Міністерство освіти і науки України
Вінницький державний технічний університет

В.І. Месюра, Л. М. Ваховська

Основи проектування систем штучного інтелекту

Вінниця ВДТУ 2000

Міністерство освіти і науки України
Вінницький державний технічний університет

В.І.Месюра, Л. М. Ваховська

Основи проектування систем штучного інтелекту

Затверджено Ученою радою Вінницького державного технічного університету як навчальний посібник для студентів спеціальності "Інтелектуальні системи прийняття рішень". Протокол № 5 від 30 грудня 1999 р.

Вінниця ВДТУ 2000

УДК 007

Основи проектування систем штучного інтелекту. Навчальний посібник / В. І. Месюра, Л.М. Ваховська. - В.: ВДТУ, 2000, - 96 с. Укр. мовою /

Посібник містить теоретичний матеріал та практичні завдання згідно з навчальною програмою з дисципліни "Основи проектування систем штучного інтелекту".

Бібліогр. 11 найм. Іл. 38.

Рецензенти : В.П. Кожем'яко, д.т.н.
Г.А. Рудомин, к.т.н.
С.М. Білан, к.т.н.

Зміст

Вступ

Для сучасного етапу розвитку науково-технічного прогресу характерним є широке поширення задач, які важко або просто неможливо формалізувати. Алгоритмічний розв'язок таких задач або не існує взагалі, або не може бути отримане на сучасних комп'ютерах, при наявних ресурсах. Ефективним засобом розв'язання цих задач є інформаційна технологія штучного інтелекту, яка заснована на здобутті, накопиченні та обробці знань.

Даний навчальний посібник призначений для вивчення студентами бакалаврату 6.0804 "Комп'ютерні науки" спеціальності 7.080404 "Інтелектуальні системи прийняття рішень" дисципліни "Основи проектування систем штучного інтелекту". Згідно навчального плану вона є першою з циклу дисциплін, що присвячені проблемам створення інтелектуальних систем, і передує таким дисциплінам як "Функціональне та логічне програмування", "Експертні системи", "Системи прийняття рішень", "Програмне забезпечення інтелектуальних систем" та ряду інших.

В зв'язку з цим, в межах даної дисципліни розглядаються такі базові питання штучного інтелекту, як вибір оптимальної форми представлення задачі, розробка стратегій пошуку, евристичні процедури пошуку рішень з використанням І/АБО- графів, альфа-бета процедура і т. ін. В посібнику здійснюється знайомство з основними поняттями формальних систем, велику увагу приділено продукційним системам, як найбільш поширеній моделі представлення знань.

Навчальний посібник, що пропонується, містить матеріали, які використовуються укладачами в лекціях, на практичних та лабораторних заняттях та при курсовому проектуванні. Теоретичні питання супроводжуються практичними прикладами, контрольними завданнями та питаннями для самоперевірки студентами рівня засвоєння матеріалу. Засвоєння матеріалу даного посібника надає добру базу для успішного вивчення інших дисциплін з напрямку штучного інтелекту.

Розділ 1 Введення до проблем штучного інтелекту

1.1 Поняття штучного інтелекту

Загальноприйнятим терміном "штучний інтелект" називають один з наймолодших наукових напрямків, який належить до розділів сучасної інформатики і виник у середині 50-х років. Цікаво, що більш ніж за сорок років існування напрямку так і не було прийняте чітке визначення самого терміну "штучний інтелект". Так сталося тому, що в ньому використовується таке загадкове поняття як "інтелект", яке є предметом протиріч між філософами, математиками, психологами та звичайними людьми. Одним з результатів цих протиріч стала значна кількість визначень поняття інтелекту людини. Так, у дуже популярному на заході словникові Вебстера (видання 1956 р.) це поняття визначається таким чином:

- здатність успішно реагувати на будь-яку, особливо нову, ситуацію шляхом відповідного корегування поведінки;
- здатність розуміти взаємозв'язки між фактами дійсності для вироблення дій, що ведуть до досягнення поставленої мети.

Видатний англійський математик та спеціаліст з обчислювальної техніки Ален Тьюринг вважав, що комп'ютер можна буде визнати розумним, якщо при спілкуванні з ним він примусить Вас повірити в те, що Ви маєте справу не з машиною, а з людиною.

Багато вчених розуміють під інтелектом "здатність мозку людини розв'язувати так звані "інтелектуальні задачі" шляхом придбання, накопичення, запам'ятовування та цілеспрямованого перетворення знань в процесі навчання на досвіді та адаптації до різних обставин. При цьому до інтелектуальних задач відносять такі випадки обробки інформації, які не можуть бути виконані за допомогою точних алгоритмічних методів. Тобто, в умовах певних обмежень необхідно зуміти знайти одне з рішень, яке задовольняє умови задачі, причому не обов'язково оптимальне. А коло таких задач і можливих варіантів їх розв'язання в реальному житті дуже велике.

Припустимо, необхідно створити інтелектуальну систему для забезпечення виконання робіт "розумними" пристроями у середовищах, де присутність людини неможлива або створює небезпеку для її життя. Такі пристрої повинні працювати в умовах великої різноманітності можливих ситуацій. Ясна річ, що такі ситуації (як і будь-які інші ситуації реального життя) неможливо описати з тим ступенем точності та однозначності, які б дозволили закласти в систему жорстко запрограмовані алгоритми поведінки. Тому системи штучного інтелекту повинні мати механізми адаптації, які б дозволяли їм будувати програми цілеспрямованої діяльності

з розв'язання необхідних задач на основі аналізу конкретної ситуації, що склалась на даний момент в навколишньому середовищі.

Таке представлення проблеми потребує вирішення питань, які не виникали раніше а ні в теорії керування, а ні в теорії проектування технічних систем. До головних задач при цьому можна віднести такі:

- опис різноманітного зовнішнього середовища та його відображення в середині системи (задача представлення знань);
- керування великим банком знань: його поповнення, усунення непотрібної інформації, виявлення суперечностей та нестачі знань;
- сприйняття зовнішнього середовища за допомогою різного роду рецепторів (зорових, тактильних, слухових);
- сприйняття друкованого тексту і усної мови і перетворення інформації, яку вони вміщують, до форми представлення знань;
- розуміння людини, зокрема розуміння природної мови, яка служить людині універсальним засобом комунікації;
- побудова логіки зовнішнього світу, яка б дозволила знаходити закономірності життєвого середовища і будувати висновки на основі інформації, яка знаходиться в даний час в моделі знань системи;
- планування діяльності, тобто формування планів досягнення мети за допомогою засобів, які система має в своєму розпорядженні;
- адаптація та навчання на базі накопиченого в системі досвіду.

В найбільш загальному випадку можна сказати, що область штучного інтелекту присвячена проблемі вирішення задач перебору в умовах обмеження ресурсів: часу, пам'яті, доступної інформації і т. д.

Слід відмітити, що головні завдання штучного інтелекту полягають в тому, щоб зробити обчислювальні машини ще більш потужними та корисними, і щоб зрозуміти принципи, на яких базується інтелект. Відповідно і дослідники штучного інтелекту, що працюють над створенням "розумних машин", розподіляються на дві великі групи:

- чисті теоретики, для яких комп'ютер є лише інструментом для експериментальної перевірки теорій процесів мислення;
- прагматики, метою яких є розповсюдження сфери застосування комп'ютерів та полегшення користування ними. Багато з них мало цікавляться механізмами мислення стверджуючи, що для їх роботи це не більш корисне заняття, ніж вивчення польоту птахів для побудови літаків.

Нині обидві ці точки зору довели своє право на життя і зробили вагомий внесок в розвиток наукового напрямку "штучного інтелекту". Але загальною метою і тих і інших все ж таки залишається виявлення принципових механізмів, які покладені в основу діяльності людини з метою їх застосування при розв'язанні конкретних науково-технічних задач. А необхідність використання комп'ютерів при з'ясуванні центральних питань і можливостей інтелекту на доповнення до традиційних методів, які

використовують психологи, філософи, лінгвісти та інші вчені обумовлюється такими причинами:

- комп'ютери є ідеальним засобом для проведення експериментів. Будь-які знання можна просто додати або усунути з комп'ютера для перевірки їх важливості;
- комп'ютери потребують використання точних моделей. Це дозволяє виявляти ті помилки та "підводне каміння", які часто залишаються непоміченими навіть найбільш прискіпливими експериментаторами;
- комп'ютери дозволяють чітко відокремити вимоги, які пов'язані з задачею. Тобто, якщо комп'ютер розв'язує задачу, то можна сформулювати висновок про те, який саме обсяг опрацьованої інформації при цьому потрібний (в усякому випадку його верхня межа).

