоритм пошуку рішення задачі схожий на попередній. Тут також оцінюється „перспективність” того або іншого шляху за допомогою евристичної оціночної 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, що знаходяться в списку OPEN. Для цього список відкритих вершин сортується в порядка зростання значень $\hat{h}(n)$. Найкраща вершина для продовження шляху знаходитиметься на першому місці в списку OPEN. Тому даний алгоритм називають також алгоритмом вибору першої найкращої вершини (Best-First).

Процедура пошуку, основана на даному алгоритмі, схожа на процедуру Optimal_Search. Відмінність полягає в тому, що замість функції вартості $\hat{g}(n)$, застосовується евристична функція $\hat{h}(n)$, тобто враховуються тільки майбутні витрати:

```
Procedure Best_First_Search;
Begin
   Помістити початкову вершину до списку OPEN;
   CLOSED := „пустий список”;
   While OPEN <> "пустой список" Do
        n := first(OPEN);
        if n = "цільова вершина" Then Вихід (УСПІХ);
        Перемістити вершину n зі списку OPEN до списку CLOSED;
        Розкрити вершину n, для кожної дочірньої вершини j обчислити
        вартість ĥ(n,nj);
        Помістити дочірні вершини, відсутні в списках OPEN і CLOSED,
до
        списку OPEN, пов'язавши з кожною дочірньою вершиною
покажчик
        на вершину n;
        Упорядкувати список OPEN за збільшенням значень ĥ(n);
```

```
    End;
    Вихід(НЕВДАЧА);
End.
```

Якщо дану процедуру застосувати до задачі пошуку шляху в лабіринті, зображеному на рис. 2.2, а, то результат буде таким же, як і при використанні алгоритму підйому на гору. Якщо ж процедуру Best_First_Search застосувати до лабіринту, поданого на рис. 2.2, б, то буде отримано такий список відкритих у процесі пошуку вершин:

(S(6))→(A(5))→(B(3) C(4))→(F(3) C(4))→(C(4))→(D(3) E(4))→
→(K(2) E(4))→(G(0) E(4)).

Таким чином, алгоритм дозволяє успішно долати проблему локальних мінімумів. Проте при цьому знижується ефективність самого процесу пошуку. Ефективність процесу пошуку залишатиметься високою, якщо оцінка вартості найкоротшого шляху $\hat{h}(n)$ з вершини n до цільової вершини буде якомога ближча до реальної вартості $h(n)$ [9].

2.4 Алгоритм A*

Перевага алгоритму рівних цін полягає в тому, що він гарантує знаходження оптимального шляху. Перевагою алгоритмів, що використовують оцінку майбутніх витрат, є можливість усікання дерева пошуку, що підвищує ефективність пошуку. Природним є прагнення комбінувати ці алгоритми і враховувати як здійснені, так і майбутні витрати. Цього можна домогтися, скориставшись оціночною функцією вигляду:

$$\hat{f}(n) = \hat{g}(n) + \hat{h}(n), \qquad (2.1)$$

де $\hat{g}(n)$ - оцінка вартості шляху з початкової вершини у вершину n;

$\hat{h}(n)$ - оцінка вартості найкоротшого шляху з вершини n до цільової вершини.

Алгоритм, що базується на використанні оціночної функції $\hat{f}(n)$, називається А-алгоритмом. Оцінки $\hat{h}(n)$ в ході пошуку не змінюють своїх значень. Якщо пошук шляху виконується на дереві станів, то оцінки $\hat{g}(n)$ також не змінюють своїх значень і завжди дорівнюють реальним витратам $g(n)$. Проте при пошуку шляху на графі станів оцінки $\hat{g}(n)$ можуть змінюватися. Отже, в загальному випадку оцінки $\hat{f}(n)$ змінюють свої значення в процесі пошуку. Це призводить до того, що вершини зі списку CLOSED можуть переміщатися назад до списку OPEN:

```
Procedure A_Search;
Begin
```

Розмістити початкову вершину s до списку OPEN: $\hat{f}(s) = \hat{h}(s)$;
CLOSED := „пустий список";
While OPEN <> „пустий список" Do
Begin
n := first(OPEN);
if n = „цільова вершина" Then Вихід(УСПІХ);
Перемістити n з OPEN до CLOSED;
Розкрити n і для всіх її дочірніх вершин n_j, обчислити оцінку

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

Розмістити дочірні вершини, відсутні в списках OPEN і CLOSED, до
списку OPEN, пов'язавши з кожною вершиною покажчик на вершину n;
Для дочірніх вершин n_j, які вже містяться в OPEN, порівняти оцінки $\hat{f}(n,n_j)$ і, $\hat{f}(n_j)$, якщо $\hat{f}(n,n_j) < \hat{f}(n_j)$, то пов'язати з вершиною n_j, нову
оцінку $\hat{f}(n,n_j)$ і покажчик на вершину n;
Якщо вершина n_j міститься в списку CLOSED і $\hat{f}(n,n_j) < \hat{f}(n_j)$, то
пов'язати з вершиною n_j нову оцінку $\hat{f}(n,n_j)$, перемістити її до списку
OPEN і встановити покажчик на n;
Упорядкувати список OPEN за збільшенням значень $\hat{f}(n_j)$;
End;
Вихід (НЕВДАЧА);
End.

Пояснимо роботу алгоритму на прикладі [5]. Нехай процес пошуку породив граф та дерево пошуку, наведені на рис. 2.3, а.

Стрілки тут відображають покажчики, що визначають батьківські вершини в дереві пошуку (при цьому послідовність покажчиків утворює оптимальний, на дану мить, шлях від даної вершини до початкової). Чорні вершини знаходяться в списку CLOSE, білі - в списку OPEN на ту мить, коли алгоритм вибирає для розкриття вершину 1.

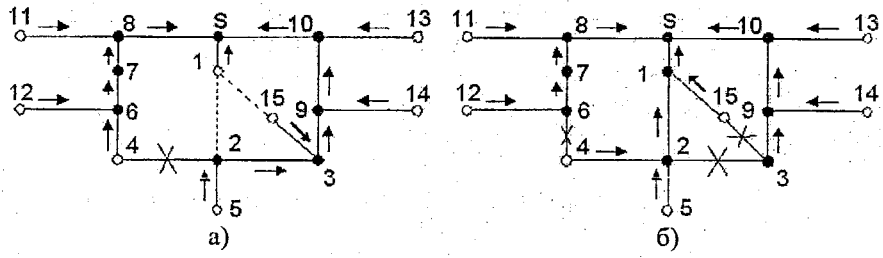


Рисунок 2.3 – Приклад побудови графа і дерева розв'язку у процесі пошуку:
а) граф і дерево пошуку до розкриття вершини 1;
б) граф і дерево пошуку після розкриття вершини 1.

На рис. 2.4, а наведено вміст списків OPEN і CLOSE до моменту розкриття вершини 1.

OPEN: $1(S)_1$ $11(8)_2$ $13(10)_2$ $14(9)_3$ $4(6)_4$ $12(6)_4$ $15(3)_4$ $5(2)_4$
CLOSE: $2(3)_4$ $3(9)_3$ $9(10)_2$ $10(S)_1$ $6(7)_3$ $7(8)_2$ $8(S)_1$

а)

OPEN: $\mathbf{2(1)_2}$ $11(8)_2$ $\mathbf{15(1)_2}$ $13(10)_2$ $14(9)_3$ $4(6)_4$ $12(6)_4$ $5(2)_4$
CLOSE: $\mathbf{1(S)_1}$ $3(9)_3$ $9(10)_2$ $10(S)_1$ $6(7)_3$ $7(8)_2$ $8(S)_1$

б)

OPEN: $11(8)_2$ $15(3)_2$ $13(10)_2$ $\mathbf{4(2)_3}$ $14(9)_3$ $12(6)_4$ $\mathbf{5(2)_3}$
CLOSE: $1(S)_1$ $\mathbf{2(3)_2}$ $3(9)_3$ $9(10)_2$ $10(S)_1$ $6(7)_3$ $7(8)_2$ $8(S)_1$

в)

Рисунок 2.4 - Зміни у вмісті списків CLOUSE і OPEN при побудові графа станів і дерева розв'язку процедурою A_Search:
а) до розкриття вершини 1; б)після розкриття вершини 1;
в) після розкриття вершини 2

Перше число ідентифікатора вузла у списках визначає номер вузла, друге число (в дужках) – номер батьківської вершини у дереві пошуку, третє число (індекс) – відстань від поточної вершини до початкової. Вузли з рівними оцінками розміщуються у списках у порядка зростання їх номерів.

При розкритті вершини 1 відбуваються такі події:

- вершина 1 переноситься до списку CLOSE;
- породжуються спадкоємці вершини 1, якими є вершини 2 і 15;
- до вершин 2 і 15 визначаються шляхи від початкової вершини через вершину 1.

Але вершини 2 і 15 вже знаходяться у списках CLOUSE і OPEN, відповідно. Отже, до них вже раніше були визначені у дереві пошуку шляхи від початкової вершини через інших попередників (у нашому прикладі вершина 3). Оскільки нові шляхи виявилися більш короткими за старі (вартості 2, замість 4), це приводить до виконання алгоритмом пошуку таких дій:

- змінюються покажчики від вершин 2 і 15 на попередника, яким стає для них вершина 1 замість вершини 3;
- вершини 2 і 15 переміщуються до нових позицій у списку OPEN відповідно до нових значень їх оцінок;
- вершина 2 переноситься зі списку CLOSE до списку OPEN для забезпечення у подальшому відношень наслідування між нею та суміжними з нею вершинами графа станів.

На рис. 2.4, б наведено новий вміст списків OPEN і CLOUSE, в яких зміни, що відбулися в результаті розкриття відповідної вершини, виділені жирним шрифтом.

Оскільки вершина 2 розміщується (відповідно до значення нової оцінки) на першу позицію списку OPEN, саме вона розкривається на наступному кроці роботи алгоритму пошуку .

Метою її повторного розкриття (зауважимо, що вона вдруге потрапили до списку OPEN) є перевірка, чи не увійдуть до множини її нащадків у дереві пошуку крім вершини 5, ще й вершини 3 і 4, які після першого її розкриття були віднесені як нащадки до вершин 9 і 6, відповідно (тобто, чи не потрібно орієнтувати покажчики вершин 3 і 4 на вершину 5). Аналіз необхідності таких змін, що базується на порівнянні вартостей „старих" і „нових" шляхів від вершин 3, 4, 5 до початкової вершини, приносить такі результати:

- вартість шляху від вершини 3 до початкової вершини через вершини 9 і 2 є однаковою (дорівнює 3), отже орієнтація покажчика не змінюється і він, як і раніше, залишається спрямованим до вершини 9;
- вартість шляху від вершини 4 до початкової вершини через вершину 6 складала 4, а через вершину 2 – складає 3. Отже, для вершини 4 покажчик змінює орієнтацію і замість вершини 6, буде вказувати тепер на вершину 2;
- покажчик вершини 5 як і раніше спрямовується на вершину 2, яка була і залишилася його попередником, але вартість шляху до початкової вершини від вершини 5 зменшилася з 5 до 3, оскільки з 4 до 2 зменшилася вартість шляху до початкової вершини його попередника, вершини 2.

Відкореговане дерево пошуку відображено на рис. 2.3, б. Результуючий вміст списків OPEN і CLOUSE показано на рис. 2.4, в.

2.4.1 Властивості алгоритму А*

Для дослідження властивостей алгоритму А*, розглянемо, спочатку, основні властивості А-алгоритму. Якщо для всіх вершин виконується

$\hat{h}(n)=0$, то А-алгоритм відповідатиме алгоритму рівних цін. При цьому гарантується знаходження оптимального шляху, але ефективність пошуку буде низькою, і у разі нескінченного простору пошуку, пошук виявиться просто нездійснимим.

Зауважимо, що А-алгоритм не дає гарантії, що знайдений шлях буде оптимальним. Шлях може бути не оптимальним у випадку, якщо оцінка витрат на шляху з вершини n до цільової вершини буде перебільшена (переоцінсна), тобто якщо буде виконуватись умова $\hat{h}(n)>h(n)$. Приклад такої ситуації наведено на рис. 2.5 [1].

На рисунку числа в дужках при вершинах відповідають оцінкам $\hat{h}(n)$, а цифри поряд з ребрами - вартостям переходів $c(n_i,n_j)$. При розкритті вершини S одержуємо дочірні вершини A і B. Оцінки $\hat{f}(n)$ для цих вершин, обчислені за допомогою виразу (2.1.), дорівнюють: $f(A)=2+4$, $f(B)=3+5$. Отже, на наступному кроці розкривається вершина A, і обчислюється $f(G)=7+0$. Отримуємо, як розв'язок, шлях S→A→G, який для даного графа не є оптимальним.

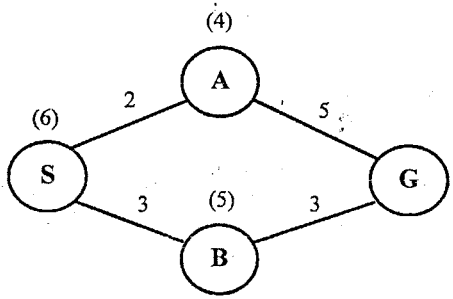


Рисунок 2.5 - Граф станів з переоцінкою витрат $h(n)$

А-алгоритм завжди знаходить оптимальний шлях, якщо виконується умова:

$$\hat{h}(n) \le h(n). \qquad (2.2)$$

Алгоритм, що враховує умову (2.2), називають алгоритмом А* або допустимим алгоритмом. Алгоритм А* завжди недооцінює витрати на шляху з проміжної вершини в цільову вершину або оцінює їх правильно, але ніколи не переоцінює.

Нехай А1* і А2* - довільні допустимі алгоритми і для всіх вершин виконується $\hat{h}_1(n)>\hat{h}_2(n)$. Тоді кажуть, що інформованість А1* вища за інформованість А2*, і що А1* має більшу евристичну силу, ніж А2*.

Евристично більш сильний алгоритм А1* більшою мірою скорочує простір пошуку.

Ефективність пошуку з використанням алгоритму А* може знижуватися через те, що вершина, яка знаходиться в списку CLOSED, може повторно потрапляти до списку OPEN, і повторно розкриватися. Приклад такого багаторазового розкриття вершин наведений на рис. 2.6.

Стани списків OPEN і CLOSED наведені в таблиці 2.1. Кожний рядок таблиці відповідає одному крокові роботи алгоритму. Числа поряд з іменами вершин відповідають значенням оціночної функції $\hat{f}(n)$. З таблиці 2.1 випливає, що вершина А розкривається чотири рази, вершина В - тричі, вершина С - двічі.

Для запобігання повторному розкриттю алгоритмом А* однієї і тієї ж вершини, необхідно забезпечити умову монотонності:

$$\hat{h}(n_i) \le \hat{h}(n_j) + c(n_i,n_j), \qquad (2.3)$$

де n_i - батьківська вершина;

n_j - дочірня вершина;

$c(n_i,n_j)$ - вартість шляху між вершинами n_i та n_j.

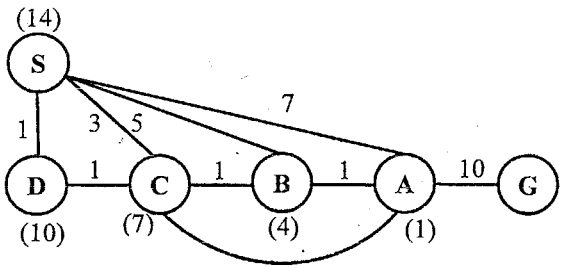


Рисунок 2.6 - Граф станів з вершинами, що розкриваються багато разів

Якщо умова монотонності виконується для всіх дочірніх вершин, то можна довести, що при розкритті будь-якої вершини n, оптимальний шлях до неї вже знайдений. Отже, оціночна функція $\hat{f}(n)$ для даної вершини надалі не змінює своїх значень, і ніякі вершини зі списку CLOSED до списку OPEN не повертаються.

Як вже відмічалося, алгоритм А* недооцінює витрати $h(n)$. Умова монотонності означає, що недооцінка на шляху до мети поступово зменшується або, у всякому випадку, не змінюється. Дійсно, якщо позначити недооцінку через $\hat{u}(n)$, то

$$\hat{u}(n) = h(n) - \hat{h}(n), \quad \hat{u}(n) \ge 0.$$

Таблиця 2.1 — Виконання алгоритму А*

Номер	OPEN	CLOSED	Коментар
1	S(14)	-	
2	A(8)B(9)C(10)D(1I)	S(14)	
3	B(9)C(10)D(n)G(17)	A(8)S(14)	
4	A(7)C(10)P(11)G(17)	B(9) S(14)	Повернення: А
5	C(10)D(11)0(16)	A(7)B(9)S(14)	
6	A(6)B(8)D(11)G(16)	C(10)S(14)	Повернення: А, В
7	B(8)D(n)G(15)	A(6)C(10)S(14)	
8	D(U)G(15)	B(8)A(6)C(10)S(14)	
9	C(9)G(15)	D(11)B(8)A(6)S(14)	Повернення: С
10	A(5)B(7)G(15)	C(9)D(11)S(14)	Повернення: А, В
11	B(7)G(14)	A(5)C(9)D(11)S(14)	
12	G(14)	B(7)A(5)C(9)D(11)S(14)	

З умови монотонності випливає, що $\hat{u}(n_j) \le \hat{u}(n_i)$, тобто недооцінка $h(n)$ у батьківських вершинах більша або дорівнює недооцінкам $h(n)$ у дочірніх вершинах. На графі, зображеному на рис.2.4, умова монотонності не виконується для початкової вершини. Наприклад, $h(S) > h(D) + c(S,D)$, тобто $14 > 10 + 1$.

Якщо $\hat{h}(n) = h(n)$, то інформованість є повною. В цьому випадку ніякого пошуку не відбувається і наближення до мети йде оптимальним шляхом.

Таким чином, властивості А-алгоритму значно залежать від умов, які задовольняє або не задовольняє евристична частина оціночної функції $\hat{f}(n)$.

1. При виконанні умови $h(n)=0$, А-алгоритм відповідає алгоритму рівних цін.
2. При виконанні умови $\hat{h}(n) \le h(n)$, А-алгоритм гарантує знаходження оптимального розв'язку, якщо такий існує; в цьому випадку він називається алгоритмом А*.
3. При виконанні умови монотонності $\hat{h}(n_i) \le \hat{h}(n_j) + c(n_i,n_j)$, А-алгоритм знаходить оптимальний шлях до вершини безпосередньо в процесі вибору її для розкриття, що виключає необхідність перевірки того, чи вже знаходиться знов породжена вершина в списку CLOSE та чи

потрібно змінювати батьківські відношення на дереві пошуку для будь-яких спадкоємців цієї вершини в графі пошуку.

4. При виконанні умови $\hat{h}_1(n) > \hat{h}_2(n)$, алгоритм А1* є евристично більш сильним за алгоритм А2*.

5. При виконанні умови $\hat{h}(n) = h(n)$, алгоритм А* є повністю інформованим.

6. При виконанні умови $\hat{h}(n) > h(n)$ А-алгоритм не гарантує отримання оптимального рішення, проте отримання розв'язку прискорюється.

Систематизацію розглянутих алгоритмів пошуку в просторі станів наведено у таблиці 2.2 [1].

Таблиця 2.2 - Алгоритми пошуку в просторі станів

<table>
<tr><td>Несистематичні</td><td colspan="5">Систематичні</td></tr>
<tr><td colspan="2">Неінформовані</td><td colspan="4">Інформовані</td></tr>
<tr><td colspan="2">Без оцінки витрат</td><td colspan="4">З оцінкою витрат</td></tr>
<tr><td></td><td>список OPEN-Механізм „LIFO”</td><td>список OPEN-механізм „FIFO”</td><td>Вартість від старту $\hat{g}(n)$</td><td colspan="2">Вартість до мети $\hat{h}(n)$</td><td>Вартість від старту до мети $\hat{f}(n) = \hat{g}(n) + \hat{h}(n)$,</td></tr>
<tr><td rowspan="2">Випадковий пошук (Random)</td><td rowspan="2">Пошук в глибину (Depth)</td><td rowspan="2">Пошук в ширину (Breadth)</td><td rowspan="2">Алгоритм рівних цін (Optimal search)</td><td rowspan="2">Алгоритм підйому на гору, (Hill climbing)</td><td rowspan="2">Глобальне врахування відповідності цілі (Best-First)</td><td>Алгоритм А* $\hat{h}(n) \le h(n)$</td></tr>
<tr><td>Алгоритм А* з обмеженням монотонності $\hat{h}(n_i) \le \hat{h}(n_j) + c(n_i, n_j)$</td></tr>
</table>

Пошук розв'язків у просторі станів може виконуватися в різних напрямках: від початкового стану до цільового стану (пошук, керований даними); від цільового стану до початкового стану (пошук, керований метою, або зворотний пошук); одночасний пошук у двох напрямах (комбінований пошук). Оскільки пошукові графи (дерева) для практичних задач частіше за все є дуже великими, то двонаправлений пошук може виявитися більш ефективним за однонаправлений [11].

Пошук розв'язків в просторі станів є комбінаторною задачею, оскільки рішення отримують на основі перебору всіх станів. Проте для

багатьох практичних задач здійснити повний перебір неможливо. Для оцінки ступеня складності вирішуваної задачі використовують спеціальні методи. Наприклад, за таку оцінку беруть ступінь розгалуження вершин графа стану, який дорівнює середньому числу дочірніх вершин, що породжує один стан задачі. При цьому залежність загальної кількості станів N від ступеня розгалуження B і глибини дерева L визначається формулою [4]:

$$N = B(B^L - 1)/(B-1).$$

Підрахуємо, наприклад, загальну кількість станів для гри-головоломки „8", яка є спрощеним варіантом гри „15". У грі використовуються 8 фішок, пронумерованих від 1 до 8 (рис. 2.7). Фішки розташовуються у дев'яти комірках, що утворюють матрицю розміром 3×3. Одна з комірок завжди є пустою. Хід полягає у пересуненні будь-якої фішки, суміжної з пустою коміркою, у позицію, що відповідає пустій комірці. Необхідно знайти мінімальну послідовність ходів, яка переводить гру з початкового стану до кінцевого стану, який визначається заздалегідь заданою конфігурацією фішок.

Підрахуємо можливі варіанти переміщення для фішок, що знаходяться у різних позиціях:

- кутова фішка - два переміщення, тобто для чотирьох кутів вісім переміщень;
- центральна позиція крайнього рядка - три переміщення, всього дванадцять переміщень;
- центральна позиція - чотири переміщення.

4	7	3
2	1	8
5	*	6

а)

⇨

1	2	3
4	*	5
6	7	8

б)

Рисунок 2.7 – Гра головоломка „8":
а) поточний ігровий стан; б) цільовий стан

Загальна кількість можливих переміщень складе 24. Розділивши це число на 9 (кількість різних можливих положень для порожньої комірки), отримаємо ступінь розгалуження 2,67. Тепер можна визначити загальну кількість станів при перегляді дерева станів глибини L (табл. 2.3) [11].

Як видно з таблиці 2.3, навіть для такої простої задачі кількість станів, що переглядаються, різко зростає зі збільшенням глибини пошуку.

Таблиця 2.3 - Кількість станів у грі „8”

L	2	3	4	5	6	7	8	9	10
N	10	29	80	215	577	1545	4127	11024	29436

Складність вирішуваної задачі можна також оцінювати розмірами списків OPEN і CLOSED. Для того, щоб ці списки не розросталися, можна зберігати в списку OPEN тільки кілька найбільш „перспективних” станів: це скоротить простір пошуку, але виникне небезпека пропуску рішення. Такі стратегії пошуку з обмеженням на розмір списку OPEN, відповідають променевому пошуку (beam search). Розглянуті вище стратегії евристичного пошуку є більш безпечнішими. Чим більшою є інформованість про вирішувану задачу, тим більшою є вага евристичної частини оціночної функції і тим більше скорочується простір пошуку. Слід звернути увагу на те, що й обчислення евристик вимагає певних витрат. При цьому виникає питання мінімізації загального часу, затрачуваного на пошук розв’язку [6].

2.4.2 Пошук з розповсюдженням обмежень

Існують задачі, в яких цілеспрямований рух здійснюється завдяки урахуванню обмежень на переходи між станами. Як приклади можна навести задачі розпізнавання мови або інтерпретації тривимірних контурних рисунків.

Одна з технологій розпізнавання мови основана на розбитті мовного сигналу на елементарні ділянки з наданням кожній ділянці мітки у вигляді фонеми. Послідовність фонем, одержувана в результаті такої розмітки, повинна бути злагодженою і не суперечити правилам вимовлювання слів. Обмеження на можливі взаємні поєднання фонем, дозволяють виключити з розгляду варіанти, які не формують акустичні аналоги слів. Таким чином, елементарні ділянки мови, подані фонемами, забезпечують формування правильного слова, а послідовність слів - граматично коректну фразу. В цьому значенні обмеження, що накладаються, дозволяють відновлювати ціле по елементах, і їх часто називають обмеженнями цілісності.

У разі участі однієї змінної одночасно в кількох обмеженнях, утворюються мережі (системи) обмежень. Рішення задачі в цьому випадку полягає у визначенні всіх поєднань значень змінних, які задовольняють обмеження цілісності [10].

Розглянемо задачу злагодженої розмітки (consistent labeling), при розв’язуванні якої аналізується m елементів, пронумерованих $n_1,,n_2...,n_m$ [1]. Припустимо, що кожний елемент n_i може бути помічений (інтерпретований) декількома способами. Кількість можливих інтерпретацій кожного елемента задається числами b_1, b_2, b_m. Якщо розглядати можливі інтерпретації всієї сукупності елементів і перевіряти їх на дотримання обмежень цілісності (тобто з'ясовувати чи відображають вони допустимі

рішення), то необхідно проаналізувати $b_1 \times b_2 \times ... \times b_m$ випадків. Хоча дана задача легко подається графом станів, доцільніше все ж таки перейти до іншого подання – до графа, вершини якого є елементами n_i, а ребра відповідають бінарним відношенням між відповідними елементами.

Нехай для кожного з елементів $n_i \in [n_1, n_2.., n_m]$ існує кінцева множина інтерпретацій (міток) L_i, потужністю G_i. Можливість або неможливість злагодженої інтерпретації двох елементів n_i, і n_j за допомогою міток $l_i \in L_i$ і $l_j \in L_j$ задається бінарним відношенням b_{ij}. Якщо певна мітка l_i, для елемента n_i, не узгоджується з відповідною міткою l_j для елемента n_j, (тобто не задовольняє відношення b_{ij}), то вона вилучається з множини L_i. Аналогічним чином досліджуються всі мітки, що входять до множину L_j. У ході цього процесу з кожною вершиною графа пов'язується мітка (або множина міток). При цьому мітки на кінцях ребер графа повинні задовольняти задані відношення.

Розглянемо приклад узгодженої розмітки вершин графа, наведеного на рис. 2.8 [9]. Біля вершин графа вказані множини можливих міток, а обмеження, які повинні задовольняти відповідні мітки, записані поруч з ребрами. Основна ідея полягає у звуженні множини можливих міток для кожної з вершин на основі заданих обмежень.

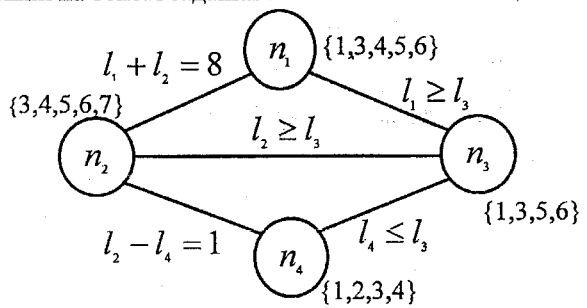


Рисунок 2.8 - Узгоджена розмітка вузлів графа

Розглянемо множину сумісних міток для вершин n_1 і n_2 при обмеженні $l_1+l_2=8$. Спочатку при заданій множині $L_1=\{1,3,4,5,6\}$ вилучимо з L_2 – всі мітки, що не задовольняють вказану умову. В даному випадку відношення b_{12} виконується для всіх міток з L_2, за винятком мітки 6. Отже, дана мітка вилучається, і $L_2 = \{3,4,5,7\}$. Потім відношення b_{12} використовується для перевірки множини міток L_1. У цьому випадку з L_1 також віддаляється мітка 6, і множина L_1 складатиметься з міток $\{1,3,4,5\}$. Таким чином, обмеження b_{12} спочатку розповсюджується з вершини n_1 на вершину n_2, а потім назад з n_2 на n_1, вилучаючи відповідні мітки з множин L_2 і L_1. Внаслідок виконання цих операції, мітки, що входять до множин L_2 і L_1, стають узгодженими.

На наступному кроці для аналізу вибирається вершина, пов'язана з найбільшою кількістю вершин, вже розглянутих на дану мить. У нашому прикладі це вершина n_3. Розповсюдивши обмеження від n_1 до n_3 і від n_2 до n_3, отримаємо L_3={1,3,5}. Далі відповідні обмеження розповсюджуються у зворотний бік. При цьому множини L_2 і L_1 не змінюються. Нарешті, аналогічні операції виконуються з вершиною n_4. У результаті отримаємо: L_4 = {3,4}, L_2 = {4,5}, L_3={3,5}. Розповсюдження обмежень між n_2 і n_3 приводить до L_3={3}, а від n_2 і n_3 до n_1 - до L_1 = {3,4}. Розповсюджуючи обмеження назад від n_1 до n_2, n_3 і n_4 одержуємо остаточний результат: l_1 = 4, l_2= 4; l_3= 3, l_4= 2. Розповсюдження обмежень здійснюється доти, доки на множині міток відбуваються зміни.

В загальному випадку вершина може мати декілька міток. При цьому довільно комбінувати мітки для різних вершин не можна. Необхідно враховувати існуючі обмеження. Очевидно в цьому випадку задача має декілька розв'язків.

2.4.3 Технологія побудови оціночної функції

Розглянемо технологію побудови оціночної функції на кількох прикладах, що відображають різні проблемні області.

Гра „хрестики – нулики"

На дошку, поля якої утворюють матрицю розміром 3×3, кожен з гравців по черзі виставляє свою фішку (рис. 2.9). Фішки гравця X помічені хрестиками, а гравця O – нуликами. Виграє той з гравців, хто першим повністю заповнить своїми фішками будь-який рядок, стовпець або діагональ матриці.

	X	O
	X	
	O	

⇨

O	X	O
	X	
X	X	O

а) б)

Рисунок 2.9 – Гра „хрестики - нулики":
а) поточний ігровий стан;
б) приклад одного з цільових станів для гравця X

Не дивлячись на простоту правил гри, здійснення вичерпного пошуку з використанням повного перебору всіх можливих станів потребуватиме виконання дуже великої, хоча і досяжної для сучасних комп'ютерів, кількості операцій. На кожен з перших дев'яти ходів можна відповісти одним з восьми ходів, потім одним з семи і т .д. Отже, загальна кількість можливих станів складе 9!. Урахування симетричності конфігурацій ігрової дошки свідчить, що насправді існує не дев'ять можливих ходів, а лише три: у кут дошки, у центральне поле крайнього

рядка (стовпця) та у центральне поле дошки (усі інші ходи перетворюються на описані відповідним поворотом дошки). Редукція на другому і наступних рівнях зменшує розмірність простору станів до значення 12×7!.

Подальше зменшення простору станів може бути досягнуто використанням евристичної функції. За основу міркувань при побудові оціночної функції для ігрової ситуації можна використати ціль гри, її правила, методичні рекомендації провідних гравців, власний досвід гри і т. ін. Тобто, треба або здійснити глибокий теоретичний аналіз правил гри, або просто пограти в неї і спробувати формалізувати причини, з яких в тій або іншій ситуацій Ви надаєте перевагу тому або іншому ходу.

а) *якісне визначення евристики для побудови евристичної функції.* У нашому випадку задача полегшується доступністю експертного досвіду гри у „хрестики – нулики", який свідчить, що перший хід доцільно робити у центральне поле дошки. Отже, необхідно з'ясувати причини ефективності цього ходу і формалізувати їх, щоб виробити правила, за якими можна буде обчислювати конкретні значення оцінок як цього, так і інших станів гри. Дійсно, у чому полягає причина того, що найкращим першим ходом здається розміщення фішки у центральному полі дошки, трохи гіршим, але теж популярним – розміщення фішки у кутовому полі, і найменш популярним є хід у середнє поле крайнього рядка (стовпця) дошки?

В нашому випадку відповідь отримується дуже просто. Очевидно, що вибір ходу обумовлюється бажанням забезпечити максимально можливу різноманітність варіантів подальшого продовження гри. Отже, поточний стан здається нам тим привабливішим, чим більшу кількість варіантів досягнення одного з цільових станів він надає.

б) *визначення кількісної оцінки вибраної евристики.* Відповідно до вибраної стратегії оцінки станів, кількісну оцінку поточного стану доцільно визначити як потужність множини варіантів досягнення з поточного стану цільового. Принцип обчислення значення оцінки ілюструється на рис.2.10.

На рис.2.10 переривистими лініями позначені комбінації полів, заповнення яких веде до цільової (виграшної) ситуації.

Найкращою є оцінка ходу у центральне поле дошки.

в) *пошук розв'язку у просторі станів.*

Рис. 2.11 ілюструє перші три кроки евристичної процедури пошуку рішення у просторі станів гри „хрестики - нулики".

Вибір чергового ходу здійснюється відповідно до максимальної оцінки наступного стану. При рівності оцінок кількох станів, для переходу вибирається будь-який з них. Для першого ходу вибираємо центральне поле дошки. Зауважимо, що при цьому з подальшого розгляду вилучаються не тільки два інших можливих ходи, але й усі їх нащадки.

Отже, вже після першого ходу простір станів зменшується на 2/3. Після першого ходу опонент може обрати лише два ходи (усі інші ходи є симетричні). Після цього наступний хід може бути знов обраний за допомогою оціночної функції (один зі станів з оцінкою $\hat{f}(n)=5$). Таким чином, завдяки вибору найкращого ходу серед можливих (з вилученням з подальшого розгляду не тільки інших варіантів ходу, але й множини усіх станів, породжуваних відкинутими станами), евристичний пошук забезпечує на кожному ході суттєву редукцію простору станів.

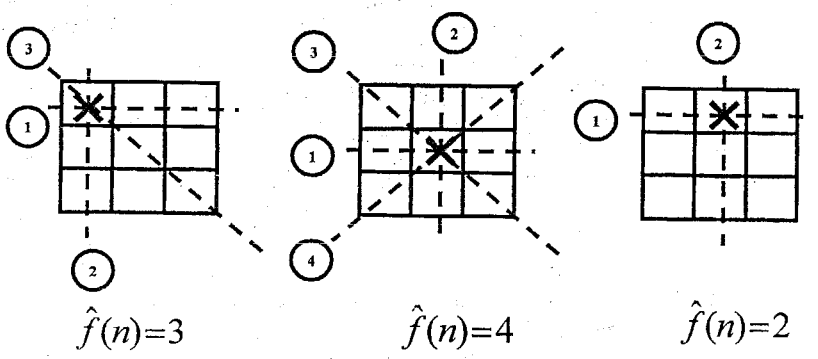


Рисунок 2.10 – Застосування евристики „найбільша кількість виграшних ліній" до оцінки першого ходу у грі „хрестики - нулики"

До того ж, дошка заповнюється у процесі гри і з кожним ходом кількість можливих відповідей зменшується. Більш того, опонент робить лише половину усіх можливих ходів. Таким чином, груба верхня межа складе 9×8=72 стани, що на чотири порядки менше за величину 9!.

Гра „8" (рис. 2.7).

Середня вартість розв'язку у цій грі для випадково сформованих початкових станів складає близько 22 ходів, коефіцієнт розгалуження – близько 3. Тобто, при вичерпному пошуку на глибину 22 необхідно переглянути приблизно $3^{22}\approx 3.1\times 10^{10}$ станів. З урахуванням симетрії цю кількість можна скоротити приблизно у 170000 разі, оскільки існує лише 9!/2=181440 різних станів, що є досяжними. Для гри „п'ятнадцять" (дошка 4×4), кількість досяжних станів складе близько 1,3 трильйона, а для гри „двадцять чотири" (дошка 5×5) – 10^{25}, і її й досі не просто розв'язати для випадково вибраних початкових станів, навіть з використанням сучасних комп'ютерів і алгоритмів [2].

а) *якісне визначення евристики для побудови евристичної функції.* Для пошуку розв'язків з використанням алгоритму А*, необхідно вигадати евристичну функцію, яка б не переоцінювала кількість кроків досягнення

цілі, але максимально наближалася б до точного її значення. Для даного випадку сформулювати певні евристичні правила оцінки, виходячи лише з досвіду гри, виявляється складніше ніж у попередньому прикладі.

Отже, можна виходити при цьому з аналізу цільового стану (відмінностей між поточним та цільовим станом), або з аналізу правил гри.

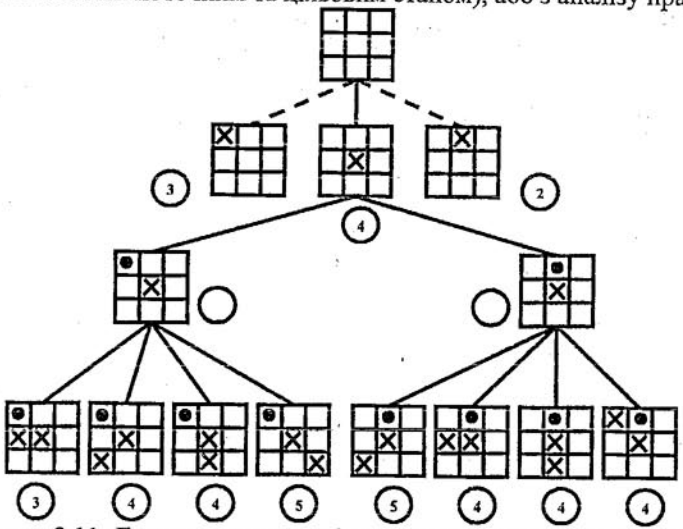


Рисунок 2.11- Евристична редукція простору станів у грі „хрестики - нулики"

Звернемось до рис. 2.12, на якому зображено три ігрових та цільовий стан гри.

2	1	6
4		8
7	5	3

а)

2	8	3
1	6	4
7		5

б)

8	1	2
7	6	3
	5	4

в)

1	2	3
8		4
7	6	5

г)

Рисунок 2.12 – Поточні ігрові (а - в) та цільовий (г) стани гри „8"

Задамо питання, яка з трьох поточних ситуацій (рис. 2.12, а – рис. 2.12, в) здається нам кращою (простішою з точки зору досягнення цільової ситуації)? Не треба багато часу, щоб дати тверду відповідь, що це ситуація рис. 2.12, б. Чому? Тому що від неї можна легко знайти шлях до цільової ситуації без допомоги комп'ютера і, навіть, без паперу і олівця. Не набагато складнішою виглядає ситуація, зображена на рис. 2.12, в. Чим обумовлюється „простота" цих ігрових ситуацій? Тим, що вони є найбільш

схожими на цільову. Тобто, якісно евристика має надавати міру подібності поточної ситуації до цільової.

б) *визначення кількісної оцінки вибраної евристики.* Але яким чином можна кількісно виміряти відмінність між ситуаціями у грі „8"? Уявивши собі опис стану гри у вигляді дев'ятирозрядного вектора, можемо переформулювати питання на інше: „Як кількісно виміряти відмінність між двома векторами?", яке здається більш знайомим і простим. Дійсно, який з векторів (рис. 2.13, а) А2-А4 більше схожий на еталонний вектор А1 і як кількісно можна виразити різницю між ступенями подібності векторів А2-А4 до вектора А1?