1.2 Завдання та основні напрямки розвитку штучного інтелекту

Найбільш важливими результатами, яких очікують сьогодні від дослідників штучного інтелекту і без яких важко представити подальший розвиток науково-технічного прогресу є такі:

- Максимальне наближення комп'ютера до користувача. Мається на увазі, що функції програміста у перспективі повинні бути передані комп'ютеру, а складність спілкування із комп'ютером не повинна перевищувати складність спілкування з іншими побутовими системами. Для цього в комп'ютер треба закласти велику кількість знань про механізми розв'язання задач, спеціальні процедури автоматичного синтезу програм, засоби "природного" спілкування з користувачем та таке інше.
- Втілення нових безпаперових технологій, які дозволять людині швидко орієнтуватись у дійсно безмежному океані інформації.
- Придбання нових знань, за рахунок великих можливостей комп'ютерів по обробці інформації, накопиченої людством у вигляді баз даних, знань, та автоматичного виявлення нових, невідомих раніше закономірностей.
- Роботизація промислового виробництва та сільського господарства.
- Автоматизація проектування складних виробів, що дозволить у прийнятний термін розв'язувати такі задачі, на розв'язання яких нині не вистачає життя кількох наукових поколінь.

Основними напрямками досліджень в галузі штучного інтелекту, які направлені на вирішення зазначених задач є такі:

- Представлення знань, пов'язане із вивченням їх джерел та створенням прийомів та процедур отримання, формалізації та представлення знань в інтелектуальних системах (ІС). Ця проблема є однією з найважливіших,

тому що функціонування ІС базується на знаннях про предметне середовище, які зберігаються у її пам'яті.

- Маніпулювання знаннями, яке забезпечує узагальнення знань і формування на їх основі абстрактних понять, одержання методів достовірного і правдоподібного виведення на основі доступних знань, створення моделей міркувань.
- Спілкування, що вміщує проблеми розуміння текстів та природної мови.
- Сприйняття, яке представляє проблему розпізнавання зорової (трьох мірної) інформації.
- Навчання, яке надає можливість навчання ІС розв'язанню задач, з якими вона раніше не зустрічалася.
- Поведінка, що визначає адекватну взаємодію між ІС та навколишнім середовищем за рахунок використання спеціальних методів багаторівневого планування та корегування планів у динамічних ситуаціях.

1.3 Загальне поняття представлення задачі

Як і у випадку з терміном "штучний інтелект", загальноприйнятого визначення терміну "задача" не існує. Причому йдеться не тільки про жорстке словесне формулювання цього терміну, але і про відсутність необхідних знань про структуру задачі, про її основні компоненти, про вигляд її конкретних представлень і таке інше.

Сам термін "розв'язання задачі" використовується у штучному інтелекті (ШІ) в обмеженому сенсі. Про існування задачі йдеться у тому випадку, коли стан навколишнього світу, що сприймається, відрізняється від бажаного. При цьому задача має бути розв'язана таким чином, щоб зникла існуюча відмінність. Вважається, що існує можливість виконання таких дій, які призведуть до необхідних змін. Однак здійснення таких дій "всліпу" не вважається розв'язанням задачі. Розв'язання задачі обов'язково повинно супроводжуватися деяким процесом вибору відповідних дій.

Усі існуючі задачі можна поділити на два основні типи:

- Повністю визначені - це ті задачі, які ми розв'язуємо на заняттях з математики. Такі задачі мають чіткий опис на відповідній кожному випадку мові.
- Неповністю визначені - задачі, які частіше за все зустрічаються у реальному житті. Вони мають часткове описання і формулюються з використанням різних засобів передачі інформації: голос, інтонації, жести, малюнки і таке інше. Головним засобом передачі інформації при цьому служить природна мова, яка з точки зору розв'язання задач має чотири істотних недоліки: неповноту, надлишковість, неоднозначність та неточність.

Наприклад, у діалозі значну кількість інформації не висловлюють точно і ясно, оскільки припускається, що співрозмовники однаково добре знають тему розмови. В іншому випадку можлива повністю помилкова інтерпретація розмови. У технічних галузях цей недолік виключають за рахунок використання технічних вимог, в яких використовують суворі специфікації. Надлишковість часто використовують для того, щоб підкреслити важливість деяких моментів задачі. Однак, при цьому нічого не говориться про те, що труднощі розв'язання задачі пов'язані безпосередньо з цими моментами.

Поставити задачу - означає зрозуміти її умови, тобто усунути неповноту, надлишковість та неоднозначність, або, іншими словами, знайти для неї відповідне представлення. По справжньому мудрі люди не займаються безпосереднім розв'язанням задач у тому вигляді, в якому вони подаються, але займаються пошуком вдалого представлення задачі. Цей факт настільки зрозумілий, що його важливість не відчувається до того часу, доки ми не розпочинаємо створювати програму для розв'язання задачі. Ясна річ, що така програма працює не з реальним фізичним світом, а з його символічним представленням.

При вдалому виборі форми представлення задачі простір пошуку рішень стає добре визначеним і, як правило, забезпечується виявлення головної складності задачі. Задача стає водночас і більш абстрактною і більш жорсткою. При цьому кажуть про представлення задачі у замкненій формі, або про замкнене формулювання задачі.

Під час сприйняття здійснюється інтерпретація задачі в пам'яті людини і така інтерпретація може бути різною. Так, прикладом неоднозначної інтерпретації, яку здійснює зорова система може служити рисунок куба: передня грань якого може знаходитися зліва і знизу, або справа і зверху. В задачі, в якій треба провести ламану лінію, що складається не більш ніж з 4-х лінійних фрагментів через кожну з 9 точок масиву 3 х 3, необхідно подолати внутрішнє обмеження, яке насправді не задане, відносно того, що пряма може виходити за межі квадрату. Така ж ситуація виникає з задачею побудови 4-х рівновеликих трикутників за допомогою 6-ти сірників (обмеження на перетинання сірників та розташування їх в одній площині) [5].

У найбільш загальному вигляді умови задачі записуються математично таким чином:

знайти в заданій множині X точки x, які задовольняють множину заданих обмежень K(x).

При цьому, разом із простором X в загальному випадку неявно задається і структура X, і дозволені операції над X. Знання X є визначальним в початкових даних.

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 представлення задачі.

Процес розв'язання задачі і є саме такою послідовністю зміни представлень, останнє з яких і дає безпосередній розв'язок задачі.

Існують два основних представлення задачі у замкненій формі:

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 в варіанті, що розглядається, має такий вигляд:

X=(послідовність всіх можливих операторів) .

При цьому розв'язком задачі буде певна послідовність операторів:

x=*(Oab, Obc, Ocd, ... , Ofu)*,

де $S_a=S_1$ та $S_u=S_0$.

Множина обмежень $K(x)$ представляє дозволене правило проходження операторів, згідно якому 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 цьому випадку не задані оператори переходу від одного стану до іншого. Мистецтво математика і полягає в тому, щоб знайти ті оператори, які будуть корисні для заданих умов задачі. Можливі і такі випадки, коли кінцевий стан $C(x)$ не задано взагалі. При цьому задача може мати такий вигляд: "Розрахувати $\sum_{i=1}^{n} i^3$", в наслідок 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.

Питання для самоперевірки

1. Визначте поняття штучного інтелекту.
2. Які задачі виникають при створенні систем штучного інтелекту ?
3. Які переваги надає застосування комп'ютерів в дослідженнях з штучного інтелекту?
4. На вирішення яких глобальних задач направлені сьогодні зусилля розробників систем штучного інтелекту?
5. Назвіть основні напрямки досліджень в галузі штучного інтелекту.
6. Який зміст і які особливості має термін "задача" з точки зору штучного інтелекту?
7. Як з математичної точки зору можна в найбільш загальному вигляді сформулювати умови задачі?
8. Наведіть основні типи представлення задач (два основні та додатковий).

Розділ 2 Машинне представлення у вирішенні задач

2.1 Основні форми представлення задач

Серед основних форм представлення задач можна виділити такі:

1. В представленні, яке дозволяє перерахування, розглядається множина U можливих рішень та деяке правило їх оцінки. Якщо U нескінченна, але злічена, то елементи представлення можна породжувати в деякому порядку і досліджувати їх правильність. При сліпому породженні розв'язків цей метод зветься методом проб та помилок. Однак, на практиці розроблено ряд дуже вдалих методів на основі перерахування, які засновані на такому узагальненому алгоритмі:

1. Покласти $k=0$ і розпочати з деякої множини S_0 пробних розв'язків $S_0 \subseteq U$.
2. Якщо S_k вміщує потрібний розв'язок, закінчити роботу. В іншому випадку перейти до п. 3.
3. Побудувати $S_{k+1} = f(S_k)$, збільшити k на1 і перейти до п.2.

Метод на основі перерахування відрізняється від інших тим, що на кожному крокові алгоритму здійснюється вибір множини S_{k+1}, який залежить тільки від S_k і не залежить від результату порівняння останнього з відповіддю. Для того, щоб уникнути розв'язання задачі всліпу, елементи множини S_k порівнюють із цільовим станом, причому таке порівняння закладають у визначення самої функції f.

2. Метод розв'язання задач заснований на поняттях станів та операторів є підходом з точки зору простору станів. Можна помітити, що задача знаходження послідовності операторів подібне до задачі знаходження шляху в графі.