А1:	1	2	3	4
А2:	1	2	2	4
А3:	1	2	2	2
А4:	2	2	2	2

а)

Б1:	2	1	6	4	0	8	7	5	3
Б2:	2	8	3	1	6	4	7	0	5
Б3:	8	1	2	7	6	3	0	5	4
Б4:	1	2	3	8	0	4	7	6	5

б)

Рисунок 2.13. – Визначення кількісної міри подібності векторного подання станів гри „8"

Відповідь здається очевидною. Найбільш подібним до А1 є вектор А2, оскільки вони відрізняються значенням лише однієї координати. Найменш подібним до А1 є вектор А4, який відрізняється від А1 значеннями трьох координат. Отже, кількісною мірою відмінності векторів може служити кількість координат, значення яких не збігаються. Аналог цього висновку стосовно нашої задачі може бути такий: „кількісною мірою відмінності поточної ситуації від цільової (значенням евристичної функції $\hat{h}_1$) для гри „8" може служити кількість фішок, місце розташування яких у поточному стані не збігається з місцем їх розташування у цільовому стані".

Порівнявши вектори Б1-Б3 з еталонним вектором Б4 (рис. 2.13, б) бачимо, що найвище значення ступеня подібності до вектора Б4 має вектор Б2, значення якого можна кількісно оцінити числом 4, відповідно до кількості координат, що мають в обох векторах однакові значення – $\hat{h}_1(Б2)=4$. Аналогічно, для векторів Б1 і Б3 отримаємо: $\hat{h}_1(Б1)=2$, $\hat{h}_1(Б3)=0$.

Отже, як значення першої евристичної функції виберемо кількість фішок, розташованих у поточній ситуації „не на своїх" місцях (відносно ситуації цільової). Зауважимо, що ця оцінка задовольняє вимогу $\hat{h}(n) \le h(n)$, оскільки кожна фішка, місце розташування якої у поточній ситуації не відповідає місцю її розташування у цільовій ситуації, має зробити в процесі пошуку розв'язку хоча б один хід.

Однак, ситуація рис. 2.13, в виглядає набагато привабливішою за ситуацію рис. 2.13, а, хоча запропонована нами оцінка $\hat{h}_1(Б3)$ має менше значення за оцінку $\hat{h}_1(Б1)$: $\hat{h}_1(Б1) > \hat{h}_1(Б3)$. Тобто, запропонована евристична функція не враховує якісь значущі фактори. Спробуємо розібратися, які ж саме?

Перш за все, наша задача не дозволяє змінити значення координати вектора на потрібне за один крок. Тобто, для пересування фішки на місце з цільової ситуації може знадобитися один крок, а може й набагато більше, що не враховується запропонованою евристичною функцією. Яким же чином враховують відстань між векторами у n-вимірному просторі? Існують різні формули для цього, але спільним в них є те, що загальна відстань є певною інтеграцією відстаней по окремих координатах вектора. Отже, в оціночній функції доцільно підрахувати не тільки кількість фішок, що знаходяться не на своїх місцях, але й суму відстаней кожної фішки від своєї еталонної позиції. Оскільки фішки не можуть пересуватися по діагоналі, розрахунок відстані являє собою суму горизонтальних та вертикальних відстаней. Таку відстань називають відстанню, вимірюваною у міських кварталах, або манхетенською відстанню. Запропонована евристична функція $\hat{h}_2(n)$ також задовольняє умову $\hat{h}(n) \le h(n)$, оскільки за один хід можна пересунути одну фішку лише на одну позицію ближче до цілі.

Для поточних станів рис. 2.13 отримаємо такі значення оціночної функції: $\hat{h}_2(Б1) = 12$; $\hat{h}_2(Б1) = 5$; $\hat{h}_2(Б1) = 8$. Як бачимо, вони дійсно збігаються з нашою неформальною оцінкою поточних станів.

Розглянемо інший, більш формалізований підхід до побудови евристичних функцій алгоритму А*.

Звернемо увагу, що отримані евристичні функції $\hat{h}_1(n)$ і $\hat{h}_2(n)$ є оцінками довжини шляху, який ще треба подолати для розв'язування задачі гри „8". Але вони, крім того, повертають ідеально точні значення довжини шляху для спрощених версій цієї гри. Якби правила гри „8" змінилися так, щоб будь-яку фішку можна було пересувати куди завгодно, а не тільки на сусідній порожній квадрат, то евристична функція $\hat{h}_1(n)$ повертала б точну кількість кроків найкоротшого розв'язку. Аналогічно, якби будь-яку фішку можна було переміщати на один квадрат в будь-якому напрямку, навіть якщо він зайнятий, то точну кількість кроків найкоротшого розв'язку повертала б евристична функція $\hat{h}_2(n)$. Задача з меншою кількістю обмежень на можливі дії називається послабленою задачею. Вартість оптимального рішення послабленої задачі завжди може служити допустимою евристикою для первинної задачі. Така евристична функція є допустимою, оскільки оптимальний розв'язок первинної задачі, за означенням, служить також розв'язком більш простої задачі і тому

повинний мати не меншу вартість ніж її оптимальний розв'язок. Оскільки значення похідної евристичної функції є точною вартістю розв'язування більш простої задачі, ця функція повинна бути монотонною. Опишемо припустимі у грі „8" дії:

Фішка може бути пересунена з квадрата А до квадрата В, якщо квадрат А є суміжним по горизонталі або по вертикалі з квадратом В, і квадрат В є порожнім. Бачимо, що з цього опису можна сформувати три більш прості задачі шляхом вилучення однієї або обох з наведених вище умов:

а) фішку можна пересувати з квадрата *A* до квадрата *B*, якщо квадрат *A* є суміжним з квадратом *B*;

б) фішку можна пересувати з квадрата *A* до квадрата *B*, якщо квадрат *B* порожній;

в) фішку можна пересувати з квадрата *A* до квадрата *B*.

На підставі більш простої задачі (а) можна вивести функцію $\hat{h}_2$ (манхетенська відстань). В основі цих міркувань лежить те, що $\hat{h}_2$ має бути правильною оцінкою, якщо кожна фішка пересувається до місця її призначення по черзі. На підставі послабленої задачі (в) можна отримати евристичну функцію $\hat{h}_1$ (фішки, що стоять не на своїх місцях), оскільки ця оцінка була б правильною, якби фішки можна було пересувати в призначене для них місце за один етап. На підставі послабленої задачі (б) можна сформувати евристичну функцію не гіршу за $\hat{h}_1$, а в деяких ситуаціях навіть й кращу за $\hat{h}_2$.

Одна з проблем, що виникають при складанні нових евристичних функцій, полягає в тому, що часто не вдається отримати евристичну функцію, яка була б „кращою у всіх відношеннях" порівняно з іншими. Якщо для розв'язування будь-якої задачі може застосовуватися ціла колекція допустимих евристичних функцій $\hat{h}_1 \ldots\ldots \hat{h}_n$ і жодна з них не домінує над будь-якою з інших функцій, то яку з цих функцій треба вибрати? Виявилося, що такий вибір робити не треба, оскільки можна взяти від них всіх найкраще, визначивши такий критерій вибору:

$$\hat{h}(n) = \max\{\hat{h}_1(n) \ \ldots\ldots \ \hat{h}_n(n)\}.$$

У цій складеній евристиці використовується та функція, яка є найточнішою для даного вузла. Оскільки евристичні функції, що входять до складу евристики h, є допустимими, сама ця функція також є допустимою і монотонною.

Зауважимо, що при визначенні евристичної сили алгоритму впорядкованого пошуку саме вибір евристичної функції $\hat{h}(n)$ відіграє вирішальну роль. Використання $\hat{h} \equiv 0$ гарантує знаходження шляху до цільової вершини, але призводить до надто неефективного сліпого перебору. Вибір за $\hat{h}$ найбільшої з можливих нижніх границь h призводить до розкриття

найменшої кількості вершин при збереженні гарантії знаходження шляху мінімальної вартості до цілі (якщо такий шлях існує).

Але часто евристична сила алгоритму може бути підвищена за рахунок відмови від гарантій знаходження шляху мінімальної вартості при виборі такої функції $\hat{h}$, яка не є нижньою границею для h. Така функція виявляється здатною вирішувати набагато складніші задачі.

Прикладом такої функції для гри „8" є така евристична функція:

$$\hat{h}_3(n) = P(n) + 3S(n),$$

де $P(n)$ – сума манхетенських відстаней кожної фішки до свого місця у цільовій ситуації;

$S(n)$ - кількість балів, які враховують порядок розташування фішок: за фішку, що знаходиться у центральному полі дошки нараховується один бал, а за кожну фішку яка передує не тій фішці, що має йти за нею у цільовій ситуації, нараховується два бали. Ця функція не надає гарантованого знаходження шляху мінімальної вартості, але чудово працює для гри „8".

в) *пошук розв'язку у просторі станів.*

Розглянемо ефективність запропонованих евристичних функцій при пошуку розв'язку задачі гри „8" у просторі станів. У таблиці 2.4 [11] наведені дані про кількість розкритих в процесі пошуку вершин і значення коефіцієнта розгалуження для випадково сформованих 1200 початкових станів гри „8" з довжиною розв'язку від 2 до 24 (по 100 екземплярів для кожного парного значення довжини). Зауважимо, що вже при довжині пошуку в 14 кроків, застосування евристичної функції $\hat{h}_2$ забезпечує у 30000 раз вищу сфективність у порівнянні з нсінформованими процедурами пошуку.

Таблиця 2.4 – Порівняння ефективності евристичних функцій $\hat{h}_1(n)$ і $\hat{h}_2(n)$

D		2	4	6	8	10	12	14	16	18	20	22	24
Кількість розкритих вершин	A($\hat{h}_1$)	6	13	20	39	93	227	539	1301	3056	7276	18094	39135
	A($\hat{h}_2$)	6	12	18	25	39	73	113	211	363	676	1219	1641
Коефіцієнт розгалуження	A($\hat{h}_1$)	1,79	1,48	1,34	1,33	1,38	1,42	1,44	1,45	1,46	1,47	1,48	1,48
	A($\hat{h}_2$)	1,79	1,45	1,30	1,24	1,22	1,24	1,23	1,25	1,26	1,27	1,28	1,26

На рис. 2.14 наведено дерево перебору, сформоване з використанням евристичних функцій $\hat{h}_1(n)$ і $\hat{h}_2(n)$. Як можна побачити з рисунку, функція $\hat{h}_2(n)$ є більш ефективною, оскільки має більшу роздільну здатність.

На рис.2.15 наведене дерево перебору евристичної функції $\hat{h}_3$. Зна-

чення оціночної функції кожної вершини розміщено у колі, цифри у квадратах вказують порядок, у якому розкривалися вершини. Шлях розв'язування має довжину 18, хоча оскільки $\hat{h}_3$ не є нижньою межею для $\hat{h}$, знаходження оптимального шляху не було гарантоване. Зауважимо, що функція $\hat{h}_3$ забезпечує найбільш спрямований пошук з невеликою кількістю розгалужень, сконцентрованих біля початкової вершини.

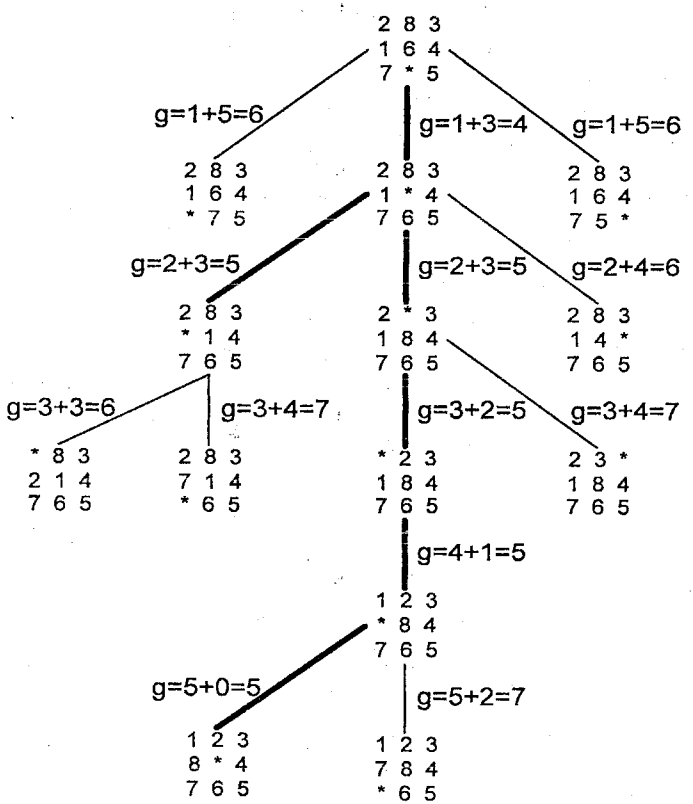


Рисунок 2.14 – Дерево пошуку, побудоване з використанням евристичних функцій $\hat{h}_1(n)$ і $\hat{h}_2(n)$

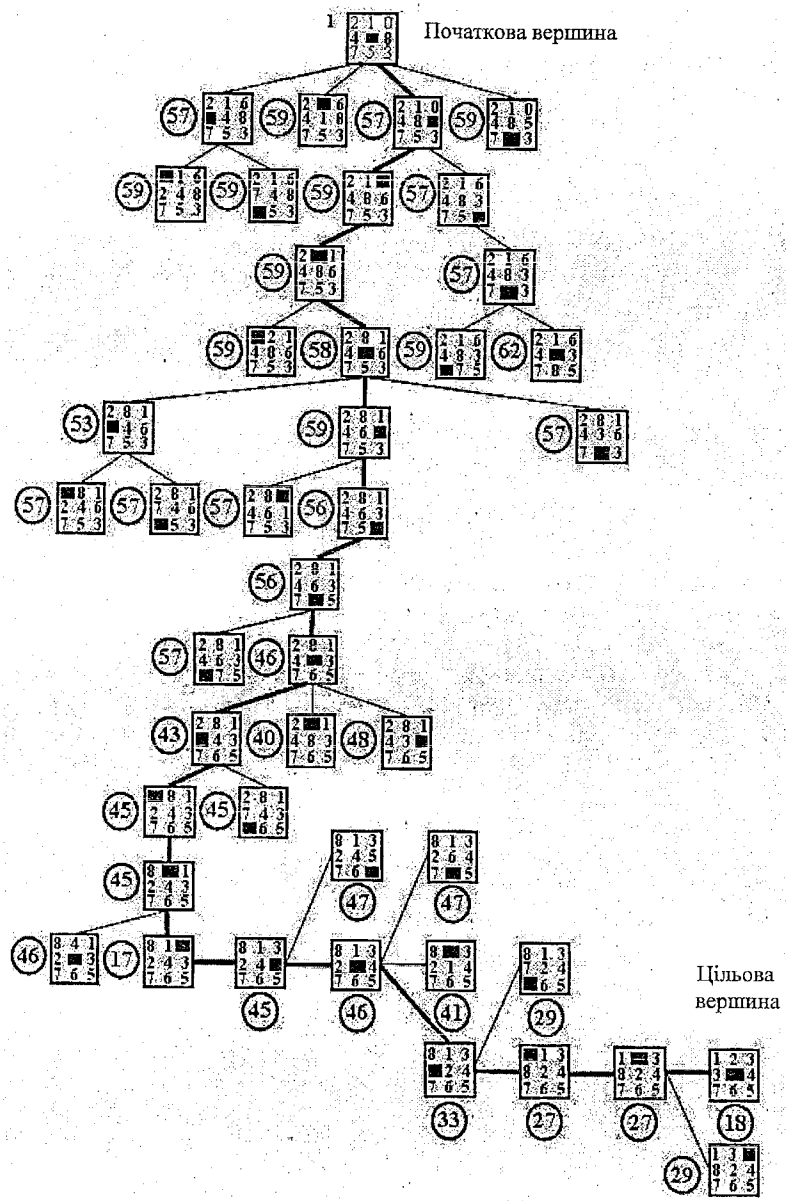


Рисунок 2.15 – Дерево пошуку, побудоване з використанням евристичної функції $\hat{h}_3$

2.5 Контрольні питання

1. Сформулюйте загальну постановку задачі пошуку розв'язку у просторі станів.
2. Наведіть означення евристики та сформулюйте декілька евристичних правил для різних задач.
3. Визначте основні властивості алгоритму А* та визначте умови їх існування.
4. Доведіть спроможність алгоритму А*.
5. Визначте поняття інформованості алгоритму А*. Проаналізувати два алгоритми А* різного ступеня інформованості.
6. Визначте поняття монотонного обмеження. Проаналізувати властивості алгоритму А*, який задовольняє монотонне обмеження.
7. Проаналізуйте евристичну силу функції оцінювання.
8. Проаналізуйте приклади поширених евристик.
9. Наведіть приклади пошуку з розповсюдженням обмежень.
10. Визначте двонаправлений пошук з використанням функції оцінювання.
11. Проаналізуйте евристичні процедури пошуку на графі.
12. Визначте алгоритм А*.
13. Проаналізуйте загальну процедуру пошуку на графах.
14. Визначте поняття дерева пошуку. Проаналізувати процедуру формування дерева пошуку.
15. Проаналізуйте основні властивості функції оцінювання.
16. Проаналізуйте вплив двох складових функції оцінювання для алгоритму А* на характер процесу пошуку.
17. Поясніть з алгоритмічної точки зору функціонування неінформованих процедур пошуку.
18. Поясніть принципи роботи алгоритму рівних цін, його основні переваги та недоліки.
19. Сформулюйте основні властивості А-алгоритму. Порівняйте алгоритми сліпого та спрямованого пошуку по критеріях отримання гарантованого результату та ефективності пошуку.

ЛАБОРАТОРНА РОБОТА № 2
ПОШУК РОЗВ'ЯЗКІВ У ПРОСТОРІ СТАНІВ

Мета роботи

Надбання практичних навичок з визначення евристичних оцінок станів, розробки алгоритмів евристичного пошуку розв'язків задачі у просторі станів і створення відповідного програмного забезпечення.

У результаті виконання лабораторної роботи студент повинен вміти:

- вибирати оптимальний алгоритм пошуку розв'язків задачі у просторі станів;
- визначати найбільш інформативні ознаки об'єкта, явища, процесу з точки зору обчислення евристичних оцінок поточного стану задачі;
- будувати формули для обчислення значень евристичних оцінок поточного стану задачі;
- проектувати алгоритми пошуку розв'язків у просторі станів;
- здійснювати програмну реалізацію алгоритмів пошуку у просторі станів.

Завдання на підготовку

Студент повинен знати:

- основні поняття штучного інтелекту;
- поняття та особливості інтелектуальної задачі;
- форми та методи подання інтелектуальних задач;
- методологію подання задачі у просторі станів;
- мови програмування LISP, PROLOG, C++.

Студент повинен вміти:

- створювати програми мовами LISP, PROLOG, C++.

Для допуску до виконання роботи необхідно:

- вміти відповісти на теоретичні питання щодо ходу виконання роботи;
- показати викладачу заготівку звіту про лабораторну роботу, яка повинна містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1. Побудувати оціночну функцію для пошуку розв'язку узгодженої з викладачем задачі у просторі станів.
2. Написати програму пошуку розв'язку задачі у просторі станів, узгодженою з викладачем мовою програмування

Порядок виконання лабораторної роботи

1. Запропонувати власну задачу (або вибрати задачу з наведеного нижче переліку) для розв'язування її у просторі станів з використанням евристичної функції.

2. Для вибраної задачі:

- визначити найбільш інформативні ознаки, які будуть використовуватися як параметри для оцінювання „перспективності" станів задачі.
- визначити правила оцінювання станів;
- визначити співвідношення для обчислення кількісного значення оцінки станів;
- відповідно до вимог алгоритму A* запропонувати три оціночних функції ($\tilde{h}_1 \le h$; $\tilde{h}_2 > h$, $\tilde{h}_3 >> h$);
- перевірити ефективність кожної із запропоноваших оціночних функцій, застосувавши їх для розв'язування фрагмента задачі;
- порівняти отримані результати і сформулювати відповідні висновки.

3. Графічно відобразити фрагменти графів пошуку, отримані з використанням кожної із запропонованих оціночних функцій, з відображенням значень оцінок для кожної з розкритих вершин.
4. Узгодити з викладачем мову програмування (LISP, PROLOG, C++, JAVA, Pascal) і написати програму для моделювання роботи запропонованих оціночних функцій, яка повинна забезпечувати можливості:

- перегляду бази знань (операторів переходів між станами)
- покрокового виведення на екран кожного з наступних можливих станів в процесі здійснення пошуку з відображенням значення його оцінки;
- автоматичного виконання процедури пошуку з відображенням розв'язування задачі у вигляді переліку пройдених станів і значеннями їх оцінок.