Ми вже розглянули поняття станів та операторів. Простір станів, наприклад, для гри в 15, зручно представляти у вигляді графа, вершини якого відповідають цим станам і пов'язані між собою дугами, які відповідають операторам, що переводять систему з одного стану до іншого. Розв'язок для гри 15 можна було б отримати використовуючи процес пошуку (перебору), при якому оператори застосовуються перш за все до початкового стану з метою одержання нових станів, до яких знов застосовуються оператори і так далі, доки не буде досягнуто кінцевого стану.

3. В певному сенсі більш творчим підходом до вирішення задачі є метод, пов'язаний із підзадачами. При цьому здійснюється аналіз початкової задачі з метою виділення з неї такої множини підзадач, розв'язок певної підмножини якої вміщував би в собі і розв'язок всієї

задачі. Такий процес може застосовуватися рекурсивно для породження підзадач, підпідзадач і так далі, доки не буде одержана множина таких "простих" задач, рішення яких відоме. Прикладом може служити задача вибору шляху для проїзду, припустимо, від Вінниці до Москви.

Метод зведення задачі до підзадач, зветься методом заснованим на редукції задачі. Метод використання простору станів є, жорстко кажучи, виродженим випадком цього методу, тому що кожне застосування оператора перетворює початкову задачу у більш просту задачу.

Як і у методі простору станів, в методі редукції задачі важливу роль відіграє метод проб та помилок, тому що на кожному з етапів може виникнути деяка множина підзадач в яку можна перетворити початкову задачу. Оскільки не кожна з цих множин зможе привести до вирішення задачі, виникає необхідність пошуку в просторі множини підзадач.

4. Використовування формальної логіки. Логічний аналіз ситуації дозволяє інколи значно полегшити пошук розв'язку задачі. Інколи він може показати, що деякі проблеми є нерозв'язними. Наприклад, для гри 15 цільова конфігурація не може бути одержана з початкової конфігурації (рисунок 2.1):

15	14	13	12
11	10	9	8
7	6	5	4
3	2	1	*

Рисунок 2.1 – Початкова конфігурація для гри 15, для якої не існує розв'язку

Необхідність в логічних висновках може виникнути при використанні будь-якого з раніше розглянутих методів.

Далі при вивченні курсу ми побачимо, що багато нематематичних задач можуть бути сформульовані як теореми, які можуть бути доведені в автоматичному режимі. Використання формальної логіки та методів автоматичного доведення теорем дозволяє думати про створення "універсального" розв'язувача задач. Нова інформація могла б вводитися до нього у вигляді додаткових аксіом, що не потребують зміни програми.

Питання про вибір форми представлення задачі є дуже важливим тому, що він завжди є попереднім відносно самого етапу пошуку роз'вязку. У розв'язуванні задачі людиною завжди вражає не швидкість і впорядкованість проведення нею пошуку в просторі всіх можливих розв'язань, а вміння знайти таку ясну точку зору на задачу, що розглядається, яка робить розв'язання задачі елегантно простим. Справа полягає в тому, що для однієї задачі може існувати декілька варіантів представлень, і деякі з них можуть давати незрівнянно більш вузькі

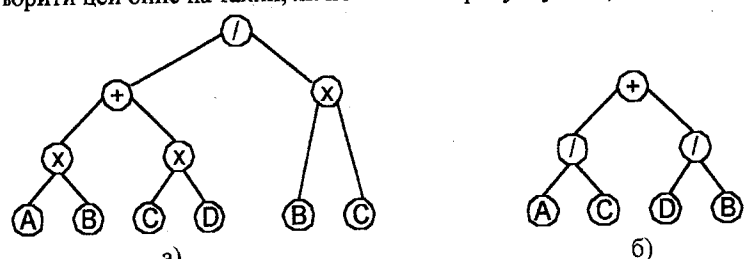
простори станів ніж інші. Тому дуже важливо представити задачу найбільш економним чином з усіх можливих.

2.2 Представлення задач у просторі станів

Важливим етапом побудови опису задачі з використанням простору станів (ПС) є вибір деякої конкретної форми опису станів цієї задачі. Наприклад, для розв'язання гри 15 без реального переміщення фішок треба мати деякий опис конфігурацій, що виникають у грі. Для цього можливо використовувати самі різні структури даних. Це можуть бути і символи, і вектори, і масиви, і дерева, і списки і таке інше. При виборі форми опису треба враховувати простоту перетворення одного стану в інший, яке виконується при застосуванні відповідного оператора.

Розглянемо приклад вибору форми опису станів для задачі перетворення алгебраїчного виразу *(AB+CD)/BC* в більш простий вираз *A/C+D/B*. Зрозуміло, що в якості станів задачі тут повинні виступати алгебраїчні формули.

Дуже поширеною формою опису таких задач є двійкові дерева, некінцеві вершини яких відображають арифметичні знаки (+,-,x,/), а кінцеві - змінні, чи константні символи виразу. Для нашого виразу таке дерево буде мати вигляд, показаний на рисунку 2.2а. Застосування законів алгебраїчних перетворень (операторів у просторі станів) повинно перетворити цей опис на такий, як показано на рисунку 2.2б).



а) б)

Рисунок 2.2 - Двійкові дерева для задачі перетворення алгебраїчного виразу:

а) вираз *(AB+CD)/BC* ;

б) вираз *A/C+D/B*

Іншою відомою формою опису є лінійний рядок з префіксними операторами: */+×AB×CD×BC*. Оператори називаються префіксними тому, що вони стоять в рядку перед операндами. Тут відсутня необхідність в пунктуації, оскільки відомо, що кожен оператор має відношення рівно до двох операндів. Використовуючи рядкові [illegible]

можна сформулювати як задачу перетворення рядка */+×AB×CD×BC* у рядок *+/AC/DB*.

Оскільки оператори переводять систему з одного стану до іншого, вони можуть розглядатись як функції, які визначені на множині станів і приймають значення з цієї ж множини. Такі операторні функції можна визначити у вигляді таблиць, які пов'язують з кожним "вхідним" описом деякий "вихідний" опис. Однак у більшості випадків така таблиця буде практично неприйнятною і тому у більшості випадків ми будемо вважати що оператори є обчисленнями, які перетворюють один стан у інший.

Для опису станів у вигляді рядків найбільш зручним є спосіб представлення операторних обчислень, який базується на правилах переписування, які широко відомі під назвою продукцій. Множина правил переписування визначає можливі способи перетворення одного рядка в інший у вигляді $S_i \Rightarrow S_j$. Прикладом правила переписування може бути таке: $A\$ \Rightarrow B\$$, це означає, що при появі символу A в першій позиції деякого рядка, його можна замінити на символ B. Знак $ означає довільний підрядок, серед них і 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 видалений).

Наприклад, відповідно до двох останніх правил, рядок ABCBABC можна перетворити в рядок ABCAB або ABABC.

До одного і того ж рядка правило переписування може застосовуватись декількома різними способами. Так, правило 2 до рядка ABCBABC взагалі не може бути застосовано, в той час як правило 1 може бути застосовано до нього різними способами. Правило переписування є одним з основних правил доведення в формальних системах, з якими ми ознайомимося пізніше.

Правила переписування, що визначені над масивами, можна застосувати для опису гри 8, що є спрощеним варіантом гри у 15. Наприклад, одне з правил може бути представлено у вигляд, показаному на рисунку 2.3.

x1	x2	x3
x4	*	x5
x6	x7	x8

$\Leftrightarrow$

x1	x2	x3
*	x4	x5
x6	x7	x8

Рисунок 2.3 – Приклад застосування правила переписування для гри 8

В дійсності тут представлені два правила перетворення ситуацій, що відображається двонаправленою стрілкою. В кожному з правил переписування дозволенний хід визначається підстановкою чисел 1, 2,...8 на місця змінних x1, x2, ..., x8 по кожну сторону від стрілки за умови, що $x_i \neq x_j$.

До будь-якої процедури дослідження простору станів входить побудова нових описів станів із старих і перевірка, чи задовольняє новий стан поставлену мету. Крім того, як правило потрібно знайти не будь-який шлях, що веде до мети, а шлях, оптимізований за деяким критерієм (наприклад, шлях, що потребує мінімальної кількості застосування операторів).

Таким чином, для повного представлення задачі в просторі станів необхідно задати:

а) форму опису станів і, зокрема, опис початкових станів;

б) множину операторів і їх впливів на описи станів;

в) властивості опису цільового стану.

2.3 Графові представлення у вирішенні 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 представлені простору станів вершини графа описують стани, а дуги - оператори. Послідовність з k вершин, в якій кожна наступна вершина є дочірньою для попередньої, зветься 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 представлення 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 Г, застосування якого до будь-якої вершини дозволяє отримати всі її дочірні вершини і вартості відповідних дуг. Послідовне застосування

множини $\{s_i\}$ і їх дочірніх елементів дає представлення для графа, що заданий неявно через Г 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 представлено також за допомогою недетермінованих алгоритмів. Наприклад, процес породження ПС можна представити блок-схемою, яка приведена на рисунку 2.4.

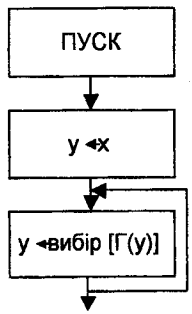


Рисунок 2.4 – Блок схема породження ПС

Тут ***x*** та ***y*** - довільні сукупності даних. У вихідному положенні змінна ***y*** вважається рівною вхідній структурі даних ***x***, що описує початковий стан. Оператор присвоювання ВИБІР (встановити ***y*** таким, що дорівнює деякому члену множини Г(y) елементів, які безпосередньо йдуть за y), є недетермінованим в тому розумінні, що при його виконанні може бути обрано будь-який член множини Г(y). При цьому множина всіх можливих способів виконання програми (можливо нескінчена), 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 приймають 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. Наприклад, задача для гри 8 у недетермінованій формі буде мати такий вигляд (рисунок 2.5):

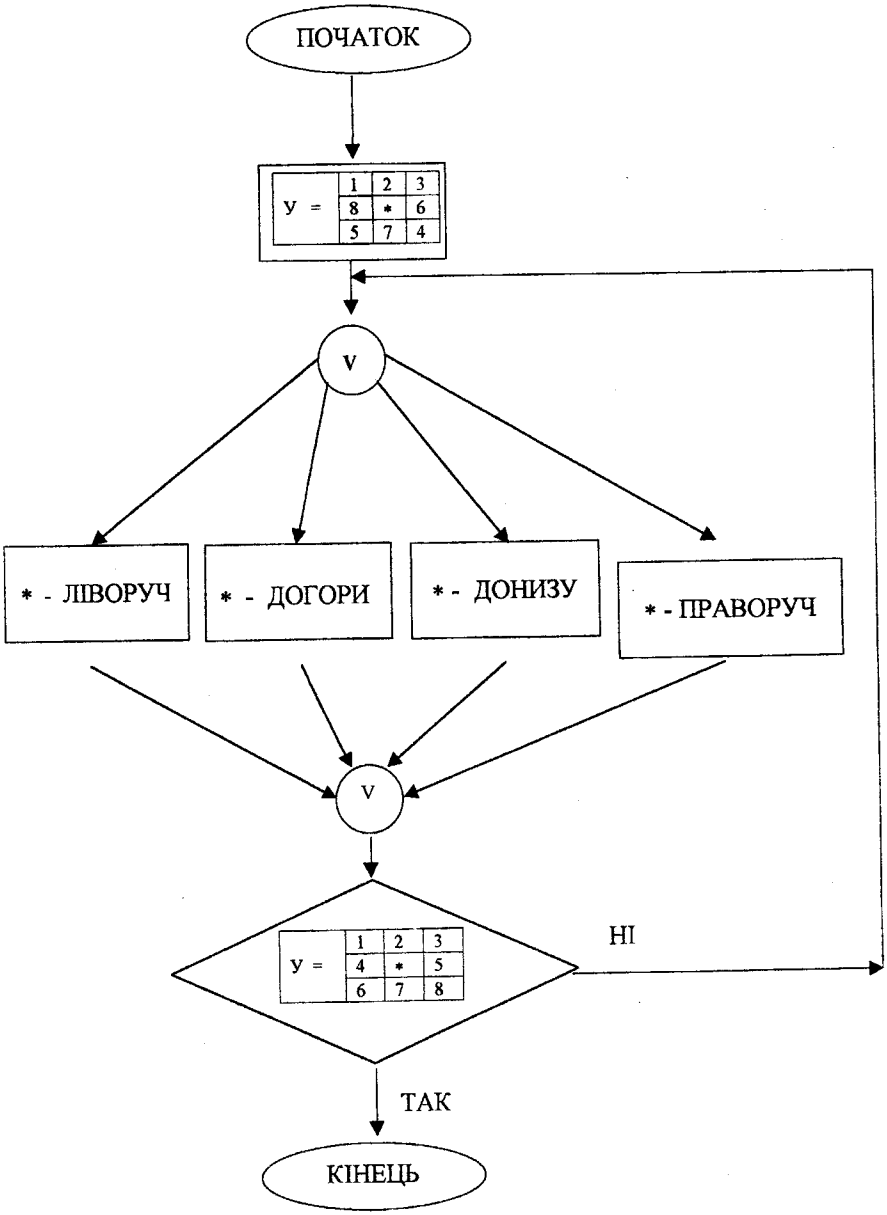


Рисунок 2.5 – Представлення задачі для гри 8 у недетермінованій формі

Приклади представлення задач

Поставити задачу - означає зрозуміти її умови, тобто усунути неповноту, надлишковість та неоднозначність, або, іншими словами, знайти для неї відповідне представлення. Кожному відомо, що часто розв'язати задачу допомагає її вдале графічне представлення або словесне переформулювання. Відомий фахівець в галузі ШІ, Г. Саймон взагалі стверджував, що розв'язати задачу - означає знайти таку форму її представлення при якій відповідь стає очевидною.

Розглянемо декілька прикладів, які підтверджують сказане [2,3].

1. 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 2.6. Треба визначити мінімальну послідовність ходів, які дозволять білим коням зайняти місця чорних, і навпаки.

При першому знайомстві з 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 (рисунок 2.6 а). Але, якщо зрозуміти, що в задачі не є важливим моделювання фізичного розміщення коней на дошці, а головним є лише урахування взаємозв'язків між ходами при переході від однієї позиції до іншої, то задача значно спрощується.

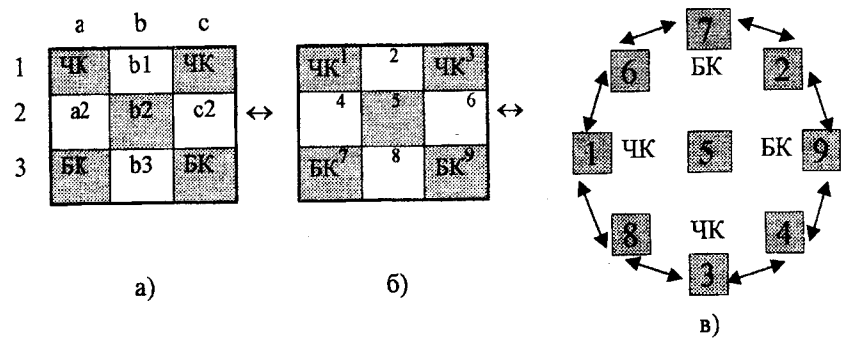


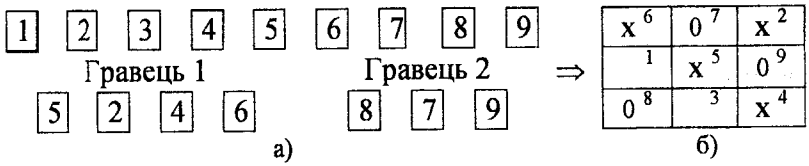
Рисунок 2.6 – Зміни представлень задачі про чотирьох коней

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. Таки поля "з точки зору" коня є для нього сусідніми. Якщо тепер пронумерувати поля дошки цифрами від 1 до 9, то сусідніми , наприклад,

прийдемо до нового представлення задачі, в якому дошка має форму кола і коні ходять по колу, пересуваючись за один хід на сусіднє поле (рисунок 2.6 в). Розв'язання задачі при такому представленні стає очевидним, необхідно лише повернути коло на 180°. Таке повертання відповідає чотирьом ходам кожного коня, тобто мінімальна кількість ходів необхідна для розв'язання задачі складає шістнадцять.

2. Гра "набери 15"

На столі розкладено дев'ять карток, кожна з яких містить цифру від 1 до 9 (Рисунок2.7). Гравці, які бачать цифри, по черзі беруть по одній картці. Виграє той гравець, який першим зможе показати рівно три картки, з тих що він взяв зі столу, сума цифр яких складає 15.



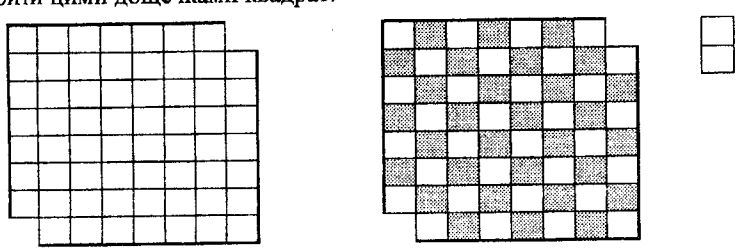
а) б)

Рисунок 2.7 – Зміни представлення гри в "Набери 15" у гру в хрестики - нулики

Спочатку важко сказати яку саме виграшну стратегію необхідно обрати в цій грі і чи існує вона взагалі. Але, якщо побачити, що ця гра повністю ізоморфна грі в хрестики - нулики, поле якої представлено магічним квадратом (рисунок 2.7 б), то стратегія гри стане очевидною.

3. Покриття геометричних фігур

Задано квадрат розміром 8×8 полів, в якому вирізані діагональні поля (рисунок 2.8 а). Є також 31 дощечка розміром 1 х 2 поля. Необхідно покрити цими дощечками квадрат.