Зміст звіту

Теоретична частина

1. Опис умов задачі.
2. Обґрунтування вибору ознак стану для формування трьох оціночних функцій.
3. Обґрунтування вибору правил формування трьох оціночних функції.
4. Побудова співвідношення для обчислення кількісного значення трьох оцінок станів.
5. Фрагменти трьох дерев розв'язку, побудованих з використанням запропонованих оціночних функцій.

Практична частина

1. Схема алгоритму програми пошуку розв'язку з використанням оціночної функції.
2. Приклад роботи програми проілюстрований скріншотами.
3. Інструкція користувача для роботи з програмою.
4. Лістінг програми з коментарями.

Приклади завдань на лабораторну роботу

Запропонуйте оціночні функції для таких ситуацій

1. Загін солдат підходить до ріки. через яку необхідно переправитись. Міст зламано, але на човні вздовж берега катаються двоє хлопчаків. В човні може переправитися лише один солдат або два хлопчики і не більше. Необхідно забезпечити переправу всіх солдат через ріку.

2. В квадраті, який має 16 комірок, треба розмістити 16 літер (4 літери *A*, 4 літери *B*, 4 літери *C*, 4 літери *E*) так, щоб на кожній вертикалі і на кожній горизонталі будь-яка літера зустрічалась тільки один раз.

3. Дано два глечики: перший - з водою ($V = 5$л), другий - пустий ($V = 2$л). Необхідно отримати рівно 1л в глечику ємністю 2л. Воду можна виливати і переливати з одного глечику до іншого.

4. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів.9. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів.

Запропонуйте по три оціночних функції для таких ігрових ситуацій:

1. Гра „п'ятнадцять"
2. "Квадрат, який обертається".
3. "Вовк та вівці".
4. "Сім" , [1д].
5. "Клац" , [1д].
6. "Гонки" , [1д].
7. "Розсада" , [2д].
8. "Поліцейська машина" , [5].
9. "Так-тікс" (гра Хейна), [3д].
10. "Гекс", [4д].
11. "Двокольорові шашки", [5].
12. "Чорний ящик" , [7].
13. "Рума" , [6д].
14. "Гра Гейла" , [4д].
15. "Французька військова гра" , [7].
16. "Тривимірні хрестики-нулики", [7].
17. "Морська битва".
18. "Пентамімо", [3д].
19. "Кутки".

3 ПОШУК РОЗВ'ЯЗКІВ У ПРОСТОРІ ЗАДАЧ

3.1 Загальні принципи зведення задачі до підзадач

Дуже часто розв'язування великих складних задач значно спрощується, якщо вдається помітити, що вони складаються з множини більш дрібних і простих підзадач. Основна ідея такого підходу полягає у поступовому розбитті задачі на підзадачі, які, в свою чергу, також розбиваються на підзадачі, і так продовжується доти, доки початкова задача не буде зведена до елементарних підзадач, розв'язки яких відомі (або не викликають проблем), або поки не буде доведено, що задача не має розв'язку. Процес розбиття (декомпозиції) задачі на підзадачі називається редукцією. Тому подання задачі у вигляді множини підзадач є поданням у системі редукцій, а пошук розв'язку – є пошуком у просторі описів задач. Зауважимо, що розбиття задачі на підзадачі надає перевагу в тому випадку, коли підзадачі є взаємно незалежними, отже й розв'язувати їх можна незалежно одна від одної.

Простір задач подається у вигляді направленого графа, який називається графом редукції задачі або, інакше, І-АБО графом, і містить у собі два типи вершин: кон'юнктивні (вершини типу І) та диз'юнктивні (вершини типу АБО) [1, 7].

Наявність в І-АБО графі вершин двох типів обумовлюється тим, що за його допомогою подаються два типи задач: редукції (декомпозиції) стану і перетворення стану. До вершин типу І застосовуються правила редукції, які здійснюють розбиття задачі на підзадачі, кожна з яких (і перша, і друга, і наступні) мають бути обов'язково розв'язані для отримання розв'язку головної задачі. До вершин типу АБО застосовуються оператори переходу між станами, які визначають, яка самс дія (перша або друга, або інша) має бути застосована до даного стану, щоб просунутися по шляху розв'язання даної підзадачі [5].

Зауважимо, що з кожною вершиною І-АБО графа можна пов'язати певне висловлювання у вигляді булевої функції, тому його називають також пропозиційним графом

На рис.3.1 наведено окремий випадок графа редукції задачі - дерево редукції [1].

Тут *A* - початкова задача, для розв'язання якої необхідно розв'язати підзадачі *B*, *C* і *D*. Те, що вершина *A* є кон'юнктивною, відображено на рис.3.1 дужкою, що зв'язує гілки, які ведуть до підзадач *B*, *C* і *D*. Для розв'язання задачі *B* необхідно розв'язати одну з її підзадач *E* або *F*. Вершина *B* є диз'юнктивною. Граф редукції задачі відрізняється від графа стану тільки тим, що в ньому присутні вершини типу І. В окремому випадку, коли на графі редукції відсутні вершини типу І, І-АБО граф трансформується у граф простору станів. У цьому випадку розв'язок може бути знайдено за допомогою методів пошуку розв'язків у просторі станів.

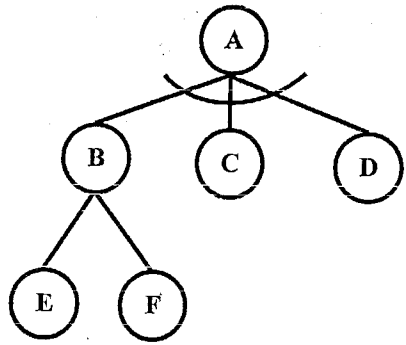


Рисунок 3.1 – Дерево редукції

Початкова вершина (корінь І-АБО дерева) подає початкову задачу, яку необхідно розв'язати. Вершини, що відповідають задачам, розв'язки яких є відомими, називаються кінцевими. Задачі й відповідні їм вершини, що не мають розв'язків, називаються тупиковими. Кінцеві вершини є розв'язуваними, а тупикові - нерозв'язуваними. Розв'язуваними є також такі вершини типу І, у яких розв'язуваними є всі дочірні вершини, та такі АБО-вершини, у яких розв'язуваною є хоча б одна дочірня вершина. Метою пошуку на І-АБО графі є доведення розв'язності початкової вершини. Для цього будується спеціальний граф, який називається графом розв'язку, складається з розв'язуваних вершин і є підграфом графа редукції задачі з початковою вершиною, яка відповідає початковій задачі. Перегляд графа розв'язку в напрямку від кінцевих вершин до початкової вершини визначає множину підзадач, які необхідно розв'язати, і порядок їх розв'язання.

Класичним прикладом задачі, що доцільно подавати І-АБО графом є, наприклад, задача розв'язання невизначених інтегралів. Нехай треба знайти невизначений інтеграл

$$\int sin^2 x \, cos^4 x dx \, .$$

І-АБО граф розв'язку такої задачі наведений на рис 3.2. При знаходженні невизначених інтегралів часто використовують такі оператори, як різноманітні підстановки (заміни), інтегрування по частинах, алгебраїчні перетворення, тригонометричні перетворення і т. ін. За допомогою таких операторів початкова задача може бути зведена до множини підзадач, наведених на рис.3.2. Декомпозиція початкової задачі виконується доти, доки не будуть знайдені елементарні підзадачі, розв'язки яких є відомими. При отримані таких задач, вони позначаються як розв'язані. Далі виконується спеціальна процедура підстановки отриманих розв'язків у всі можливі підзадачі верхніх рівнів, з

відзначенням, у разі потреби, кожної підзадачі більш високого рівня як розв'язаної. У разі відзначення початкової задачі, вона вважається розв'язаною.

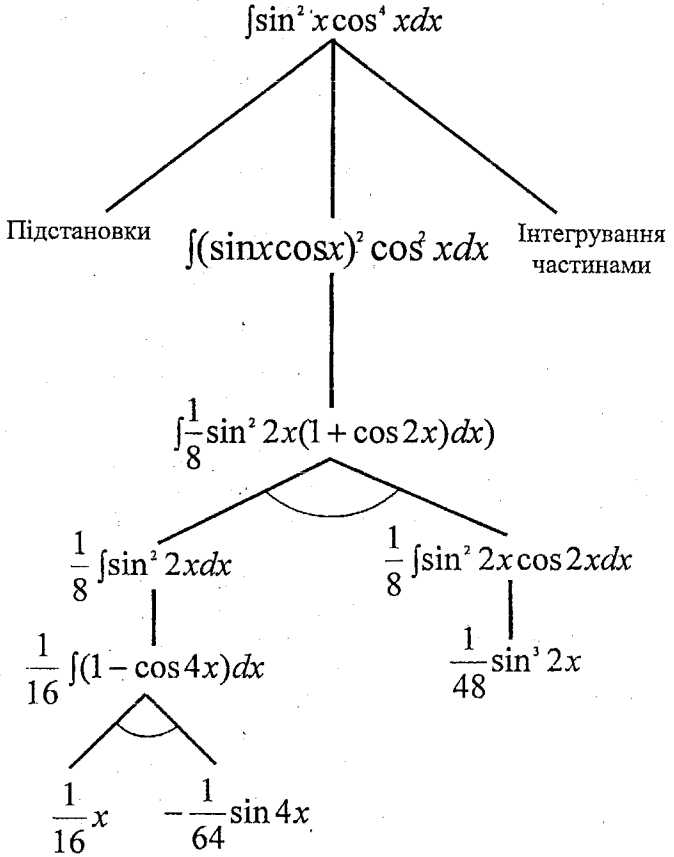


Рисунок 3.2 – Подання задачі обчислення невизначеного інтеграла І-АБО графом

Іншою класичною задачею, що подається з використанням І-АБО графів, є задача формування речень за допомогою формальних граматик. Цікаво відзначити, що кінцевий І-АБО граф може подавати породжуючу контекстно-вільну граматику. В цьому випадку вершини типу І відповідають продукційним правилам, вершини типу АБО – допоміжним символам, кінцеві вершини - термінальним символам.

Розглянемо приклад процесу генерації синтаксично правильних речень англійською мовою як приклад задачі, яка зводиться до розв'язання підзадач.

Задача полягає у формуванні речення відповідно до граматики, заданої сукупністю продукційних правил [1]:

1) S → SUB PRED;
2) SUB → PRON;
3) SUB → NP;
4) PRED → V NP;
5) NP → DET N;
6) PRON → he;
7) PRON → it;
8) DET → a;
9) DET → the;
10) V → saw;
11) V → carried;
12) N → dog;
13) N → man.

У прикладі маленькими літерами позначені термінальні елементи, а великими – нетермінальні. Символ *S* позначає початковий символ. Починаючи з *S*, треба побудувати послідовність з термінальних символів, що подає синтаксично правильні речення. Застосування до *S* правила 1, зводить задачу формування речення *S* до задачі формування символів *SUB* і *PRED*. Тепер задача полягає в тому, щоб перетворити кожний з символів *SUB* і *PRED* на послідовність термінальних символів. Ці підзадачі є простішими за початкову. Застосуванням правил 2, 3 і 4, вони також можуть бути зведені до більш простих підзадач. Продовжуючи вказаний процес, отримаємо дерево редукції задачі, наведене на рис. 3.3.

Штриховими лініями на рисунку показано можливе дерево розв'язку, відповідно до якого формується послідовність з термінальних елементів, що утворюють речення „he carried a dog" (він ніс собаку). За допомогою даної граматики можна сформувати 48 правильних речень: „the man saw a dog", „the dog carried the man" і т. ін. Безумовно, дані речення коректні лише синтаксично. Семантика речень тут не розглядається. Таким чином, дерево розв'язку, подане на рис.3.3, відповідає дереву виведення деякого висловлювання у контекстно-вільній граматиці. Пошук дерев розв'язку може виконуватися із застосуванням методів теорії формальних граматик.

В дереві, зображеному на рис. 3.3, є вершини, що повторюються, наприклад NP. Такі вершини відповідають одним і тим самим підзадачам. При суміщенні двох вершини NP дерево редукції трансформується у граф редукції.

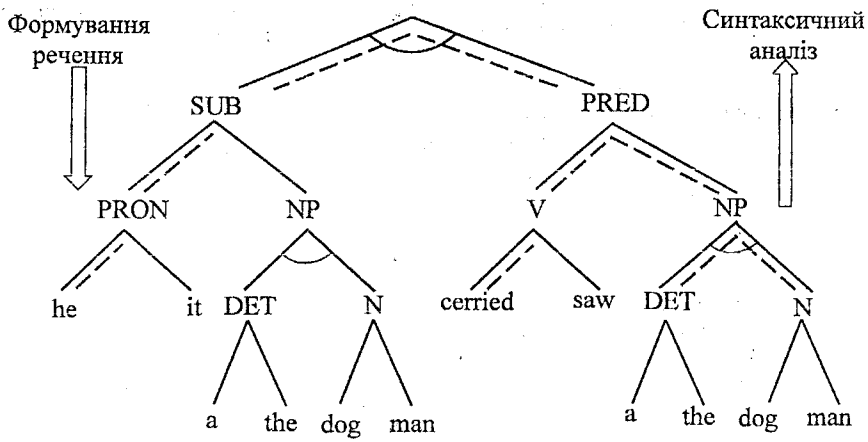


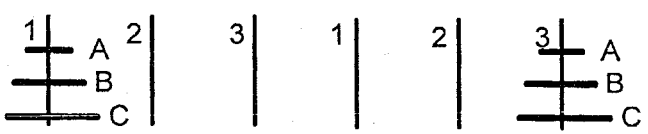
Рисунок 3.3 – Граф редукції задачі формування речень

3.2 Подання задач графом редукції

Розглянемо підхід до розв'язання задачі методом редукції на прикладі головоломки про ханойську вежу [7].

Дано три стержні 1, 2, 3 та три диски різного розміру: A, B, C. Необхідно перекласти диски з першого стержня на третій. На кожному крокові можна перекладати тільки один диск, причому не дозволяється класти більший диск на менший. Диск більшого розміру не може розміщуватися над диском меншого розміру.

Початкова та кінцева конфігурація задачі, наведені на рис. 3.4, а та 3.4, б, відповідно.



а) б)

Рисунок 3.4– Початкова (а) та кінцева (б) конфігурація задачі про ханойську вежу

Таку задачу можна розв'язати методами, які використовують простір станів. Граф простору станів для неї буде мати 27 вершин, кожна з яких відображає одну із дозволених конфігурацій розташування дисків на стержнях. На рисунку 3.4 стани відображаються трирозрядними

векторами, значення першого розряду яких збігається з номером стержня на якому розташований диск С, значення другого розряду – з номером стержня на якому розташований диск В і значення третього розряду – з номером стержня на якому розташований диск А.

Цю задачу можна також розглядати як задачу, яка складається з трьох підзадач:

1. Перекласти диск А на стержень 3;
2. Перекласти диск В на стержень 3;
3. Перекласти диск С на стержень 3.

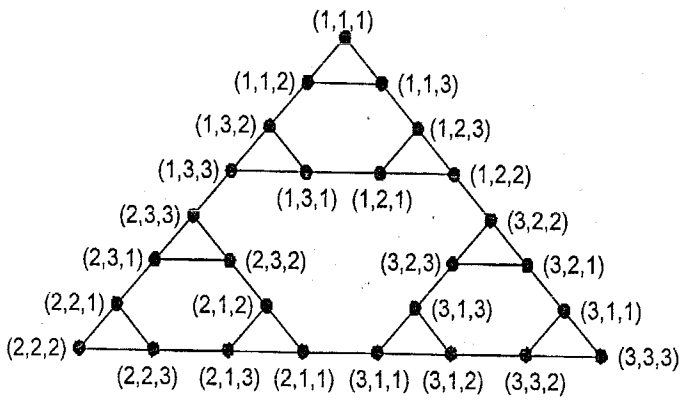


Рисунок 3.5 – Граф простору станів для задачі про ханойську вежу

Зауважимо, що ці підцілі не є незалежними. Наприклад, можна одразу перекласти диск А на стержень 3 і перша підціль буде досягнута. Але в цьому випадку неможливо стане досягнути дві інші підцілі (якщо не відмінити першу дію). Існує і інший порядок досягнення всіх трьох підцілей, який забезпечує розв'язання задачі. Цей порядок відповідає такому ланцюжку міркувань: найбільш складною підціллю є підціль 3 (перекласти диск С на стержень 3), оскільки на диск С накладено найбільшу кількість обмежень. В таких випадках часто спрацьовує така ідея: спробувати досягти першою найбільш складну підціль. Цей принцип засновано на такій логіці: оскільки інші підцілі досягнути простіше (на них накладено меншу кількість обмежень), то є підстави сподіватись, що їх досягнення можливе без відміни дій, які було використано для вирішення найбільш складної задачі. Відносно нашої задачі сказане відповідає такій стратегії: першою досягти підціль "перекласти диск С на стержень 3", а потім - всі інші.

Але першу підціль не можна досягти одразу, оскільки в початковій ситуації диск С перекладати не можна. Отже, необхідно підготувати такий хід. В цьому випадку наша стратегія матиме таку послідовність кроків.

1. Для того, щоб перекласти всі диски на стержень 3, потрібно перекласти туди диск С. При цьому на диску С не повинні знаходитись ніякі інші диски. Тобто, необхідно забезпечити можливість перекласти диск С з стержня 1 на стержень 3.

2. Перекласти диск С з стержня 1 на стержень 3.

3. Досягнути дві підцілі, які залишились (перекласти диски А і В на стержень 3).

Перекласти диск С з стержня 1 на стержень 3 можна тільки в тому випадку, коли інші два диски А і В знаходяться на стержні 2. Таким чином, початкова задача переміщення дисків А, В і С з стержня 1 на стержень 3 перетворюється на три такі задачі:

1. Зняти диски А і В з стержня 1, при цьому розташувати їх не на стержні 3, а на стержні 2.

2. Виконати переміщення диска С з стержня 1 на стержень 3.

3. Перемістити диски А і В з стержня 2 на стержень 3.

Такий хід міркувань дозволяє податити початкову задачу у вигляді трьох незалежних задач:

1. Задача про переміщення двох дисків (А і В) на стержень 2.

2. Задача про переміщення одного диска (С) на стержень 3.

3. Задача про переміщення двох дисків (А і В) на стержень 3.

Бачимо, що кожна з цих задач є більш простішою у порівнянні з попередньою. До того ж друга задача є тривіальною і виконується за один хід. Інші дві задачі можна розв'язувати незалежно від задачі 2, оскільки диски А і В можна переміщати не звертаючи уваги на розташування диска С. Використовуючи аналогічний ланцюжок міркувань задачі 2 і 3 також можливо звести до тривіальних, як це зображено на рисунку 3.6 (саме така ж схема редукції задачі може бути застосована і для початкової конфігурації з будь-якою кількістю дисків).

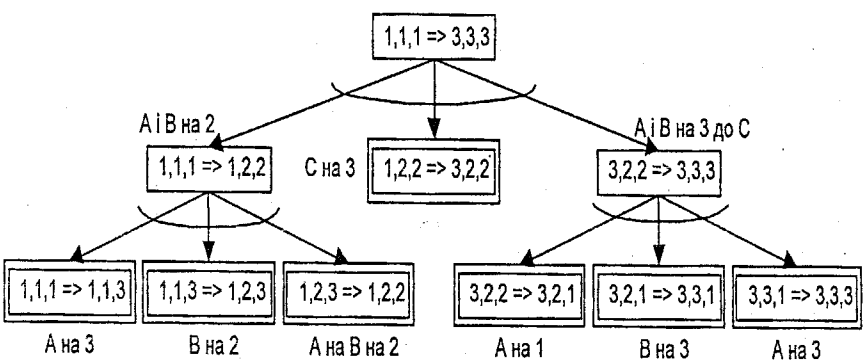


Рисунок 3.7 – Граф зведення задачі про ханойську вежу до підзадач

Відмітимо, що наявність двох типів вершин викликає необхідність розробки власних процедур пошуку розв'язків для І-АБО графів, які відрізняються від процедур пошуку на звичайних графах. Мовою І-АБО графів застосування одиничного оператора редукції задачі означає, що спочатку будується проміжна АБО вершина, а потім безпосередньо наступні за нею І вершини. Вершини І-АБО графа, що відповідають описам елементарних задач, називаються кінцевими або термінальними вершинами.