а) б)

Рисунок 2.8 – Задача про покриття квадрата дощечками

Задача стає дуже простою, якщо пофарбувати поля квадрата в білий і чорний кольори, як показано на рисунок 2.8 б. Оскільки кількість білих і чорних полів при цьому не співпадають, можна одразу сказати, що задачу розв'язати неможливо.

Приклади представлення задач у просторі станів

1. Задача комівояжера

Класична комбінаторна проблема. 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). В нашому випадку комівояжер повинен побувати у чотирьох містах, починаючи з пункту А (рисунок 2.9).

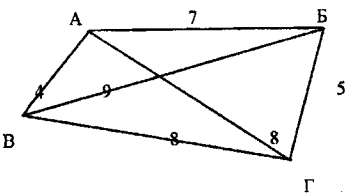


Рисунок 2.9 – Вихідні умови задачі про комівояжера

а) *Опис станів*. Стани задачі будемо задавати списками міст, які відвідані на даний час. Початковим станом буде список (А). Дозволеними є лише такі описи станів, в яких кожне місто повторюється не більш як один раз, за винятком того, що останнім в списку може бути знов записане місто А.
б) *Оператори.* Оператори є обчисленнями, які відповідають таким подіям: (1) - прямувати до міста А; (2) - прямувати до міста В і т. ін. Оператор не може бути застосовано до деякого опису стану в тому випадку, якщо він перетворює його у деякий неприпустимий опис. Наприклад, оператор (1) не може бути застосований ні до якого опису, що не містить назви усіх міст.
в) *Критерій досягнення мети.* Будь-який опис, що розпочинається і закінчується назвою міста А і містить назви всіх інших міст, задовольняє поставлену мету.

Простір станів задачі можна представити у вигляді графа (рисунок 2.10), перевагою якого в даному випадку є зручність обчислень повної довжини будь-якого маршруту, яка складає 24 для шляхів АБГВА і АВГБА.

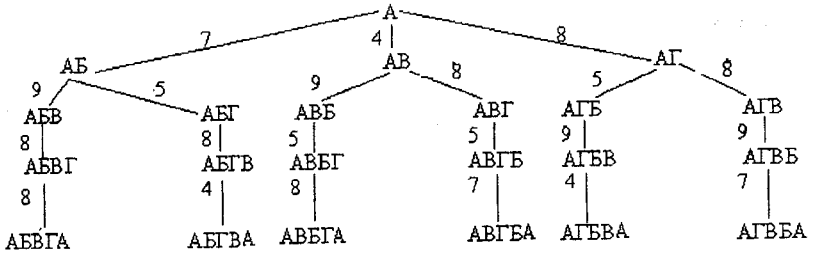


Рисунок 2.10 – Представлення задачі про комівояжера у просторі станів за допомогою графа

2. Задача синтаксичного аналізу

З такою задачею часто стикаються при роботі з формальними мовами. При цьому спочатку задається деяке формальне визначення мови шляхом задання відповідної граматики, яка виділяє певний клас рядків символів. Після цього виникає питання, чи належить даному класу деякий рядок?

Розглянемо граматику, в якій речення визначається як рядок 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 такі: aab, abaabab, aaaaab і т. і.

Приклади рядків, що не є реченнями: aaa, aba, abaa, і т. і.

Припустимо, треба визначити, чи є рядок abaabab реченням даної мови. Сформулюємо задачу в просторі станів:

а) *Опис станів*:

- початковий стан abaabab;
- множина дозволених станів;
- множина рядків, що утворюються з початкового при використанні тих правил переписування, якими визначаються оператори.

б) *Оператори*. Позначимо через S будь-яке припустиме в даній мові речення, а через $ будь-який підрядок будь-якого рядка. При цьому оператори перетворення станів можна буде представити в такому вигляді:

I: $\$_1ab\$_2$ -> $\$_1S\$_2$ (підрядок ab може бути замінений реченням S);

II: $\$_1aS\$_2$ -> $\$_1S\$_2$ (за символом a наступним є деяке речення);

III: $\$_1Sb\$_2$ -> $\$_1S\$_2$ (за деяким реченням наступним є символ b);

IV: $\$_1SS\$_2$ -> $\$_1S\$_2$ (за деяким реченням наступним є інше речення).

Дані правила описують граматику, яка визначає поняття речення.

в) *Критерій досягнення мети*. Опис

символу S.

Одна з послідовностей станів, що надає роз'вязок задачі має такий вигляд:

Saabab (1); SaSab(1); SSab(2); SSS (1); SS(4); S(4).

Граф, що відображає простір станів для цієї задачі, показано на рисунку 2.11.

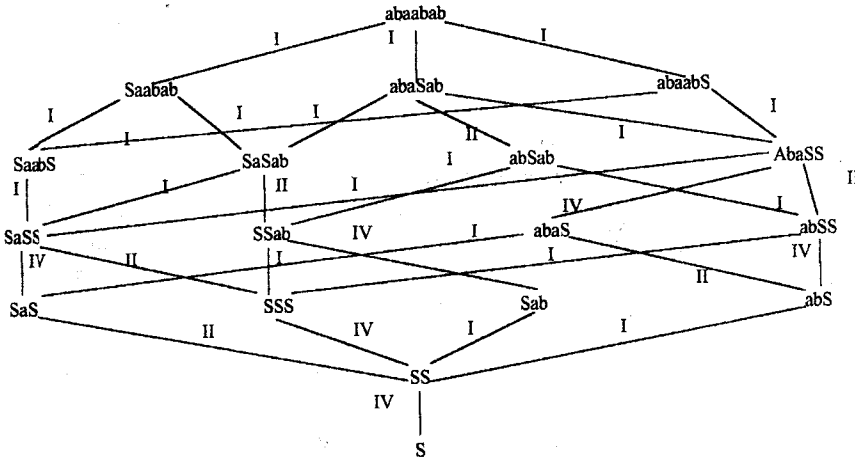


Рисунок 2.11 – Простір станів для задачі синтаксичного аналізу

3. 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.

В даному випадку доцільним є використання в опису станів змінних. При цьому вирази із змінними можуть описувати цілі множини станів. Конкретні описи станів отримують підстановкою конкретних значень.

Вираз, який містить змінні для опису станів називається схемою для опису станів. Розглянемо принципи використання схем для опису станів на прикладі задачі про мавпу і банани [1,4].

а) *Опис станів*. Очевидно, в опису станів повинні з'явитися такі елементи, як: координати мавпи у кімнаті (вертикальні та горизонтальні); координати ящика та наявність бананів у мавпи. Такі елементи можна представити списком з чотирьох елементів:

(w, x, y, z),

де w - двомірний вектор координат мавпи на горизонтальній площині;

x - 1 або 0 в залежності від того, на ящику чи ні знаходиться мавпа;

y - координати ящика (двомірний вектор);

z - 1 або 0 в залежності від того, отримала мавпа банани, чи ні.

Якби кожне значення списку (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.

Інший шлях полягає у використанні схеми опису станів.

б) *Оператори*. Відповідають у задачі чотирьом можливим діям мавпи:

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

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

III: ***залізти*** (на ящик);

IV: ***схопити*** (банани).

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

підійти (u)

(w,0,y,z) ----------> (u,0,y,z)

пересунути (v)

(w,0,w,z) ----------> (v,0,v,z)

залізти

(w,0,w,z) ----------> (w,1,w,z)

схопити

(c,1,c,0) ----------> (c,1,c,1),

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

в) *Множина цільових станів*. Будь-який список, останній елемент якого одиниця - є цільовим станом.

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 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 влізе на ящик, це призведе 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. Граф, який відображає простір станів (Рисунок2.10), є невеликим і в ньому не важко відшукати шлях розв'язання задачі. Підстановкою на місце

окремих значень, як це показано на Рисунок 2.10, отримаємо потрібну послідовність операторів: ***підійти (b)***, ***пересунути (c)***, ***залізти***, ***схопити***.

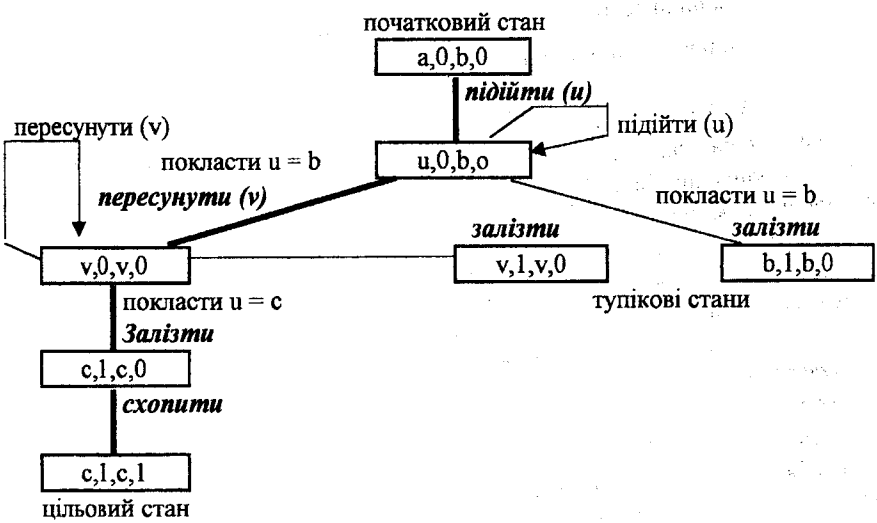


Рисунок 2.10 – Простір станів для задачі про мавпу і банани

Питання для самоперевірки

1. Наведіть Ваше особисте визначення інтелекту.
2. Що розуміється під задачею в штучному інтелекті?
3. Наведіть п'ять власних прикладів задач, які важко формалізувати.
4. Які основні види задач повинна вміти розв'язувати система штучного інтелекту?
5. Які саме переваги надає використання комп'ютерів при дослідженнях в галузі штучного інтелекту?
6. Що розуміється під представленням задачі?
7. Наведіть власний приклад задачі, ізоморфне перетворення якої значно полегшує її розв'язання.
8. Поясніть поняття формулювання замкненої задачі.
9. Які основні форми представлення задач Ви знаєте?
10. Наведіть приклади різних форм представлення опису станів.
11. На власному прикладі поясніть поняття інфіксної та префіксної форм запису алгебраїчних виразів.
12. Що необхідно зробити, щоб представити задачу у просторі станів?

13. Поясніть поняття недетермінованого алгоритму.
14. Наведіть основні терміни, переваги та недоліки представлення задач за допомогою графів.
15. Що таке схема опису станів.
16. Наведіть власні приклади задач, які недоцільно або неможливо представити у просторі станів. Поясніть чому саме.

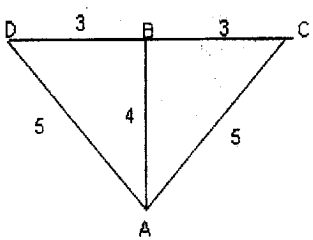
Завдання для самостійної роботи

Знайдіть форму опису станів, оператори і критерій досягнення мети для наведених нижче задач.

1. 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 ні хто зі зброєносців не знаходився разом з чужими лицарями без свого хазяїна.
2. Здійсніть перетворення позицій:

Шашку можна пересувати на сусіднє поле. Дозволяється стрибати через сусідню шашку, якщо за нею є вільне поле. Білі і чорні шашки повинні просуватись назустріч без повернень назад.

3. Є два джерела: з А витікає 1000 літрів води за хвилину, В - 500 літрів. Вони повинні забезпечити два басейни: С та D, - потреба кожного з яких є 750 літрів за хвилину. Вода може подаватися по трубах з максимальною пропускною здатністю 750 літрів за хвилину. Джерела і басейни розташовані таким чином, як показано на рисунку. Як треба з'єднати труби, щоб їх загальна довжина була найменшою



4. 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. Забезпечте переправу всіх солдат через ріку.
5. В квадраті, який має 16 комірок, треба розмістити 16 літер (4 літери А, 4 літери В, 4 літери С, 4 літери Е) так, щоб на кожній вертикалі і на кожній горизонталі будь-яка літера зустрічалась тільки один раз.
6. Дано два глечики: перший - з водою (v = 5л), другий - пустий (v = 2л). Необхідно отримати рівно 1л в глечику ємністю 2л. Воду можна виливати і переливати з одного глечику до іншого.

7. 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.
8. Розташуйте на шаховій дошці вісім ферзів так, щоб ні один з них не знаходився під ударом іншого.
9. Є три кілка на одному з яких знаходяться три диски в порядку зменшення їх розмірів. Необхідно перекласти диски з першого кілка на третій враховуючи такі правила:

   - на кожному ході один з дисків повинен переміщуватись з одного кілка на інший;

   - диск більшого діаметра заборонено класти на диск меншого діаметра.
10. По колу розташовані 5 лунок. На початку гри одна лунка, яка має назву "рума" є пустою, а кожна з інших лунок містить по 2 кулі. На кожному ході всі кулі, що знаходяться в деякій лунці A, розкладаються по одному в сусідні лунки за ходом годинникової стрілки. Перший хід робиться з будь-якої лунки. Якщо при черговому ході остання куля потрапляє до руми, то наступний хід можна зробити з будь-якої лунки крім руми, а якщо не потрапляє - то наступним ходом розподіляються кулі з тієї лунки, до якої потрапила остання куля на попередньому ході. Якщо остання куля потрапляє в пусту лунку гра вважається програною. Мета гри полягає у зборі всіх куль до руми.
11. Запишіть у інфіксній формі та представте у вигляді двійкового дерева такий вираз: $[(A + B)\times C + (A - D/E)]\times F$.
12. Запишіть у інфіксній формі та представте у вигляді двійкового дерева такий вираз: $(A/G + B\times F) \times C - (A - D/E) \times H$.
13. 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.

14. 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.

15. 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 ABABC. При позитивній відповіді наведіть розв'язання задачі.
16. Покажіть, знайшовши відповідний шлях на графі що представляє простір станів, що рядок (((),()),(),((),())) є реченням S в граматиці, яка визначається такими правилами переписування: S ← (), A ← S, A ← A,A, S ← (A).
17. Представте в просторі станів задачу про комівояжера. Необхідно прокласти маршрут таким чином, щоб побувавши в кожному місці один раз повернутись до вихідного пункту. Бажано, щоб маршрут мав мінімальну довжину. Комівояжер повинен побувати у п'яти місцях A, B, C і D, починаючи з пункту A. Місця поєднані шляхами, які мають таку довжину: AB=7; AC=6; AD=10; AE=13; BC=7; BD=10; BE=10; CD=5; CE=9; DE=6.
Наведіть не менше двох варіантів опису станів.
18. Представте в просторі станів задачу про вісім ферзів (ферзі треба розташувати на шахової дошці таким чином, щоб вони не загрожували один одному). Намалюйте частину графа станів та надайте його вершинам і дугам відповідні пояснення.

3.1 Поняття формальної системи

Методи вірних міркувань, які є ланцюжками умовиводів в логічно послідовній формі, розробляються в формальній логіці. Міркування в ній вивчаються з точки зору форми, а не змісту. З цією метою в звичайних міркуваннях відокремлюють певні елементи, які можна довільно замінювати будь-якими іншими елементами. Розглянемо, наприклад, відому фразу (силогізм) "Людина смертна, Сократ - людина, отже Сократ смертний". Тут імена класів "людина", "смертний" і "Сократ" можна замінити будь-якими іншими і при цьому саме міркування залишиться формально вірним. Дійсно, отримаємо "Якщо x є y і якщо z є x, то z є y". Таким чином вже просте заміщення імен класів в міркуваннях на символьні позначення дозволяє будувати узагальнені судження на основі подібних міркувань. Символи, які можна змінювати називають змінними, а символи, які повинні бути фіксованими, називають операторами.

Формалізація процесів логічного виведення традиційно здійснюється за допомогою так званих формальних систем. Формальна система, це сукупність чисто абстрактних об'єктів (не пов'язаних з зовнішнім світом), в якій представлено правила оперування множиною символів в чисто синтаксичному трактуванні без будь-якого урахування змісту (семантики). Формальну систему (ФС) можна визначити як четвірку:

$$\Phi = <T,L,Q,R>,$$

де: T - кінцевий алфавіт (кінцева множина символів). Тобто T це множина базових елементів, цеглинок, які не можна роздробити на більш дрібні. Прикладами таких елементів можуть бути букви (графеми) або деталі дитячого конструктора. Єдина вимога до елементів множини T полягає в тому, що для будь-якого елемента за кінцеву послідовність кроків необхідно вміти визначати, чи належить він T чи ні, а також уміти відрізняти кожен елемент від інших і ототожнювати однакові елементи;

L - певна процедура побудови формул (або слів) ФС. Тобто, L - це множина синтаксичних правил, за допомогою яких з елементів множини T будуються більш складні синтаксично вірні новоутворення. Так, з букв утворюють лінійно впорядковані сполучення, які називають словами, реченнями, текстами; з деталей дитячого конструктора будують певні моделі;

Q - деяка підмножина формул, які називають аксіомами. Q містить виділені з якихось міркувань синтаксично вірні утворення.

R - кінцева множина правил виведення, які дозволяють отримати з деякої кінцевої множини формул, деяку іншу множину формул. Такі правила в загальному вигляді можна представити так: U_1 і U_2 і ... і $U_p \rightarrow W_1$ і W_2 і ... і W_n, де V_i та W_i - формули ФС, а стрілка $\rightarrow$ позначає "спричиняє" або "випливає".

В формальних системах формальне доведення визначається як кінцева послідовність формул M_1, M_2, ..., $M_к$ така, що кожна формула M_i є або аксіомою, або виведена за допомогою одного з правил виведення, з попередніх до неї формул M_a, де $a < i$.

Формула t називається теоремою, якщо існує доведення, в якому вона є останньою, тобто $M_r = t$. Зокрема, кожна аксіома є теоремою. Те, що Т є теоремою позначається як t.

Виділяють два типи правил виведення:

1) правила, що застосовують до формул, які розглядаються як єдине ціле (такі правила називають продукціями);

2) правила, які можна застосовувати до будь-якої окремої частини формули, причому самі ці частини є формулами, що входять до ФС. Такі правила називають правилами переписування.