3.3 Пошук розв'язків при зведенні задач до підзадач

Розглянемо особливості пошуку в І-АБО графах порівняно з пошуком у просторі станів.

У просторі станів розв'язування подається шляхом від початкової вершини до цільової вершини. Розв'язок в І-АБО графах відповідає підграфа або піддереву, оскільки при розкритті І-вершини необхідно визначити, чи є розв'язною кожна з її дочірніх вершин. Наявність АБО-вершини призводить до необхідності розгляду в ході пошуку розв'язку кількох можливих підграфів розв'язку. В разі необхідності знаходження оптимального графа розв'язку використовують оціночні функції. Проте, на відміну від пошуку в просторі станів, ці функції надають вже не вартості шляхів, а вартості графів, зокрема, графа розв'язку або його підграфів.

Процес пошуку породжує підграфи, які виступають в ролі претендентів на участь у формуванні повного графа розв'язку. Називатимемо такі підграфи потенційними підграфами розв'язків. Зазвичай у процесі пошуку будуються декілька потенційних підграфів, які певним чином подаються в комп'ютерній пам'яті. Формування таких підграфів виконують повторенням двох узагальнених дій:

а) визначають, чи є вершини потенційного підграфа розв'язку розв'язними, нерозв'язними (тупиковими) або нерозкритими; у разі необхідності обчислюють вартості відповідних підграфів;

б) розширюють потенційний підграф розв'язків.

В ході виконання першої дії перевіряють усі кінцеві вершини потенційного підграфа. Якщо ці вершини відповідають елементарним задачам, розв'язки яких відомі, то вони вважаються кінцевими і позначаються міткою SOLVED (розв'язні). Якщо кінцеві вершини є тупиковими, то вони позначаються як нерозв'язні (мітка UNSOLVED). Після цього аналізуються їх 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, яка збігається з міткою дочірньої вершини, що розглядається в даний момент.

Якщо початкова вершина потенційного підграфа розв'язку позначається як розв'язувана, то даний потенційний підграф стає можливим підграфом розв'язку.

У разі пошуку оптимального графа розв'язку оцінюється вартість всіх вершин потенційних підграфів розв'язку, починаючи з кінцевих вершин. Правила таких обчислень розглянемо трохи пізніше.

При виконанні другої дії розкривають кінцеві вершини потенційного підграфа розв'язку, які ще не мають міток. Якщо розкрита кінцева вершина є вершиною типу І, то генерується новий потенційний підграф шляхом додавання всіх дочірніх вершин до розширюваного потенційного підграфа. Якщо розкрита кінцева вершина є вершиною типу АБО, то до розширюваного потенційного підграфа додається тільки одна дочірня вершина. При цьому формується стільки потенційних підграфів, скільки є у наявності дочірніх вершин.

Аналогічно пошуку в просторі станів при пошуку в просторі редукції задачі використовують алгоритми сліпого і впорядкованого (евристичного) пошуку.

3.4 Алгоритм пошуку в глибину при редукції задачі

У даному алгоритмі елементами списку OPEN є потенційні підграфи розв'язків, тобто підграфи, початкова вершина яких не має мітки або має мітку SOLVED (підграф розв'язку). В процесі здійснення пошуку знов сформовані потенційні підграфи розв'язків розміщаються на початок списку OPEN.

```
Procedure Depth_First_AO1;
Begin
    Помістити до списку OPEN потенційний підграф, що складається
    лише з початкової вершини S;
    Виконати розмітку даного підграфа;
    While OPEN <> „пустой список" Do
    Begin
        P:= first (OPEN); {вибрати з OPEN перший підграф}
        {застосувати до підграфа P функцію перевірки розв'язуваності
         solved(P))
        IF solved(P) THEN вихід („УСПІХ");
        Вилучити підграф P зі списку OPEN;
        Розширити підграф P і виконати розмітку знов сформованих
        потенційних підграфів рішень;
        Помістити всі нові потенційні підграфи рішень на початок списку
        OPEN;
    End;
    Вихід („НЕВДАЧА")
End.
```

Розглянемо застосування процедури до графа редукції, зображеного на рис. 3.8 [1].

Тут кінцеві вершини, відзначені великими літерами, відповідають нерозв'язним задачам, а вершини, відзначені маленькими літерами, є кінцевими (термінальними).

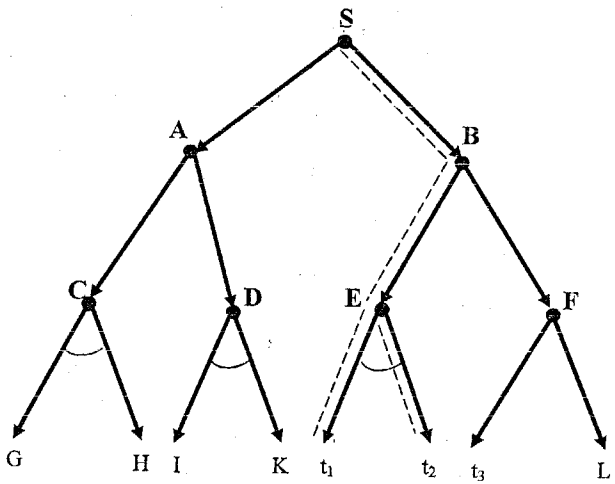


Рисунок 3.8 – Граф редукції задачі

На рис 3.9 подані потенційні підграфи розв'язків, що поміщаються в список OPEN на різних кроках виконуваної процедури. На кроці 4 розширюваний потенційний підграф розв'язку позначається як нерозв'язний і вилучається з подальшого розгляду. Аналогічна ситуація має місце на кроці 5. Надалі розширяється правий підграф вершини *S*, який і призводить до графа розв'язку, наведеного на рис. 3.8 штриховою лінією.

У наведеному прикладі в списку OPEN зберігаються кілька потенційних підграфів розв'язків. Якщо граф редукції задачі є складним, то це призводить до неефективного використання комп'ютерної пам'яті. Для усунення цього недоліку, при розкритті чергової вершини потенційного підграфа розв'язку слід розглядати не всі дочірні вершини, а лише одну. Для цього до процедури пошуку вводиться функція NextChild [2, 8].

Нехай у вершини n є дочірні вершини $n_1, n_2, \ldots, n_i, \ldots, n_m$ і вершина n_i вже включена в потенційний граф розв'язку. Тоді функція NextChild визначиться таким чином

$$NextChild(n) = \begin{cases} n_{i+1}, 0 \le i < m, \\ nil, i = m, \end{cases}$$

де до застосування функції до вершини n має місце i=0.

У цьому випадку процедуру пошуку в глибину можна переписати у такому вигляді:

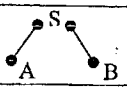
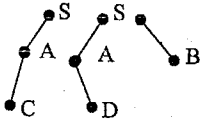
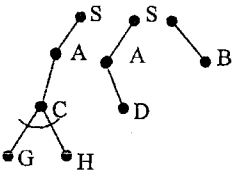
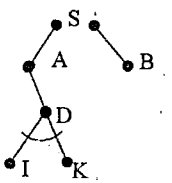
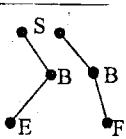
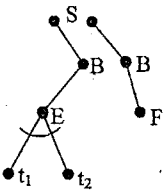
Номер кроку	Список OPEN
1	• S
2	
3	
4	
5	
6	
7	

Рисунок 3.9 – Потенційні графи розв'язку при пошуку в глибину

```
Procedure Depth_first_AO2;
Begin
      Створити потенційний підграф P, що складається тільки з однієї
      початкової вершини  S;
      n:=S;
      Виконати розмітку підграфа P;
      While True Do
      Begin
            If unsolved (P) Then вихід („НЕВДАЧА");
            If solved (P) Then вихід („УСПІХ");
            If n не має мітки Then
               Begin
                  n:=NextChild(n);
                  Побудувати новий потенційний підграф розв'язку,
                  додавши до P вершину n;
                  Виконати розмітку нового потенційного підграфа і
                  позначити його як P;
               End
            Else
               Begin
                  If n має мітку UNSOLVED Then
                  Видалити вершину n з підграфа P;
                  n:= батьківська вершина (n)
               End
      End
End.
```

Якщо описану процедуру застосувати до дерева редукції задачі, зображеного на рис.3.9, то порядок обробки вершин буде таким:

S, A, C, G(UNSOLVED), C(UNSOLVED), A, D, I(UNSOLVED), D(UNSOLVED), A(UNSOLVED), S, B, E, t1(SOLVED), E, t2(SOLVED), E(SOLVED), B(SOLVED), S(SOLVED).

Коли вершина помічається міткою UNSOLVED, вона вилучається з потенційного підграфа розв'язку. Таким чином до підграфа розв'язку включаються тільки ті вершини, що помічені міткою SOLVED.

Кожного разу, коли виявляється нерозв'язна вершина, процедура здійснює повернення до батьківської вершини, забезпечуючи обстеження інших альтернативних гілок графа редукції задачі. Якщо знайдено розв'язувану вершину і при цьому побудову підграфа розв'язків ще не завершено, то повернення виконується з метою обстеження кон'юнктивних гілок, що залишилися. Завдяки використанню повернення зникає необхідність розкривати всі дочірні вершини одночасно і запам'ятовувати альтернативні потенційні підграфи розв'язків.

3.5 Алгоритм евристичного пошуку в І-АБО графі

Алгоритми евристичного пошуку передбачають впорядкування списку OPEN відповідно до деякої евристичної оціночної функції. При пошуку в просторі станів евристична функція була оцінкою вартості шляху між поточною і цільовою вершинами. У випадку пошуку в І-АБО графі евристична функція відповідає оцінці вартості підграфа розв'язку.

Позначимо через $h(n)$ вартість підграфа розв'язку, що починається при вершині n. Вартість підграфа розв'язку визначається рекурсивно:

а) якщо n відповідає кінцевій вершині, то $h(n)=0$;

б) для будь-якої іншої вершини

$$h(n)=\sum_{i=1}^{k}\left[h(n_i)+c(n,n_i)\right],$$

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 підграфа розв'язку починається із кінцевих вершин і закінчується початковою вершиною.

В ході пошуку підграфа розв'язку розглядаються потенційні підграфи розв'язків, для яких кінцеві вершини є, як правило, ще нерозкритими вершинами. Для визначення вартості потенційних підграфів розв'язку вводяться оцінки $h(n)$, що позначаються як $\hat{h}(n)$. При цьому $\hat{h}(n)=0$, якщо n – кінцева вершина, і

$$\hat{h}(n)=\sum_{i=1}^{k}\left[\hat{h}(n_i)+c(n,n_i)\right], \qquad (3.1)$$

де n – вершина, що розглядається.

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

Розглянемо приклад. На рис.3.10 зображений потенційний граф розв'язку, у якому пунктирними лініями показані ще не побудовані гілки [1].

Цифри в дужках поряд з вершинами C і D позначають евристичні оцінки підграфів розв'язку, що починаються в цих вершинах. Цифри без дужок (вершини A, S) позначають оцінки, обчислені відповідно до формули (3.1). Вартість дуги вважається рівною одиниці.

При відомих евристичних оцінках підграфів розв'язків, що починаються при вершинах C і D, оцінка підграфа для вершини A дорівнює

$$\hat{h}(A)=\hat{h}(C)+c(A,C)+\hat{h}(D)+c(A,D)=10,$$

а повна оцінка вартості $\hat{h}(S)$ зображеного потенційного графа дорівнює 11.

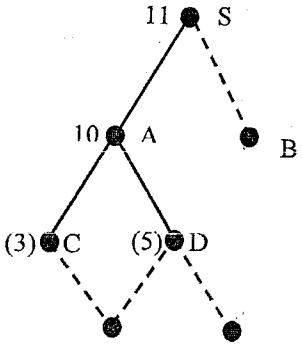


Рисунок 3.10 – Потенційний граф розв'язку

Евристичний алгоритм впорядкованого пошуку можна подати у вигляді процедури AO_Search.

```
Procedure AO_Search;
Begin
    Помістити в список OPEN потенційний граф P, що складається
     тільки з початкової вершини S;
     Виконати розмітку графа P;

     Обчислити оцінку ĥ(S);
      While OPEN <> „пустий список" Do
      Begin
          P:=first(OPEN); {вибрати з OPEN перший граф}
          If solved(P) Then вихід („УСПІХ");
          Вилучити граф P зі списку OPEN;
          Розширити граф P, виконати розмітку всіх знов
сформованих потенційних графів Pi, визначити оцінки ĥ(P);
          Внести потенційні графи P, в список OPEN з урахуванням
        оцінок ĥ(P)
      End
      Вихід („НЕВДАЧА")
End.
```

В даній процедурі використовується певний порядок розкриття вершин. Якщо вершина *n*, що розкривається, є диз'юнктивною, то в новий потенційний підграф розв'язку включається тільки одна її дочірня вершина. Якщо вершина, що розкривається, є кон'юнктивною, то в новий потенційний підграф розв'язку включаються всі її дочірні вершини. Надалі, в першу чергу, розкривається та з дочірніх вершин, яка має

максимальну оцінку $\hat{h}(n_i)$, Підставою для цього є необхідність якнайшвидше з'ясувати, чи є розв'язною 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 графа розв'язку, що починається при вершині n, вилучається з подальшого розгляду. Імовірність виявлення нерозв'язності І-вершини підвищується, якщо рухатися у напрямку максимальної оцінки $\hat{h}(n_i)$.

Розглянемо застосування процедури евристичного пошуку до графа редукції задачі, зображеного на рис. 3.11 [1].

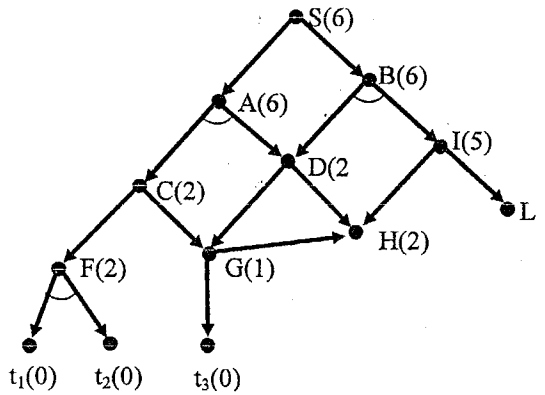


Рисунок 3.11 – Граф редукції задачі з оцінками $\hat{h}(n)$

На рис.3.12 показані потенційні графи розв'язків, що зберігаються в списку OPEN.

На кроці 3 розкривається вершина B, що має оцінку $\hat{h}(B) = 5$. Розкриття вершини B приводить до підзадач D та I з оцінками 2 і 5, відповідно. Внаслідок цього даний потенційний підграф одержує оцінку 10 і пересувається на друге місце списку OPEN. Далі будується лівий підграф вершини S і на кроці 7 потенційний граф помічається як граф розв'язку. Аналогічно пошуку в глибину в дану процедуру можна включити функцію NextChild(n). Це дозволить підвищити ефективність використання комп'ютерної пам'яті при пошуку в графах редукції задач з великою кількістю вершин.

Якщо алгоритм пошуку знаходить оптимальний граф розв'язку (такий граф має мінімально можливу оцінку $\hat{h}(s)$), то він називається гарантуючим. Можна показати, що алгоритм пошуку в І-АБО графі буде гарантуючим, якщо $\hat{h}(n) \le h(n)$ і вартості всіх дуг $c(n,n_i)$ перевищують деяку позитивну величину [2, 8].

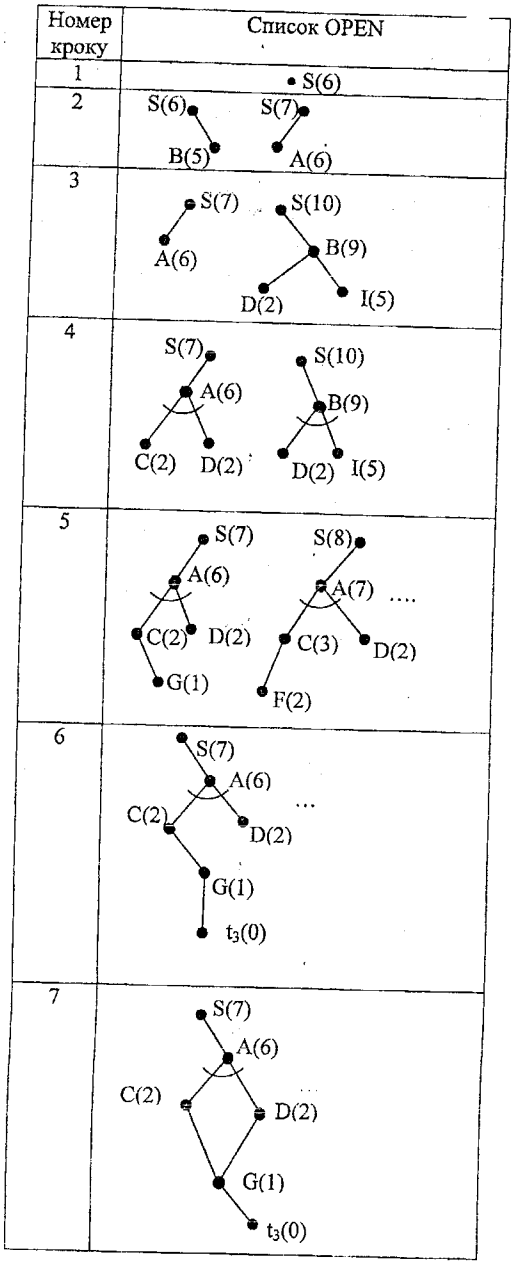


Рисунок 3.12 – Потенційні підграфи розв'язків

Якщо покласти вартість всіх дуг рівною нулю, то мінімізуватимуться тільки майбутні витрати.

3.6 Контрольні питання

1. Проаналізуйте пошук на І-АБО графах. Поняття графа розв'язку. Вартість графа розв'язку.
2. Проаналізуйте евристичну процедуру АО* пошуку на І-АБО графах.
3. Визначте поняття розв'язуваності вершин І-АБО графів.
4. Дайте оцінку використання І-АБО графів для подання задач.
5. Визначте узагальнене поняття І-АБО графів.
6. Побудуйте власний приклад І-АБО графа для довільної задачі.
7. Дайте означення І-АБО графа.
8. Що таке елементарні задачі?
9. Що таке тупикові задачі?
10. Дайте рекурсивне означення розв'язності вершини.
11. Дайте рекурсивне означення нерозв'язності вершини.

ЛАБОРАТОРНА РОБОТА № 3
ПОШУК РОЗВ'ЯЗКІВ У ПРОСТОРІ ЗАДАЧ

Мета роботи

Надбання практичних навичок з розбиття задач на підзадачі, подання простору задач за допомогою І-АБО графів та розробки алгоритмів евристичного пошуку розв'язків при зведенні задачі до підзадач.

У результаті виконання лабораторної роботи студент повинен вміти:

- виявляти задачі, які доцільно розв'язувати шляхом зведення їх до підзадач;
- здійснювати оптимальне розбиття задачі на підзадачі;
- подавати задачі, що дозволяють розбиття на підзадачі, з використанням І-АБО графів;
- створювати оціночні функції для пошуку розв'язків на І-АБО графах;
- проектувати евристичні алгоритми пошуку розв'язків на І-АБО графах;
- здійснювати програмну реалізацію евристичних алгоритмів пошуку на І-АБО графах.

Завдання на підготовку

Студент повинен знати:

- методи подання задач у просторі станів;
- методи побудови оціночних функцій;
- основні алгоритми евристичного пошуку у просторі станів;
- методологію подання задач, що зводяться до підзадач, І-АБО графами;
- основні алгоритми евристичного пошуку розв'язків на І-АБО графах.
- мови програмування LISP, PROLOG, Pascal, C++.

Студент повинен вміти:

- подавати задачі у просторі станів;
- визначати евристики для організації пошуку розв'язків у просторі станів;
- будувати оціночні функції для пошуку розв'язків у просторі станів;
- створювати програми мовами LISP, PROLOG, Pascal, C++.

Для допуску до виконання роботи необхідно:

- вміти відповісти на теоретичні питання за ходом виконання роботи;
- показати викладачу заготівку звіту про лабораторну роботу, яка повинна містити титульний лист та опис обраної предметної області.

Завдання на лабораторну роботу

1. Визначити правила редукції узгодженої з викладачем задачі.

2. Побудувати оціночну функцію для пошуку розв'язку у просторі задач.

3. Написати програму пошуку розв'язку у просторі задач, узгодженою з викладачем мовою програмування.

Порядок виконання лабораторної роботи

1. Запропонувати власну задачу (або вибрати задачу з наведеного нижче переліку) для розв'язку її методом редукції з використанням евристичного алгоритму АО*.
2. Для вибраної задачі:
   - здійснити оптимальне розбиття задачі на підзадачі;
   - подати задачу І-АБО графом;
   - вибрати ознаки для формування евристичних функцій;
   - відповідно до вимог алгоритму АО* запропонувати три оціночних функції ($\tilde{h}_1 \le h;\ \tilde{h}_2 > h,\ \tilde{h}_3 >> h$);
   - спроектувати евристичний алгоритм пошуку розв'язку на І-АБО графі;
   - перевірити ефективність кожної з запропонованих оціночних функцій, застосувавши їх для розв'язування фрагмента задачі;
   - порівняти отримані результати і сформувати відповідні висновки.
3. Графічно відобразити фрагменти графів пошуку, отриманих з використанням кожної із запропонованих оціночних функцій, з відображенням значень оцінок для кожної із розкритих вершин.
4. Узгодити з викладачем мову програмування (LISP, PROLOG, C++, JAVA, Pascal) і написати програму для моделювання роботи запропонованих оціночних функцій, яка повинна забезпечувати можливості:
   - перегляду бази знань (операторів переходів між станами)
   - покрокового виведення на екран кожного з наступних можливих підграфів розв'язу в процесі здійснення пошуку з відображенням значення їх оцінок;
   - автоматичного виконання процедури пошуку з відображенням розв'язку задачі у вигляді графа розв'язку зі значенням його оцінки.

Зміст звіту

Теоретична частина

1. Опис умов задачі.
2. Обгрунтування вибору варіанта розбиття задачі на множину підзадач.
3. Графічне подання редукції задачі І-АБО графом.
4. Якісне обгрунтування вибору правил формування оціночної функції.
5. Побудова співвідношення для обчислення кількісного значення оцінок станів.

6. Фрагменти дерев розв'язку, побудованих з використанням запропонованих оціночних функцій.

Практична частина

1. Схема алгоритму програми пошуку розв'язку на І-АБО графі з використанням оціночної функції.
2. Приклад роботи програми проілюстрований скріншотами.
3. Інструкція користувача для роботи з програмою.
4. Лістінг програми з коментарями.

Приклади завдань на лабораторну роботу

Здійсніть декомпозицію та пошук розв'язку в просторі задач для таких ситуацій.

1. Загін солдат підходить до ріки, через яку необхідно переправитись. Міст зламано, але на човні вздовж берега катаються двоє хлопчаків. В човні може переправитися лише один солдат або два хлопчики і не більше. Необхідно забезпечити переправу всіх солдат через ріку.

2. В квадраті, який має 16 комірок, треба розмістити 16 літер (4 літери *A*, 4 літери *B*, 4 літери *C*, 4 літери *E*) так, щоб на кожній вертикалі і на кожній горизонталі будь-яка літера зустрічалась тільки один раз.

3. Дано два глечики: перший - з водою ($V = 5$л), другий - пустий ($V = 2$л). Необхідно отримати рівно 1л води в глечику ємністю 2л. Воду можна виливати і переливати з одного глечика до іншого.

4. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу, враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів.

Здійсніть декомпозицію ігрової ситуації та пошук її розв'язку в просторі задач.

1. "Гра п'ятнадцать".
2. "Вовк та вівці".
3. "Сім" , [1д].
4. "Клац" , [1д].
5. "Поліцейська машина" , [5д].
6. "Так-тікс" (гра Хейна), [3д].
7. "Гекс", [4д].
8. "Чорний ящик" , [7д].

9. "Рума" , [6д].
10. "Гра Гейла" , [4д].
11. "Французька військова гра" , [7д].
12. "Морська битва".
13. "Пентамімо", [3д].
14. За допомогою алгоритму АО* пошуку на І-АБО графах перетворіть цифру 7 в рядок одиниць, використовуючи для заміни однієї цифри на рядок цифр такі правила: 7 → 6,1; 7 → 4,3; 7 → 5,2; 6 → 4,2; 6 → 5,1; 5 → 3,2; 4 → 3,1; 3 → 2,1; 2 →1,1. Вважайте, що вартість k-зв'язки дорівнює k одиниць, і що функція h в вершинах, які помічені цифрою 1, приймає значення 0, а в вершинах, які помічені цифрами n (n >1) - значення n. Побудуйте початковий І-АБО граф та граф розв'язку. Визначте вартість графа розв'язку.
15. За допомогою алгоритму АО* пошуку на І-АБО графах перетворіть цифру 7 в рядок одиниць, використовуючи для заміни однієї цифри на рядок цифр такі правила: 7 → 6, 1; 7 → 3, 2, 2; 6 → 3, 3; 6 → 4, 2; 6 → 5, 1; 4 → 3, 1; 4 → 2, 2; 3 → 2, 1; 2 → 1, 1. Вважайте, що вартість k-зв'язки дорівнює k одиниць, і що функція h в вершинах, які помічені цифрою 1, приймає значення 0, а в вершинах, які помічені цифрами n (n >1) - значення n. Побудуйте початковий І-АБО граф та граф розв'язку. Визначте вартість графа розв'язку.
16. За допомогою алгоритму АО* пошуку на І-АБО графах перетворіть цифру 7 в рядок одиниць, використовуючи для заміни однієї цифри на рядок цифр такі правила: 7 → 6, 1; 7 → 5, 2;7 → 4, 3; 7 → 3, 2, 2; 6 → 3, 3; 6 → 4, 2; 6 → 5, 1; 5 → 3, 2; 4 → 3, 1; 4 → 2, 2; 3 → 2, 1; 2 → 1, 1. Вважайте, що вартість k-зв'язки дорівнює k одиниць, і що функція h в вершинах, які помічені цифрою 1, приймає значення 0, а в вершинах, які помічені цифрами n (n >1) - значення n. Побудуйте початковий І-АБО граф та граф розв'язку. Визначте вартість графа розв'язку.

4 ПОШУК РОЗВ'ЯЗКІВ НА ІГРОВИХ ДЕРЕВАХ

4.1 Ігри для двох гравців як задача прийняття рішень

Ігрові задачі традиційно належать до інтелектуальних, і були одними з перших задач, які розглядалися в галузі штучного інтелекту. З середини 50-х років минулого сторіччя рівень гри комп'ютерів постійно підвищувався і сьогодні комп'ютери перевершують людей у шашках і „реверсі", мають перемоги над чемпіонами світу у шахах та нардах, стали конкурентноздатними у багатьох інших іграх [8].

Одна з причин привабливості ігор для штучного інтелекту полягає в тому, що в них дуже важно знайти розв'язок. Наприклад, коефіцієнт розгалуження в шахах складає близько 35, а гра продовжується до 50 ходів з боку кожного гравця. Отож дерево пошуку має приблизно 25^{100} або 100^{154} вузлів (хоча граф пошуку налічує близько 10^{40} вузлів). Тому ігри, як і реальне життя, потребують здатності прийняття хоча б якихось рішень, навіть якщо прийняття оптимальних рішень не є можливим. Крім того, ігри сурово карають за неефективність. Якщо одна реалізація алгоритму A* буде працювати вдвічі повільніше за іншу, алгоритм просто буде вимагати вдвічі більшого часу на пошук розв'язку. Шахова ж програма, вдвічі повільніша ніж її конкурентка, скоріше за все отримає поразку вже на початковому етапі гри.

У загальному розумінні іграми називають задачу пошуку в умовах протидії. Тобто, більшість ситуацій прийняття рішень в умовах протидії можна розглядати як один з типів ігор, наприклад, гру з природою, яка робить (на нашу думку) випадкові ходи, або гру з партнером, який робить найкращі (така гра часто називається кооперативною), або найгірші (конкуренція) для нас ходи.

Для теорії штучного інтелекту найбільший інтерес становлять методи знаходження планів гри і оптимальних стратегій для таких ігор як шахи, шашки, "хрестики-нулики", які з погляду теорії є ідентичними між собою.

Вказані ігри належать до класу позиційних ігор двох осіб і характеризуються такими особливостями [1, 12]:

- грають завжди двоє гравців, кожний з яких може по черзі зробити будь-який хід з тих, що дозволяються правилами гри;
- детермінованість - перебіг гри і вибір ходу не залежать від випадкових чинників;
- повна інформація - кожному гравцеві доступна вся інформація про будь-яку позицію, що утворюється в процесі гри (гравець завжди може відрізнити одну позицію від іншої);
- антагоністичність (або гра з нуль сумою), що дозволяє замість двох функцій виграшу розглядати лише одну;

- скінченність - з кожної позиції можна зробити скінченну кількість ходів, і гра не може тривати нескінченно, а повинна через скінченну кількість ходів привести до певного результату.

Можна назвати відомі позиційні ігри, які належать до інших класів. Наприклад, нарди є грою з повною інформацією, але недетермінованою, оскільки гравець не може зробити довільний хід бо його вибір обмежений випадковими чинниками (результатом викидання). Преферанс не є грою з повною інформацією і детермінованою, оскільки його початкова позиція залежить від випадку. Але після того, як початкова позиція зафіксована, гравець може зробити будь-який хід, дозволений правилами.

Базою теорії вважається результат Льюїса і Райфа: для кожного гравця у будь-якій позиції існує оптимальна стратегія, тобто алгоритм визначення чергового ходу, що призводить до певного оптимального результату. "Оптимальність" як термін слід розуміти таким чином: якщо гравці *A* і *B* дотримуються своїх оптимальних стратегій, гра повинна закінчитися певним визначеним результатом. Якщо один з гравців, наприклад *A*, відхилиться від оптимальної стратегії, а гравець *B* продовжуватиме її дотримуватися, то *A* від цього може лише втратити, бо його виграш зменшиться. Зрозуміло, що якщо обидва суперники грають неоптимальним чином, то результат гри стає непередбаченим [8].

4.2 Пошук на ігрових деревах

Ключовим поняттям у теорії позиційних ігор є дерево гри (його також називають деревом аналізу). Воно будується за такими правилами [12].

1. Кожна вершина дерева відповідає окремій позиції.
2. Корінь дерева відповідає позиції, що аналізується. Аналіз може бути проведений для будь-якої позиції, включаючи початкову; у цьому випадку йдеться про повний аналіз усієї гри.
3. Листки дерева відповідають завершальним позиціям. Для завершальних позицій завжди відома функція виграшу, тобто завжди відомий результат гри. Ця функція визначається правилами конкретної гри і може задаватися деякою формулою або конкретними числовими значеннями.
4. Дуги відповідають ходам. Якщо в позиції 1 можливий хід, який переводить позицію 1 у позицію 2, то з позиції 1 до позиції 2 йде орієнтована дуга, що відповідає цьому ходу.

Дерево гри може бути експліцитним та імпліцитним. Експліцитне дерево задається явно; імпліцитне породжується за необхідності. Для цього потрібно задати породжуючу процедуру і критерій завершення, який встановлює, чи була досягнута завершальна позиція.

За стандартною домовленістю при визначенні глибини дерева гри звичайний хід у шашках, шахах і т. п. вважається півходом, а хід складається з двох півходів: ходу гравця та відповіді суперника.

Спочатку розглянемо приклад гри з невеликим простором станів, що забезпечує можливість вичерпного пошуку. У таких іграх проблема вирішується систематичним перебором усіх можливих ходів і відповідей суперника. Розглянемо приклад застосування ігрових графів для такої гри.

Однією з таки ігор є гра„Нім". В цій грі на стіл викладається кілька фішок. Кожним ходом гравець має розбити будь-яку з купок фішок на дві, обов'язково з різною кількістю предметів. Програє той, хто не зможе зробити черговий хід, тому що всі купки будуть складатися з однієї або двох фішок.

Почнемо гру з купки, яка містить сім фішок.

Стан такої гри можна описати невпорядкованою послідовністю чисел, яка відображає кількість предметів в різних купках, а також ознаку того, хто має зробити наступний хід.

Традиційно гравців називають іменами MAX і MIN. Ці імена вибрані з таких причин. MAX є гравцем, на користь якого розв'язується задача, і який намагається виграти, або максимізувати свою перевагу. MIN є супротивником MAX, і намагається мінімізувати його перевагу. Вважається, що MIN використовує ту ж саму інформацію, що й MAX, і завжди намагається досягти найгіршого для MAX стану.

Нехай першим у нашій грі ходить супротивник, тобто MIN. При цьому початкова ситуація гри набуде опису: (7 MIN). MIN може зробити один з трьох ходів, які приведуть до ситуацій: (6,1, MAX), (5,2, MAX), (4,3, MAX). Повний граф буде мати вигляд, наведений на рис. 4.1 [5].

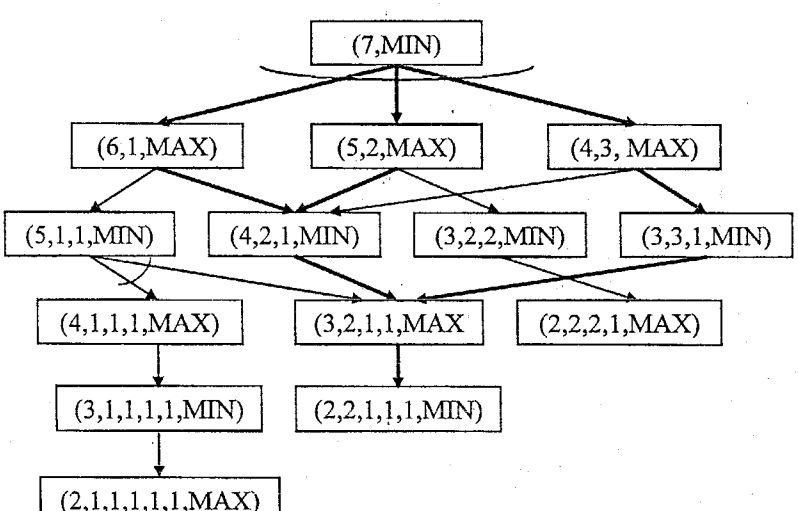


Рисунок 4.1 – Приклад графа пошуку на ігровому дереві

З графа можна побачити, що незалежно від поведінки MIN у MAX завжди існує можливість перемогти (виграшна стратегія позначена товстими стрілками). Щоб довести це, для кожної вершини MIN необхідно показати, що гравець MAX може отримати перемогу при будь-якому ході гравця MIN. Для кожної з вершин, з яких ходить MAX, достатньо показати, що MAX має можливість досягти переможної ситуації виходячи хоча б з однієї з тих позицій, до яких він може перевести гру, зробивши черговий хід.

Відмітимо подібність виграшної стратегії гравця MAX з графом розв'язання в І-АБО графі. Вершини MIN подібні до вершин типу І. З точки зору інтересів MAX розв'язання повинно бути досяжне з кожної з її дочірніх вершин. Вершини, що відповідають гравцеві MAX, подібні до вершин типу АБО, тому що з позиції інтересів гравця MAX перемогу треба здобути хоча б з однієї з дочірніх вершин. Термінальні вершини в даному випадку, це вершини, які відповідають перемозі гравця MAX.

4.3 Мінімаксна процедура перебору на ігрових деревах

У 1945 році Оскар Монгерштерн і Джон фон Непман запропонували процедуру пошуку на ігрових деревах, яка отримала назву мінімаксної. При реалізації стратегії мінімакса кожен рівень дерева в пошуку помічають іменем гравця, якому на цьому рівні належить хід: MAX або MIN. Кожному листовому вузлу надається значення 1 або 0 в залежності від переможця, відповідно, MAX або MIN. У процесі реалізації процедури мінімакса ці значення розповсюджуються уверх по графа згідно з такими правилами:

- якщо батьківський стан є вузлом рівня MAX, то йому приписують максимальне зі значень його дочірніх вузлів;
- якщо батьківський стан є вузлом рівня MIN, то йому приписують мінімальне зі значень його дочірніх вузлів.

Значення, що таким чином пов'язуються з вершинами, відображають найкращий стан, якого гравець може досягнути з цього стану (беручи до уваги дії супротивника відповідно до алгоритму мінімакса). Таким чином, отримані значення використовуються для того, щоб вибрати найкращий з можливих ходів. Результат застосування алгоритму мінімакса до графа простору станів гри „нім" наведено на рис.4.2.

Значення вершин присвоюються знизу до верху на основі принципу мінімаксу. Оскільки будь-який з можливих перших кроків MIN веде до вершин зі значенням 1, другий гравець, MAX, завжди може перемогти, незалежно від першого ходу MIN. MIN може перемогти лише у випадку, коли MAX припуститься помилки.

Але кількість ігор, які допускають повний перебір, є невеликою. Як вже відмічалось раніше, більшість ігор мають дуже великий простір можливих станів, що принципово виключає можливість повного перебору.

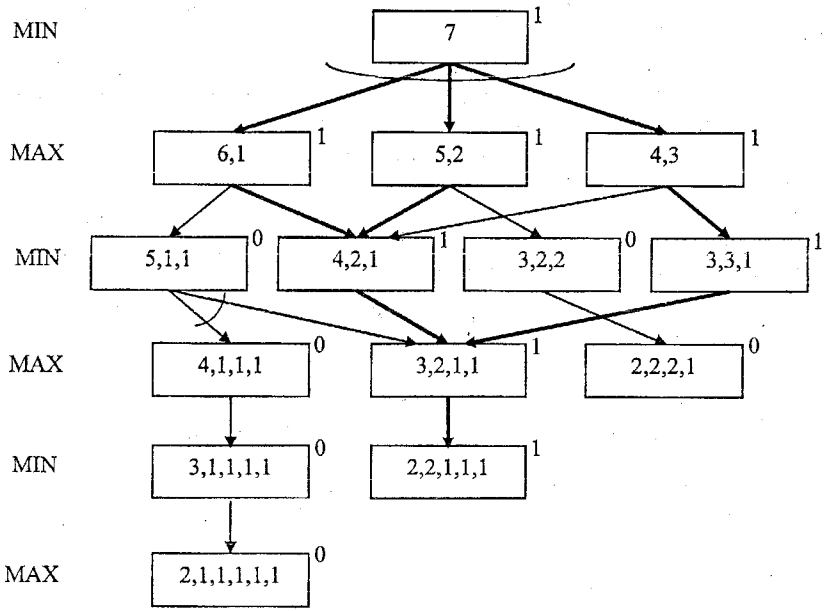


Рисунок 4.2 – Застосування принципу мінімакса для гри „Нім"

Тому для складних ігор доводиться погодитись з відсутністю можливості повного перебору, тобто відмовитись від бажання довести можливість досягнення виграшу або нічиєї (виключаючи можливі ендшпілі). При цьому можна виконати пошук на певну обмежену глибину, яка визначається наявними ресурсами часу і пам'яті комп'ютера. Така стратегія називається прогнозуванням *n*-го шару (n-ply look-ahead), де *n* – кількість досліджуваних рівнів [8]. Такий підграф не відображає усіх станів гри, і тому його вузлам неможливо надати значення, що відображають перемогу або поразку. Тому з кожною вершиною пов'язують значення, що визначається певною евристичною оціночною функцією. Значення, що надаються кожній з *n* вершин на шляху від кореневої вершини – не є при цьому показником можливості досягнення перемоги, а є лише евристичним значенням оптимального стану, якого можна досягти за *n* кроків від цієї кореневої вершини. Ці значення називаються статичними оцінками.

Прогнозування збільшує силу евристики, дозволяючи застосувати її на більшій області простору станів (припускається, що чим меншою є відстань від оцінюваного стану до цільового, тим більш точною буде оцінка стану). Стратегія мінімаксу інтегрує такі окремі оцінки у значення, яке надається батьківському стану.

На рис.4.3. наведено найпростіший приклад застосування мінімаксної процедури [5].