Наприклад, в математиці правило виведення x<y і y<z обумовлює x<z застосовується до формули як до єдиного цілого - це є продукція. В продукціях слово обумовлює (спричиняє, викликає) замінюють символом $\rightarrow$.

На відміну від попереднього, правило x – x = 0 має сенс при будь-якому вигляді виразу, що входить до нього в якості x. Це є правило переписування і в цьому випадку слово "обумовлює" замінюється знаком $\mapsto$. І продукція, і переписування мають тільки один напрямок виведення – зліва направо.

3.2 Системи продукцій

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 представляють твердження відносно цих фактів, і виводяться безпосередньо з самих цих комбінацій фактів. Отже, в такому випадку продукції представляють не пари "ситуація-дія", а пари "посилка - висновок".

Велика кількість систем штучного інтелекту базується на понятті "формальна система продукцій". Вперше продукційні системи були запропоновані Постом у 1943 р. Він довів, що продукційна система є логічною системою, яка еквівалентна машині Т'юрінга, тобто вона є універсальною. 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 С продукцію $x_i W \Rightarrow W y_i$, означає викреслити з С 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 *С* буде одержано з D за допомогою продукції $x_i W \Rightarrow W y_i$.

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 заданого порядку. При виявленні першої ж придатної продукції вона застосовується до слова і змінює його згідно відповідного правила 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 (Ньюел, Саймон, 1972) показали, що людина в своїх міркуваннях використовує правила, які подібні до продукцій, тобто правила виду "умова - дія". Н'юел запропонував використовувати продукційні системи для моделювання на ЕОМ процесів прийняття рішень.

Згідно пропозиціям Н'юела, застосування продукційної системи забезпечує чітке розмежування між такими стандартними обчислювальними компонентами як дані, операції та управління. При цьому продукційну систему можна визначити таким чином:

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

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

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

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

Відмінність продукційних систем від традиційних полягає в тому, що глобальна база даних доступна усім правилам продукцій. Одні правила не викликають інших, 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 до програми та структур даних які вже існують, то продукційні системи є значно більш модульними і зміни у базі даних, системі управління чи правилах, можуть бути здійснені відносно незалежно одні від одних.

Будемо розрізняти декілька видів систем продукцій в залежності від того, які вони мають системи управління, які властивості правил та баз даних використовують, а також в залежності від засобів їх використання при розв'язанні конкретних задач.

Розглянемо більш детально поняття системи продукцій на прикладі гри 8.

Хай треба перетворити початкову ситуацію у цільову (рисунок 3.1):

2	8	3
1	6	4
7	*	5

$\Rightarrow$

1	2	3
8	*	4
7	6	5

Рисунок 3.1 - Приклад початкової та цільової ситуацій для гри 8

Рішенням тут є правильна послідовність ходів.

Для розв'язання задачі за допомогою системи продукцій треба визначити глобальну базу даних, набір правил та задати стратегію управління, тобто вирішити проблему представлення задачі. Для гри 8 дуже просто визначити елементи задачі відповідно до трьох компонентів - стани, ходи та мета задачі. Кожна конфігурація фішок у грі є станом задачі. Множина всіх можливих конфігурацій створює простір станів, який для гри 8 не є дуже великим і складає 9! = 362800 станів. Для описання станів можна обрати матрицю чисел розміром 3x3. Початкова глобальна база даних буде описом початкового стану.

Кожний хід перетворює один стан на інший. Зручно інтерпретувати гру 8 як таку гру, що має чотири ходи: пересування пустої клітинки в одному з чотирьох напрямків. Ці ходи визначаються

які відповідним чином застосовуються до описів станів. Кожне правило має попередню умову, якій повинен задовольняти опис стану для того, щоб правило можна було застосувати до нього. Наприклад, попередньою умовою для правила, яке відповідає за переміщення пустої клітинки догори, є вимога, щоб пуста клітинка не знаходилась у верхньому рядку поля.

Основний алгоритм управління системою продукцій для вирішення задачі типу гри 8 можна записати у такому вигляді [6]:

Procedure PRODUCTION

1. DATA ⇐ початковий стан глобальної бази даних;
2. **until** DATA задовольняє термінальній умові, **do**;
3. **begin**
4. **select** (обрати) деяке правило П з множини правил, яке можна застосувати до DATA;
5. DATA ⇐ результат застосування П до DATA;
6. **end**.

3.3 Режими управління системами продукцій

Розглянута процедура є недетермінованою, тому що в ній не надається точного визначення того, яке ж саме з придатних правил обирати на крокові 4. Те що зветься стратегією управління для системи продукцій включає правила виведення та принципи зберігання вже випробуваних послідовностей правил і описів станів бази даних, які породжувались застосуванням цих правил. В більшості задач штучного інтелекту ні одна з доступних стратегій управління не дозволяє вибрати найбільш задовільне правило при кожному зверненні до кроку 4. Тому роботу систем продукцій можна характеризувати як процес пошуку, в якому всі правила випробовуються до того часу, доки не буде знайдена така послідовність правил, яка породжує базу даних, що задовольняє термінальну умову. Ефективні стратегії управління передбачають наявність достатньої інформації про задачу, щоб сформулювати на крокові 4 найбільш відповідне правило.

Розрізняють два основних типи стратегій: беззворотний та пробний, який, в свою чергу розподіляється на режим із поверненням та режим з пошуком на графі.

При беззворотному режимі обране правило використовується беззворотно, без можливості перегляду в подальшому.

В пробному режимі при використанні правила резервується можливість надалі знов повернутись до тієї ж самої ситуації, щоб мати змогу випробувати деяке інше правило.

У режимі із поверненням при виборі кожного правила визначається деяка точка повернення. Якщо наступні обчислення призведуть до труднощів в одержанні розв'язку, то процес обчислення повертається до попередньої точки повернення і продовжується з неї.

В режимі з пошуком на графі передбачається зберігання результатів виконання одночасно декількох послідовностей правил і використовуються спеціальні методи пошуку на графах.

Беззворотний режим управління використовують при наявності вірогідної локальної інформації, яку можна використовувати для побудови явної глобальної інформації про вирішення задачі.

Найбільш відомим прикладом реалізації режиму є метод "підйому у гору" при пошуку максимуму функції. Згідно з цим методом в кожній точці здійснюється пересування у напрямку найбільшої крутизни (локальна інформація), щоб знайти кінець - кінцем максимум функції (глобальна інформація). Є певний клас функцій для яких знання про найбільш крутий підйом достатній для пошуку рішення. Для організації "підйому вгору" треба мати деяку дійсну функцію визначену на глобальній базі даних. Стратегія управління використовує цю функцію для беззворотного вибору правила, яке породжує базу даних, що дає максимальне зростання значення цієї функції. При цьому максимальне значення функції "підйому вгору" має бути таким, що задовольняє термінальній умові.

Наприклад, у грі 8 в якості функції від опису станів можна взяти кількість фішок (з від'ємним знаком), які стоять не на своїх місцях порівняно з описом цільового стану. Результат застосування такої функції для нашого прикладу приведений на рисунок 3.2.

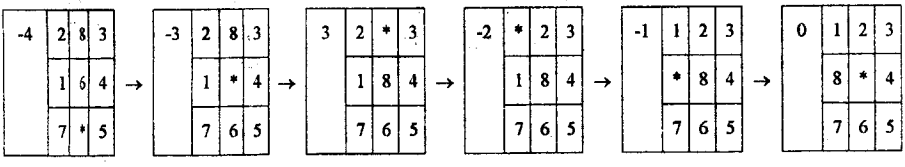


Рисунок 3.2 - Приклад застосування функції оцінювання для гри 8

Можна побачити, що в жодному випадку значення функції не збільшилось. Але це не є забороненою умовою. Процес зупиняється тільки в тому випадку, коли не знаходиться правил, які б не зменшували значення функції.

Однак, метод "підйому вгору" дав результат лише у деякій конкретній початковій ситуації прикладу, що розглядається. В загальному випадку подібні функції можуть мати багато локальних максимумів, що робить метод некоректним.

Наприклад, для початкового стану (рисунок 3.3):

1	2	5
3	*	4
7	6	8

Рисунок 3.3 – Приклад початкового стану для гри 8

будь-яке дозволене правило буде зменшувати значення функції "підйому вгору", тому що початковий стан розташовано на локальному (але не глобальному) максимумі функції. Наявність таких "плато" та "хребтів" зводить до мінімуму можливості застосування цього методу, хоча труднощі з "плато" та "хребтами" можна вирішити в загальному випадку вибором іншої функції, яка б мала лише один максимум (глобальний) і не мала ні одного "плато".

Крім того існують випадки, коли вибір "сумнівного" правила не виключає можливість вибору у подальшому іншого, придатного правила, ми ні чим не ризикуємо беззворотно використовуючи правило.