Гравці MAX і MIN ходять по черзі. Статичні оцінки вершин найнижчого рівня MAX визначаються за допомогою деякої функції оцінювання. Динамічні оцінки всіх внутрішніх вершин визначають, просуваючись знизу до верху, від рівня до рівня, доки не досягнуть кореневої вершини. Остаточна оцінка дорівнює 4, і найкращими для гравців ходами в кожній позиції є ті, що помічені товстими лініями. Така послідовність ходів називається основним варіантом і визначає "мінімаксно - оптимальну" гру суперників.

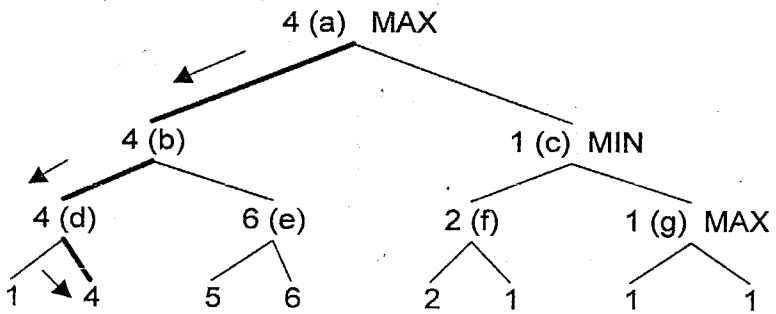


Рисунок 4.3 – Приклад застосування мінімаксної процедури

4.4 Альфа-бета пошук

Одна з найбільш складних проблем, що виникають при створенні ігрових програм, полягає в уникненні перегляду безперспективних шляхів і пов'язаних з цим витрат ресурсів. Наприклад, в мінімаксній процедурі процес породження дерева пошуку повністю відокремлений від процесу оцінки позиції, яка виконується після завершення породження дерева. Таке розділення є причиною дуже низької ефективності стратегії пошуку. Набагато кращі результати можна отримати, якщо вести пошук у глибину і при породженні кінцевої вершини одразу обчислювати її статичну оцінку. Повернене значення також повинно приписуватись будь-якій позиції одразу, як тільки з'явиться можливість його обчислення. Підвищення ефективності пошуку можна досягти за рахунок використання такої ідеї. Якщо існує два варіанти ходу, то зрозумівши, що один хід є кращим за інший, можна прийняти правильне рішення не уточнюючи, наскільки саме один з ходів є кращим за інший.

Розглянемо використання цього принципу на наведеному раніше прикладі (рис. 4.3). Розпочинаючи пошук з позиції (*a*) (рис. 4.4) переходимо послідовно в позиції (*b*) та (*d*). Тут обчислюємо значення статичних оцінок кінцевих вершин-спадкоємців вершини (*d*) і обираємо

більшу з них, яка дорівнює 4. Далі повертаємося в (*b*) і переходимо до (*e*), де розпізнаємо спочатку першого спадкоємця позиції *e*, який має оцінку 5. При цьому MAX (який ходить в позицію *e*) виявляє, що йому гарантована в цій позиції оцінка не менша за 5, незалежно від оцінок інших (можливо і більш кращих) варіантів ходу. Цього виявляється достатньо для того, щоб MIN, навіть не знаючи точної оцінки позиції *e*, зрозумів, що для нього з позиції *b* хід в позицію *d* є кращим, ніж хід в позицію *e*, оцінка якої буде не меншою за 5.

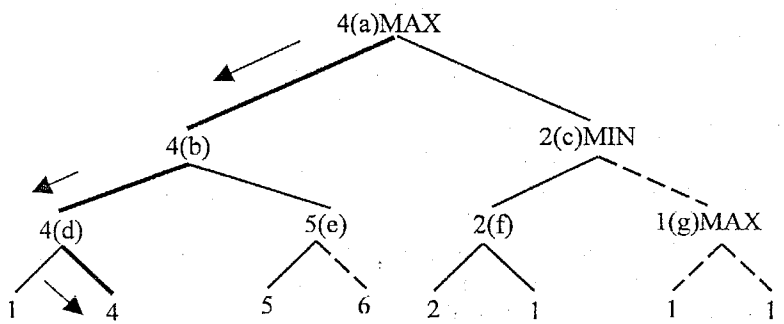


Рисунок 4.4 – Приклад реалізації альфа-бета процедури (пунктирними лініями позначені відсічені гілки дерева пошуку)

Розмірковуючи таким чином, можна знехтувати другим спадкоємцем позиції *e* і приписати їй наближену оцінку 5. Не дивлячись на наближений характер цієї оцінки, прийняте припущення ніяк не впливає на оцінку позиції *b*, а тим самим і на оцінку позиції *a*. На цій ідеї базується широко відомий альфа-бета алгоритм, що є однією з найбільш ефективних реалізацій мінімаксного принципу. Ми бачимо з прикладу, що застосування цього алгоритму призвело до того, що значення деяких оцінок (*e* і *c*) стало не точним. Але таких наближених значень виявилось достатньо для точної оцінки кореневої вершини. При цьому складність пошуку зменшилася до п'яти звернень до функції оцінювання замість восьми (в першому варіанті дерева пошуку).

Припинення аналізу деякої вершини разом з усіма її нащадками називають відсіканням.

Формалізація алгоритму полягає у визначені двох граничних умов, що позначаються як альфа й бета, між якими має знаходитися оцінка кореневої вершини. Через альфу позначають найменше значення оцінки, гарантованої на даний момент часу гравцю MAX (або найкраща з точки зору MIN оцінка, на яку він ще може сподіватись). При цьому бета відповідає найбільшому значенню оцінки, на яку може сподіватись MAX

(або найгірша, але гарантована на даний момент часу оцінка з точки зору гравця MIN). Дійсно, шукане значення завжди розташоване в інтервалі між значеннями альфа і бета. Таким чином, якщо деяка оцінка лежить за межами цього інтервалу, то це означає, що вона не входить в основний варіант і знання точного значення потрібне лише для тих оцінок, які входять до інтервалу альфа-бета. Помітимо також, що значення оцінки альфа в процесі пошуку не можуть зменшуватись, а значення оцінки бета не можуть зростати. Завдяки цим обмеженням можна сформулювати такі умови припинення пошуку (або виявлення "безперспективних" шляхів) [3].

1. Можна не проводити пошук на піддереві, що зростає з будь-якої MIN вершини, бета-значення якої (тобто гарантоване MIN значення), менше або дорівнює альфа-значенню будь-якої з її батьківських MAX вершин (тобто їх гарантованому MAX значенню). Такій вершині можна приписати остаточне повернене значення, яке дорівнює її бета-значенню. Хоча таке значення може і не збігатися з тим, яке можна отримати при здійсненні повного пошуку, але воно достатнє для вибору одного і того ж найкращого в даній ситуації ходу.

2. Можна не проводити пошук на піддереві, що зростає з будь-якої MAX вершини, альфа-значення якої є більшим за бета-значення будь-якої з її батьківських MIN вершин. Такій вершині можна приписати остаточне повернене значення, яке дорівнює її альфа-значенню. В процесі пошуку альфа і бета значення обчислюють за такими правилами:

а) для MAX вершини значення альфа приймається рівним найбільшому на даний час значенню серед остаточно повернених значень для її дочірніх вершин;

б) для MIN вершини значення бета приймається рівним найменшому на даний час значенню серед остаточно повернених значень для її дочірніх вершин.

Припинення пошуку згідно з першим правилом називається альфа-відсіканням, а згідно з другим – бета-відсіканням (рис. 4.4).

Ще раз зупинимося на головному принципі процедури відсікання. Припустимо, що вершина S відповідає позиції, в якій хід належить гравцю MAX. При цьому він може зробити кілька ходів, два з яких показано на рис. 4.5 [4].

Внаслідок одного з них виникає позиція A, а в наслідок іншого Y. Припустимо також, що позиція A вже повністю вивчена і знайдено значення її оцінки (α). Розпізнаємо позицію Y. Припустимо, що один хід з неї приводить до позиції Z, оцінка якої, згідно з методом мінімаксу, дорівнює z. Далі припустимо, що виконується умова $z \leq \alpha$.

Якщо s і y є оцінками вершин S і Y, то в будь-якому випадку виконується умова $y \leq z$, оскільки в позиції Y хід належить MIN, який завжди обирає такий хід, що веде до позиції з мінімальною оцінкою. Звідси випливає нерівність $y \leq z \leq \alpha$, а оскільки зрозуміло, що s більше або

дорівнює α і y, то маємо $y \le z \le \alpha \le s$. Тоді невідоме значення оцінки біля вершини Y не буде впливати на наступні результати, і аналіз всіх гілок, що виходять з вершини Y, за винятком вершини Z, буде марним. Тобто, після аналізу вершини Z всі інші гілки дерева, що виходять з вершини Y, можна відкинути. Аналогічно, у випадку, коли перший хід належить гравцю MIN, оцінка z на глибині 2 повинна задовольняти умову $z \ge \beta$ (бета-відсікання), де через β позначимо в даному випадку оцінку вершини A.

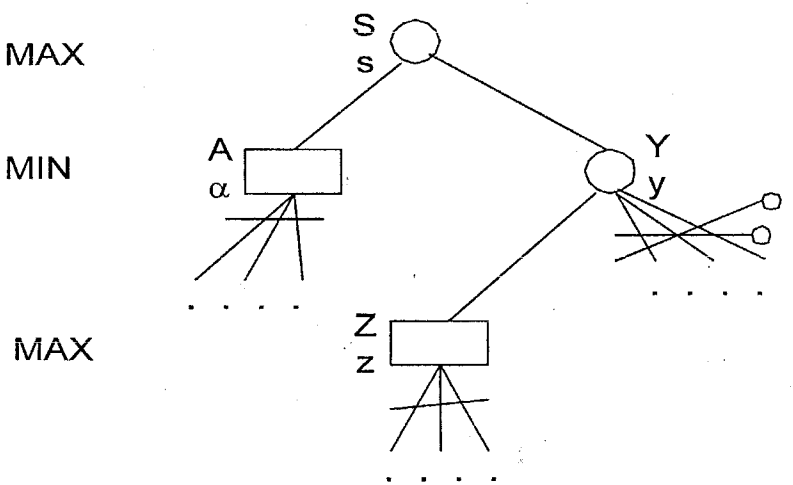


Рисунок 4.5 – Приклад застосування відсічень

Сказане виконується і для вершин, що розташовані на суттєво різних глибинах дерева, при умові, що вони належать рівням одного і того ж самого гравця (рис.4.6).

Нехай хід з позиції S може привести до позицій A або U. Оцінку A вже здійснено і вона дорівнює α. Починаючи з позиції U помітимо, що після непарної кількості проміжних ходів, яка в даному випадку дорівнює 3, виникає позиція Z, оцінка якої дорівнює z. Знов припустимо, що z менша за α. Для позиції Y знайдемо оцінку y, яка буде задовольняти умову $y \le z$, оскільки хід в Y належить гравцю MIN. Покажемо, що і в цьому випадку, результат аналізу інших позицій, які йдуть за Y, не може вплинути на кінцевий результат оцінки s. Розглянемо позицію V. Її аналіз може привести до двох різних результатів: $v > y$, або $v = y$. В першому випадку, результати аналізу інших ходів не мають впливу на оцінку v, оскільки вже відомо, що $y \le z$, а значення y в наслідок аналізу може лише зменшуватись. В другому випадку з $y = v$ маємо: $v = y \le z \le \alpha$. Тобто вершина V отримує оцінку меншу за A, що приведе до схеми, яка раніше

розглядалася. Таким чином, в обох випадках оцінка s позиції S не залежить від інших гілок дерева, які виходять з вузла Y і їх можна не аналізувати (глибинне альфа-відсікання). Аналогічно можна розмірковувати і для позицій, з яких перший хід робить MIN. В цьому випадку β-відсікання можна робити кожен раз, коли оцінка позиції, що виникає після ходу MIN, перевищує значення β.

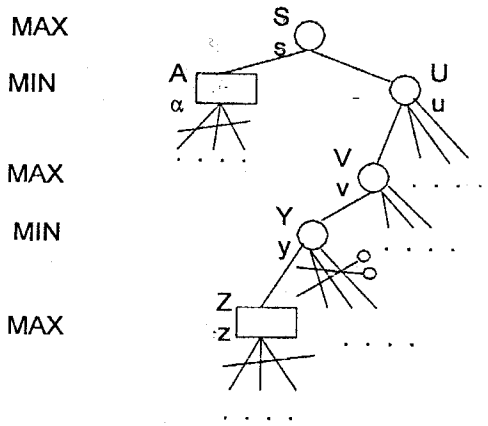


Рисунок 4.6 – Глибинне альфа-бета відсікання

Зауважимо, що процедура відсікання може застосовуватись тільки з моменту отримання оцінки не менше ніж для двох вершин.

4.5 Ефективність пошуку за допомогою альфа-бета процедури

Для проведення альфа-бета відсікання хоча б частина дерева повинна бути побудована на мінімальну глибину, оскільки обчислення значень альфа і бета основані на статичних значеннях кінцевих вершин. Крім того, кількість відсічень, що виникають в процесі пошуку, залежить від того, наскільки початкові значення альфа та бета були наближені до остаточного поверненого значення. Таке значення початкової вершини збігається зі статичною оцінкою однієї з кінцевих вершин. Якщо при пошуку в глибину таку вершину буде досягнуто першою, то кількість відсічень буде максимальною, тобто буде породжено і оцінено мінімальну кількість вершин. Нехай глибина дерева дорівнює D і кожна некінцева вершина має рівно B спадкоємців. В такому дереві буде B^D вершин. Якщо альфа-бета процедура буде породжувати дочірні вершини в такій послідовності, що для MIN вершин завжди будуть першими породжуватись вершини з мінімальними поверненими значеннями, а для вершин MAX - з максимальними поверненими значеннями, то кількість відсічень буде

максимальною, тобто буде породжено і оцінено мінімальну кількість вершин.

Нехай глибина дерева дорівнює D і кожна вершина (за винятком кінцевих) має рівно B дочірніх вершин. На рівні D в такому дереві буде B^D вершин. Якщо альфа-бета процедура буде породжувати дочірні вершини в такій послідовності, що для MIN вершин завжди будуть першими породжуватись вершини з мінімальними повернеными значеннями, то кількість відсічень буде максимальною. При цьому, при кількості кінцевих вершин ND, буде виконуватися таке співвідношення [3, 4, 12]:

$$ND = \begin{cases} 2 \cdot B^{\frac{D}{2}} - 1, \text{для парних D} \\ B^{\frac{D+1}{2}} + B^{\frac{D-1}{2}} - 1, \text{для непарних D} \end{cases}$$

Це співвідношення означає, що при оптимальному альфа-бета пошуку на рівні D породжується приблизно така ж кількість кінцевих вершин, яка породжується на рівні $D/2$ без його застосування. Тобто альфа-бета процедура з ідеальною послідовністю породження дочірніх вершин дозволяє подвоїти глибину пошуку при одних і тих же обсягах пам'яті.

Таким чином, альфа-бета відсікання не усувають експоненційного зростання, але дозволяють зменшити його порядок. Якщо врахувати, що в шахах вже при аналізі на 3-5 ходів доводиться переглядати 10^{10} —10^{14} позицій виявляється, що максимальні альфа-бета відсікання можуть скоротити час на вибір ходу в сотні тисяч або мільйони разів. При заданих же часових обмеженнях можна приблизно вдвічі збільшити глибину перебору.

На практиці максимальні альфа-бета відсікання ніколи не зустрічаються. Гравець, як правило, не може заздалегідь знати, які ходи є оптимальними — якби це знати, ні альфа-бета відсікання, ні сама мінімаксна процедура не були б потрібні. Але гравець може на основі тих чи інших евристичних міркувань надати наступним позиціям робочі оцінки, що відображають їх міру перспективності, упорядкувати їх за цими оцінками і відповідно до цього розглядати передусім ходи, які видаються найперспективнішими.

Існує багато підходів до такого впорядкування. Найпростішим з них є такий: в першу чергу розглядати наступні стани з найкращою статичною оцінкою. Очевидною є також доцільність іншого підходу: насамперед розглядати ходи, які істотно змінюють статичні оцінки (наприклад, взяття фігур у шахах) або різко скорочують кількість можливих відповідей (наприклад, шах королю).

Широко застосовуються і процедури неглибокого пошуку. Вони полягають у тому, що спочатку проводиться повний аналіз на невелику глибину q, і робочі оцінки розставляються відповідно до результатів цього

аналізу. Витрати на такий попередній пошук, як правило, компенсуються зростанням ефективності подальших відсічень.

Неважко пересвідчитися, що впорядкування за статичними оцінками фактично є варіантом неглибокого пошуку при $q = 1$.

Упорядкування позицій за робочими оцінками може бути здійснено один раз і далі не змінюватись. У такому разі йдеться про фіксоване (інша назва - жорстке) упорядкування. Якщо ж при додаванні нової інформації, що з'являється в процесі аналізу, позиції можуть перевпорядковуватися, то йдеться про динамічне упорядкування. Однією з давніх і найвідоміших процедур альфа-бета відсікання з використанням такого перевпорядкування є альфа-бета процедура. Очевидно, що динамічне упорядкування є гнучкішим і часто дозволяє збільшити кількість відсікань порівняно з фіксованим варіантом. Розглянемо відомий приклад на цю тему [3].

Розглянемо дерево гри, зображене на рис. 4.7. Позиції другого рівня є завершальними, їх оцінки, які виступають функціями виграшу, наведені всередині відповідних кружечків. Статичні оцінки позицій першого рівня зображені поряд з ними.

При цьому дерево є імпліцитним, тобто породжується в процесі аналізу. Відповідно до цього і оцінки завершальних позицій не можуть бути відомі одразу. Спочатку процедура здійснює упорядкування наступників першого рівня за їх статичними оцінками; породження цих вершин здійснюється на основі стратегії "спочатку в ширину". Одразу слід відмітити, що відповідно до цього упорядкування процедура оцінює хід P_1 як найперспективніший у той час, коли насправді він є найгіршим.

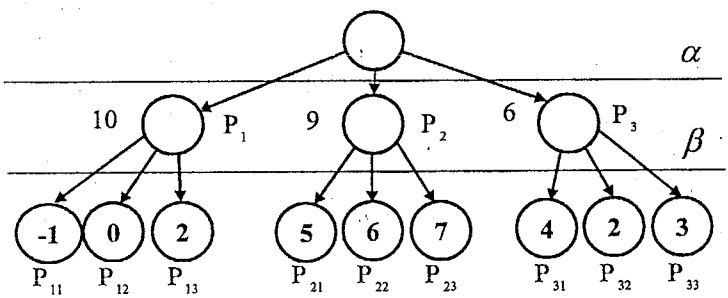


Рисунок 4.7 - Ілюстрація фіксованого і динамічного упорядкування

При фіксованому упорядкуванні, тобто якщо порядок аналізу вершин-нащадків надалі не змінюється, процедура спочатку аналізує всіх нащадків вершини P_1. Це дає остаточну оцінку цієї вершини "-1", і відповідно змінюється попередня оцінка кореня.

Аналіз наступної позиції (P_2) змінює попередню оцінку кореня на 5. І лише після розкриття вершини P_{31}, яка має оцінку 4, вдається здійснити відсікання, яке в даному випадку завершує аналіз.

При використанні динамічного перевпорядкування ситуація змінюється. Аналіз вершини P_1, надає їй попередню оцінку -1, яка значно менша за робочу. Саме це змушує процедуру здійснити перевпорядкування, на основі якого перейти до вершини P_2.

Після аналізу нащадка P_{21} робоча оцінка P_2 зменшується до 5. Але, на відміну від попереднього випадку, це зменшення є незначним, і тому дана процедура приймає рішення не проводити перевпорядкування, а продовжувати аналіз P_2. І це рішення в даному разі виявляється цілком правильним. У результаті аналізу вершина P_2 набуває остаточної оцінки 5, а вершина P - такої ж попередньої оцінки. Тепер можна одразу ж здійснити відсікання у вершині P_1, а після аналізу P_{31} - й у P_3. Отже, проаналізовано значно менше вершин, ніж при фіксованому упорядкуванні.

Можна сформулювати таке загальне правило [12]. Нехай поточна робоча оцінка деякої вершини дорівнює K. Для альфа-вершин перевпорядкування здійснюється у разі, якщо ця вершина набуває попередньої оцінки, меншої за $K - R$, а для бета-вершин - попередньої оцінки, більшої за $K + R$. Інакше кажучи, перевпорядкування здійснюється, якщо оцінка позиції погіршується на величину, що перевищує R.

Важливий параметр R, який називається опірністю, визначає, якою мірою процедура робить опір перевпорядкуванню. При досить великому значенні R процедура ніколи не здійснює перевпорядкування і діє подібно до жорсткого упорядкування. Якщо ж R досить мале, упорядкування буде надлишковим. Тому належний вибір R, який залежить від специфіки конкретної гри, має дуже велике значення. Очевидно також, що задачу підбору R можна, зокрема, переформулювати в межах теорії нечітких множин.

Критеріями закінчення пошуку, які мають особливе значення при використанні імпліцитних дерев, що породжується за необхідності, служать: досягнення завершальної позиції; досягнення максимальної глибини; досягнення "мертвої" позиції, тобто такої, статична оцінка якої мало відрізняється від робочої оцінки, отриманої внаслідок виконання деякого попереднього аналізу та інші.

На завершення зауважимо, що ігрові програми, основані на аналізі дерева ходів, мають два основних недоліки.

1. Повна відсутність стратегії. Програма не веде гру, а просто аналізує деяку послідовність позицій. Програма опирається лише на гігантську продуктивність комп'ютера, а не на здібність до формального мислення з логічним аналізом позицій та ефективним пошуком рішень. Так в шаховій позиції, де чорні мають три пішаки та короля (Крg8, f7, g7, h7), а білі два пішаки та короля (КреЗ, а2, Ь2), людина одразу визначить

перемогу білих. Але машина оцінить таку позицію як програшну, тому що для кінцевої оцінки потрібно проаналізувати її на велику кількість півходів вперед.

І навпаки, у шахах цілком ймовірною є ситуація, за якої з „оптимального" стану програма на наступному ході отримує мат. Тобто, відсутність стратегії приводить до того, що машина не може "побачити", а тим більше оцінити наслідки ходу, які проявляться на глибині, яка перевищує встановлену хоча б на один хід. Тому саме глибина перебору є одним з вирішальних чинників, що визначає силу ігрових програм та якість їх гри.

2. Ефект горизонту. Внаслідок першого недоліку було встановлено, що програми, побудовані на принципах систематичного перебору дозволених ходів, згідно зі своєю структурою намагаються зробити все для того, щоб віддалити всі несприятливі події за межу максимальної глибини аналізу, тобто за межу горизонту. Вони не бажають бачити наближення катастрофи і мають один засіб боротьби з нею - відсувати невдалий результат за допомогою абсурдних ходів. Програма буде віддавати велику кількість менш цінних фігур, лише для того, щоб віддалити за межу горизонту втрату ферзя, хоча така втрата є неминучою. Цей недолік породжено основною ідеєю алгоритму перебору і не може бути усунений. З його наслідками і пов'язані основні відмінності гри ігрових програм від гри людини.

Узагальненням класичної мінімаксної процедури є *m^n*-процедура [9]. Згідно з цією процедурою, на відміну від класичного мінімаксу, оцінка альфа-вершини визначається не одним найкращим наступником, а є функцією від *m* найкращих дочірніх вершин. Аналогічно оцінка бета-вершини є функцією від *n* найкращих дочірніх вершин. Ця ідея ґрунтується на міркуванні, що часто буває краще мати декілька гарних наступників, ніж лише одного.

Отже, звичайна мінімаксна процедура утворюється з *m^n*-процедури при $m = n = 1$. За певних умов *m^n*-процедура може показувати кращі результати, ніж звичайна мінімаксна, про що свідчать окремі експерименти, зокрема з процедурою (2, 2) [12].

Вибір оптимальних значень параметрів *m* та *n* суттєво залежить від специфіки задачі. Очевидно, якщо гра є порівняно простою і припускає повний аналіз, то класична мінімаксна процедура є оптимальною, і збільшення *m* та *n* тільки погіршить якість гри. Але *m^n*-процедуру при *m* або *n*, що не дорівнюють одиниці, можна порекомендувати за наявності різних суб'єктивних та об'єктивних невизначеностей, наприклад: неможливість отримання надійних оцінок окремих варіантів; залежність результатів ходу від випадкових чинників; вибір неоптимальних ходів суперниками і т. ін.

Наприклад, у шахах цілком ймовірною є ситуація, за якої з

„оптимального" стану програма на наступному ході отримує мат. Тобто, відсутність стратегії приводить до того, що машина не може "побачити", а тим більше оцінити наслідки ходу, які проявляться на глибині, яка перевищує встановлену хоча б на один хід. Тому саме глибина перебору є одним з вирішальних чинників, що визначає силу ігрових програм та якість їх гри.

На завершення відмітимо основну принципову відмінність пошуку на дереві гри від пошуку на графі розв'язку задачі. В іграх відсутнє повернення назад, це означає, що алгоритми пошуку в просторі станів тут не використовуються.

4.6 Контрольні питання

1. Які характерні особливості ігрових задач відрізняють їх від інших задач прийняття рішень?
2. Що таке оптимальна стратегія? Для якого класу ігор можна довести існування оптимальних стратегій?
3. Що означає термін „гра з повною інформацією"?
4. Наведіть означення дерева гри та наведіть приклад дерева гри для конкретної ігрової ситуації.
5. Які різновиди ігрових дерев Ви знаєте?
6. У чому полягає сутність мінімаксної процедури?
7. Які параметри впливають на силу ігрових програм? Як змінюється мінімаксна процедура при обмежені глибини пошуку?
8. Що таке *m^n*-процедура і як вона співвідноситься з класичною мінімаксною процедурою?
9. Назвіть переваги використання альф-бета процедури.
10. Поясніть, за рахунок чого альфа-бета процедура має переваги перед мінмаксною процедурою.
11. Поясніть алгоритм пошуку в глибину при зведенні задачі до підзадач.
12. Чи залежить ефективність альфа-бета процедури від порядку розгляду ходів? Підтвердіть свою відповідь конкретним прикладом.
13. Що таке фіксоване і динамічне упорядкування?
14. Проаналізуйте глибинне альфа-відсікання і бета-відсікання.
15. Проаналізуйте задачу пошуку на ігрових деревах.
16. Визначіть ефективність пошуку з використанням альфа-бета процедури.
17. Проаналізуйте методи підвищення ефективності альфа-бета процедури.
18. Поясніть аналогію ігрових дерев з І-АБО графами.
19. Поясніть, в якому випадку альфа-бета відсікання є максимальним.
20. Назвіть основні критерії зупинки пошуку на ігровому дереві.

ЛАБОРАТОРНА РОБОТА №4
ПОШУК РОЗВ'ЯЗКІВ НА ІГРОВИХ ДЕРЕВАХ

Мета роботи

Надбання практичних навичок з використання мінімаксних процедур пошуку розв'язків на ігрових деревах на основі базової альфа-бета процедури і розробки відповідного програмного забезпечення

Завдання на підготовку

Студент повинен знати:

- основні стратегії пошуку в системах штучного інтелекту;
- неінформовані процедури пошуку на графах;
- мінімаксні процедури пошуку на графах;
- методи підвищення ефективності альфа-бета процедури.

Студент повинен вміти:

- будувати евристичні оціночні функції;
- обчислювати значення статичних та динамічних мінімаксних оцінок;
- виявляти відсікання на основі значень параметрів альфа і бета;
- застосовувати альфа-бета процедуру для пошуку на ігрових деревах;
- створювати програми мовами LISP, PROLOG, JAVA, C++, PASCAL.

Для допуску до виконання роботи необхідно:

- вміти відповісти на теоретичні питання за ходом виконання роботи;
- показати викладачу заготівку звіту про лабораторну роботу, яка повинна містити титульний лист та опис задачі з обраної предметної області.

Завдання на лабораторну роботу

1. Побудувати дерево альфа-бета пошуку для узгодженої з викладачем інтелектуальної задачі.
2. Здійснити програмну реалізацію альфа-бета процедури, узгодженою з викладачем мовою програмування

Порядок виконання лабораторної роботи

1. Для вибраної задачі:

- визначити найбільш інформативні ознаки, які будуть використовуватися у поданні станів ігрової ситуації;
- подати гру у просторі станів;
- побудувати евристичну оціночну функцію станів гри;

- подати у вибраній формі початковий стан задачі;
- визначити перелік операторів переходів між станами;
- побудувати та графічно відобразити дерево альфа-бета перебору для фрагмента вибраної гри.
- проаналізувати ефективність запропонованої альфа-бета процедури;
- здійснити заходи із забезпечення підвищення ефективності альфа-бета процедури;

2. Здійснити програмну реалізацію альфа-бета процедури пошуку розв'язку задачі, яка повинна забезпечувати можливості:
- формування дерева пошуку розв'язку ігрової ситуації на задану глибину;
- покрокового перегляду дерева альфа-бета пошуку з відображенням значень статичних та динамічних оцінок та відсічень;
- автоматичного здійснення пошуку розв'язку за допомогою альфа-бета процедури.

3. Проаналізувати ефективність отриманої альфа-бета процедури та надати рекомендації щодо її вдосконалення.

4. Програмно реалізувати механізм підвищення ефективності альфа-бета процедури.

Зміст звіту

Теоретична частина
1. Опис ігрової ситуації.
2. Обґрунтування вибору значущих ознак станів гри.
3. Подання гри у просторі станів.
4. Обґрунтування вибору евристичної оціночної функції.
5. Фрагмент дерева пошуку розв'язку гри
6. Фрагмент альфа-бета дерева пошуку з відображенням статичних й динамічних оцінок та відсічень.
9. Аналіз ефективності отриманої альфа-бета процедури.
10. Пропозиції щодо підвищення ефективності альфа-бета процедури.

Практична частина
1. Схема алгоритму програмної реалізації альфа-бета процедури.
2. Приклад роботи програми, проілюстрований скріншотами.
3. Опис змін програми для підвищення ефективності альфа-бета процедури.
4. Результати роботи модифікованої програми.
5. Інструкція користувача для роботи з програмою.
6. Лістінг програми з коментарями.

Приклади завдань на лабораторну роботу

Здійсніть пошук розв'язків наведених ситуацій з використанням альфа-бета алгоритму.

1. Загін солдат підходить до ріки, через яку необхідно переправитись. Міст зламано, але на човні вздовж берега катаються двоє хлопчаків. В човні може переправитися лише один солдат або два хлопчики і не більше. Необхідно забезпечити переправу всіх солдат через ріку.

2. В квадраті, який має 16 комірок, треба розмістити 16 літер (4 літери *A*, 4 літери *B*, 4 літери *C*, 4 літери *E*) так, щоб на кожній вертикалі і на кожній горизонталі будь-яка літера зустрічалась тільки один раз.

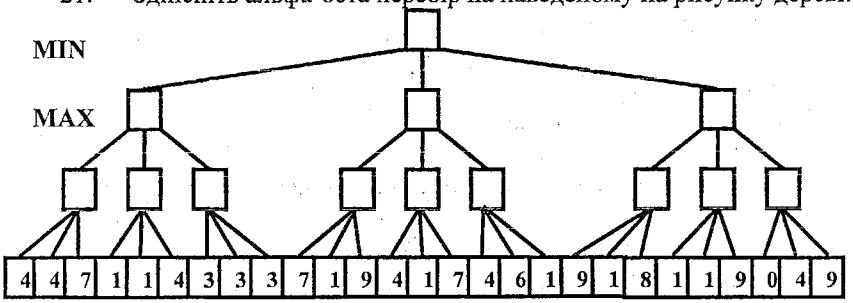
3. Дано два глечики: перший - з водою ($V = 5$л), другий - пустий ($V = 2$л). Необхідно отримати рівно 1л води в глечику ємністю 2л. Воду можна виливати і переливати з одного глечика до іншого.

4. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів. Три місіонери і три туземці знаходяться на лівому березі ріки. Їм треба переправитись на правий берег, однак вони мають лише один човен і в човні можуть знаходитись лише дві особи. Треба забезпечити переправу враховуючи, що якщо на будь-якому березі ріки кількість місіонерів стане меншою за кількість туземців, то останні з'їдять місіонерів.

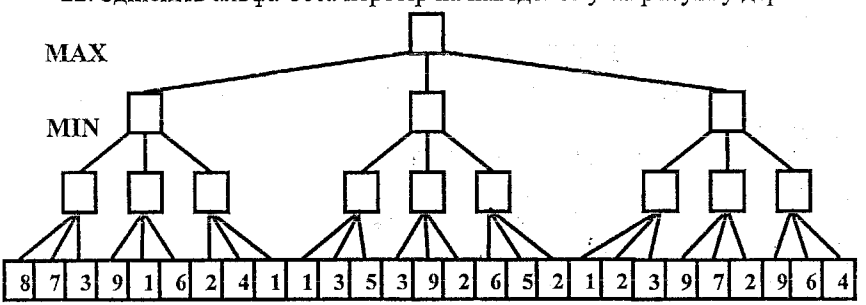
Здійсніть пошук розв'язків ігрової ситуації з використанням альфа-бета алгоритму.

1. Гра „п'ятнадцять".
2. "Квадрат, який обертається".
3. "Вовк та вівці".
4. "Сім", [1д].
5. "Клац", [1д].
6. "Гонки", [1д].
7. "Розсада", [2д].
8. "Поліцейська машина", [4д].
9. "Так-тікс" (гра Хейна), [3д].
10. "Гекс", [4д].
11. "Двокольорові шашки", [3д].
12. "Чорний ящик", [2д].
13. "Рума", [3д].
14. "Гра Гсйла" , [4д].
15. "Французька військова гра", [2д].
16. "Тривимірні хрестики-нулики", [2д].
17. "Морська битва".

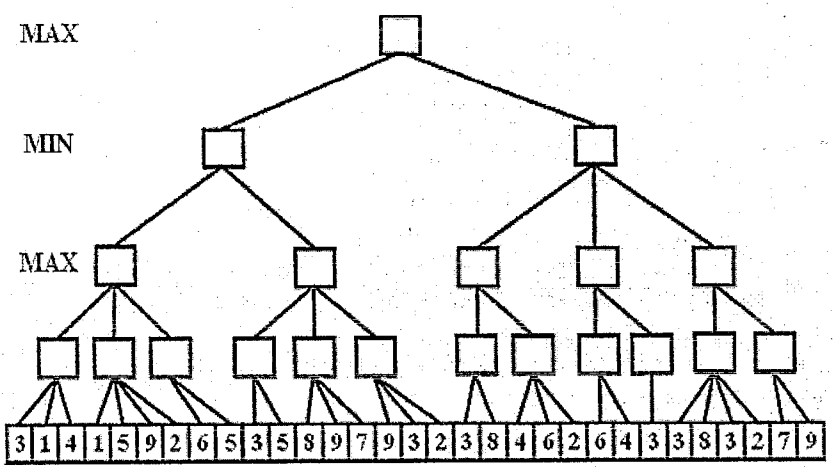
18. "Пентамімо", [3д].
19. "Кутки".
20. "Хід конем".
21. Здійсніть альфа-бета перебір на наведеному на рисунку дереві:



22. Здійсніть альфа-бета перебір на наведеному на рисунку дереві:



23. Здійсніть альфа-бета перебір на наведеному на рисунку дереві:



ГЛОСАРІЙ

Штучний інтелект – artificial intelligence
Простір станів – state space
Граф простору станів – graph state space
Можливі стани – possible states
Початковий стан –initial state
Поточний стан – current status
Продукційна система – production system
Формальна система продукцій – formal production system
Алгоритм управління системою продукцій – production system control algorithm
Метод сліпого пошуку – blind search method
Методи впорядкованого (евристичного) пошуку – ordered (heuristic) search methods
Список OPEN– OPEN list
Список CLOSED – CLOSED list
Випадковий пошук – random search
Пошук „в глибину"– depth-first search
Пошук „в ширину"– breadth- first search
Алгоритм рівних цін – equal prices algorithm
Алгоритм „підйому на гору" – hill-climbing algorithm
Алгоритм А*– А* algorithm
Оціночна функція – value function
Пошук з розповсюдженням обмежень – constraint distribution search
Дерево пошуку – search tree
Зведення задачі до підзадач – problem redaction
Вершини типу І – AND type vertices
Вершини типу АБО – OR type vertices
І-АБО граф – AND-OR graph
Граф редукції задачі – problem reduction graph
Позиційні ігри двох осіб – position games for two persons
Пошук на ігрових деревах – search in playing trees
Мінімаксна процедура перебору – minimax procedure of surplus
Батьківська вершина – parents node
Дочірня вершина – child node
Альфа-бета процедура – alpha-beta pruning
Альфа-відсікання – alpha cutoff
Бета-відсікання – beta cutoff
Пошук у просторі станів – search in state space
Пошук у просторі задач – search in problem space

ПЕРЕЛІК ВИКОРИСТАНОЇ ЛІТЕРАТУРИ

Основна література

1. Бондарев В. Н., Аде Ф. Г. Искусственный интеллект: Учебн. пособие для вузов. – Севастополь: Изд-во СевНТУ, 2002.

2. Каллан. Р. Основные концепции нейронных сетей.: Пер.с англ. – М.: Издательский дом «Вильямс», 2001.

3. Нильсон Н. Проблемы искусственного интеллекта.: Пер. с англ. – М.: Радио и связь, 1985.

4. Лорьер Ж-Л. Системы искусственного интеллекта: Пер.с франц.: - М.: Мир, 1991.

5. Месюра В. І., Ваховська Л. М. Основи проектування систем штучного інтелекту. – Вінниця: ВДТУ, 2000.

6. Девятков В. В. Системы искусственного интеллекта: Учеб. пособие для вузов. – М.: Изд-во МГТУ им. Н. Э. Баумана, 2001.

7. Братко И. Алгоритмы искусственного интеллекта на языке PROLOG, 3-е издание.: Пер. с англ. – М.: Издательский дом «Вильямс», 2004.

8. Люгер Джордж Ф. Искусственный интеллект: стратегии и методы решения сложных проблем, 4-е изд. Пер. с англ. – М.: Издательский дом „Вильямс", 2003.

9. Shirai Y. Artificial intelligence/ Y Shirai – Salisbary: Jojn Wiley & Sons Ltd, 1984.

10. Рассел С., Норвиг П. Искусственный интеллект, современный подход, 2-е изд.: Пер. с англ. – М.: Издательский дом «Вильямс», 2006.

11. Luger G.F. Artificial intelligence: Structures and Strategies for complex problem solving/ G.F. Luger, W.A.Snubblefield. – 2-nd. ed. – Redwood: The Benjamin Publishing Company Inc., 1993.

12. Глибовець М. М., Отецький О. В. Штучний інтелект: Підручн. для студ. вищ. навч. закладів. – К.: Вид. дім „КМ Академія", 2002.

13. Джонс М.Т. Программирование искусственного интеллекта в приложениях/ М.Тим Джонс; Пер. с англ. Осипов А. И. – М.: ДМК Пресс. 2004.

Додаткова література

1. Арсак Ж. Программирование игр и головоломок. Пер. с франц. – М.: Наука. Гл. ред. физ.-мат. лит., 1990.

2. Гарднер М. Математические головоломки и развлечения: Пер. с англ. - М.: Мир, 1971.

3. Гарднер М. Математические досуги: Пер. с англ. - М.: Мир, 1972.

4. Гарднер М. Математические новеллы: Пер. с англ. - М.: Мир, 1974.

Навчальне видання

Володимир Іванович Месюра,
Любов Михайлівна Ваховська

Основи проектування систем штучного інтелекту

Лабораторний практикум

Частина 1. Способи представлення задач і пошуку розв'язків

Лабораторний практикум

Оригінал-макет підготовлено Ваховською Л. М.

Редактор В. О. Дружиніна
Коректор З. В. Поліщук

Науково-методичний відділ ВНТУ
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ

Підписано до друку 11.06.2009 р.
Формат 29,7х42 ¼
Друк різографічний
Тир. 85 прим.
Зам. № 2009-118

Гарнітура Times New Roman
Папір офсетний
Ум. друк. арк. 6.7

Віддруковано в комп'ютерному інформаційно-видавничому центрі
Вінницького національного технічного університету
Свідоцтво Держкомінформу України
серія ДК № 746 від 25.12.2001
21021, м. Вінниця, Хмельницьке шосе, 95, ВНТУ