Режим із поверненням. В більшості задач використання "непридатного" правила може зашкодити успішному закінченню процесу розв'язання, або істотно його затримати. В цих випадках має перевагу стратегія управління, яка дозволяє випробувати деяке правило, і якщо пізніше виявиться що воно було непридатним повернутись назад і спробувати використати замість нього якесь інше правило, забувши при цьому всі кроки, які йшли за невдалим правилом. З формальної точки зору стратегію з поверненням можна використовувати незалежно від того наскільки повну інформацію відносно вибору правил ми маємо. Навіть якщо інформація взагалі відсутня, існує можливість вибору правил за деякою довільною схемою. Така можливість забезпечується завдяки реалізації механізму повернення, який дозволяє шляхом перебору вибрати кінець кінцем придатне правило, якщо воно взагалі існує. Зрозуміло, що при наявності вірогідної інформації про те, як вибирати правила, повернень буде менше і процес буде більш ефективним.

Наприклад, для гри 8 можна обрати такий порядок застосування правил до кожного стану: спочатку пересунути пусту клітинку ліворуч, потім угору, далі праворуч і далі до низу. Повернення буде виконуватись кожен раз коли:

а) породжується опис стану, який вже зустрічався раніше на шляху від опису початкового стану;

б) було застосовано деяку довільно обрану кількість правил (задана глибина пошуку), але цільовий стан не було досягнуто;

в) більше не існує придатних правил.

Кінець - кінцем рішення буде знайдено, але якщо задати малу межу глибини, цього може й не статись.

Режим із поверненням буває більш ефективним, якщо здійснювати не "сліпий" пошук, а направляти пошук за допомогою відповідної інформації, якій можна довіряти. Так, можна використати деяку функцію з методу "підйому вгору" як засіб вибору правил. Якщо при беззворотному режимі управління "підйом вгору" може загрузнути у локальному максимумі, то механізм повернення залишає можливість в даному випадку для пошуку альтернативних шляхів.

Пошук на графі. Графи, а точніше дерева, є дуже корисними структурами для зберігання інформації про використання декількох послідовностей правил. Припустимо, ми вирішили використати режим управління з пошуком на графі для розв'язання нашої задачі. Ми можемо зберігати різні правила, що ми вже застосували, та відповідні породжені бази даних за допомогою структури, яка зветься деревом пошуку. У корені такого дерева пошуку знаходиться опис початкової ситуації. Таке дерево буде нарощуватись шляхом породження за допомогою застосування правил нових вершин, доки не буде породжена база даних, що задовольняє термінальну умову. Тут йдеться про повний перебір, який є вкрай неефективним. Розумна стратегія повинна використовувати деякі специфічні для задачі знання, щоб отримати більш вузьке дерево пошуку, тобто сфокусувати зростання дерева у напрямку більш близькому до мети.

Представлення за допомогою графів використовується тільки в режимі із поверненням. При цьому дерево пошуку не зберігається цілком. Зберігається лише інформація про поточний шлях, достатня для його зміни у випадку виникнення такої необхідності.

Підкреслимо, що беззворотному режиму управління відповідає єдиний шлях у дереві рішення.

3.4 Зворотні і двосторонні системи продукцій

Система продукцій для гри 8 працювала безпосередньо від початкового стану до цільового. В зв'язку з цим вона зветься прямою системою продукцій. Можна вирішувати задачу і в зворотному напрямку, використовуючи зворотні ходи та просуваючись від цільового стану до початкового. Кожний зворотний хід буде породжувати при цьому підцільовий стан, з якого безпосередньо цільовий стан може бути досягнуто за один хід уперед. При цьому система продукцій просто змінить призначення станів і мети та буде використовувати правила, що відповідають зворотним ходам.

Якщо в задачі можна визначити чіткі стани і цілі, і вирішено використовувати описи станів як глобальну базу даних, то така система зветься прямою системою продукцій. При цьому правила, які використовують для отримання нових станів звуть П-правилами. Навпаки, якщо в якості глобальної бази даних використовують опис цільових умов задачі, то така система називається зворотною системою продукцій, а правила, які застосовуються для опису цільових станів та для породження опису підцілей - О-правилами. В розглянутому нами прикладі існує лише по одному початковому і цільовому стану, тому у даному випадку не має значення в якому напрямку вирішувати задачу. Однак, існує правило - якщо початковий стан один, а цільових багато - треба використовувати пряму систему продукцій. При спробі розв'язати у цій ситуації задачу в зворотному напрямку не існує змоги сказати, який з цільових станів є найближчим до початкового, тому прийшлося би розпочинати повний перебір станів.

Інколи вдалим буває рішення про двобічний пошук. Для цього в глобальну базу даних заносять опис як початкових станів, так і цільових. До однієї частини описів при цьому застосовують П-правила, а до іншої - О - правила. Термінальна умова визначається як умова відповідності між частиною опису станів та опису мети у глобальній базі даних.

3.5 Системи продукцій, які розкладаються

Розглянемо систему продукцій з початковою базою даних БД(C,B,Z), правила продукцій якої засновані на таких правилах переписування:

$$П1: C \Rightarrow (D, L);$$
$$П2: C \Rightarrow (B, M);$$
$$П3: B \Rightarrow (M, M);$$
$$П4: Z \Rightarrow (B, B, M),$$

а термінальною умовою є наявність у базі даних тільки символів M.

Існує багато еквівалентних шляхів одержання потрібної бази даних. Існування багатьох шляхів може привести до зниження ефективності системи - стратегія управління передбачає розгляд їх усіх або, що ще гірше, і тих шляхів які не ведуть до успіху.

Запобігти розгляду надлишкових шляхів можна, якщо помітити, що початкова база даних розбивається на окремі компоненти, які можна обробляти незалежно одну від одної. Дійсно, у прикладі початкова база даних розкладається на складові C, B та Z . Правила продукцій можна застосувати незалежно до кожної з цих складових. Результати цих операцій

можна знов піддати декомпозиції і. т. д., доки кожна компонента не буде вміщувати тільки символ М.

Тобто, в даному випадку, можна представити глобальну базу даних як молекулу, що складається з окремих атомів, які з'єднуються тим чи іншим способом. Якщо умови застосування правил передбачають перевірку тільки окремих атомів, і якщо наслідком застосування цих правил буде заміна кожного атома, що розглядається, на деяку нову молекулу, що в свою чергу теж складається з декількох атомів, то нову молекулу можна теж розбити на її атомарні компоненти і працювати незалежно з кожною компонентою. Кожне правило впливає лише на ту компоненту глобальної бази даних, яка задовольняє попередній умові цього правила. Оскільки деякі правила застосовуються паралельно, порядок їх використання не має значення.

Однак, для розкладення бази даних необхідно розкласти і відповідну термінальну умову. Тобто, щоб працювати з кожною складовою окремо, треба зуміти виразити глобальну термінальну умову через термінальні умови кожної складової. Найбільш важливим випадком при цьому є той, коли глобальну термінальну умову можна виразити у вигляді кон'юнкції термінальних умов кожної складової бази даних. В подальшому завжди буде мати на увазі цей випадок, якщо не буде застережене зворотне.

Системи продукцій, глобальні бази даних і термінальні умови яких дозволяють декомпозицію, називаються такими, що розкладаються. Основна процедура для системи продукцій, що розкладається, може бути, наприклад, такою [6]:

Procedure SPLIT

1. DATA ⇐ початкова база даних
2. $\{D_i\}$ ⇐ декомпозиція бази даних DATA. Тепер кожна D_i розглядається як окрема база даних
3. **until** усі $\{D_i\}$ задовольняють термінальній умові, то **do:**
4. **begin**
5. **select D*** з тих $\{D_i\}$, які не задовольняють термінальну умову
6. **delete D*** з $\{D_i\}$
7. **select** деяке правило П, яке можна застосувати до D*
8. D ⇐ результат застосування П до D*
9. $\{d_i\}$ ⇐ декомпозиція D
10. **add** $\{d_i\}$ до $\{D_i\}$
11. **end**

Для описання роботи системи продукцій у режимі декомпозиції (редукції) задачі, використовують структури, що називаються І/АБО-графами. Такий граф складається з вершин, які відображають стани глобальної бази даних.

Вершини, які відображають бази даних, що можуть розкладатись, мають множини вершин-нащадків, кожна з яких відображає одну з складових. Ці вершини називаються вершинами типу І, бо для повної обробки бази даних, що може розкладатись, треба обробити всі її складові. Вершини І позначаються спеціальними дужками, які об'єднують дуги графа, що виходять з них.

До кожної одержаної в результаті декомпозиції складової бази даних можна застосувати деякі правила. Вершини, які відображають ці складові бази даних мають вершини-нащадки, що відображають бази даних, отримані в наслідок застосування правил. Такі вершини називаються вершинами типу АБО, оскільки для повної обробки складової бази даних, достатньо обробити будь-яку одну базу даних, яка була породжена в результаті застосування лише одного з дозволенних правил.

Для наведеного прикладу граф І/АБО буде мати вигляд, який показано на рисунку 3.4. У прикладі всі складові бази даних, що задовольняють термінальну умову, взяті у подвійну рамку. Ці вершини називаються термінальними. Підграф вирішення задачі показано на графі товстими лініями. Він являє собою граф, "кінцеві вершини" якого відображають множину баз даних, кожна з яких задовольняє термінальну умову.

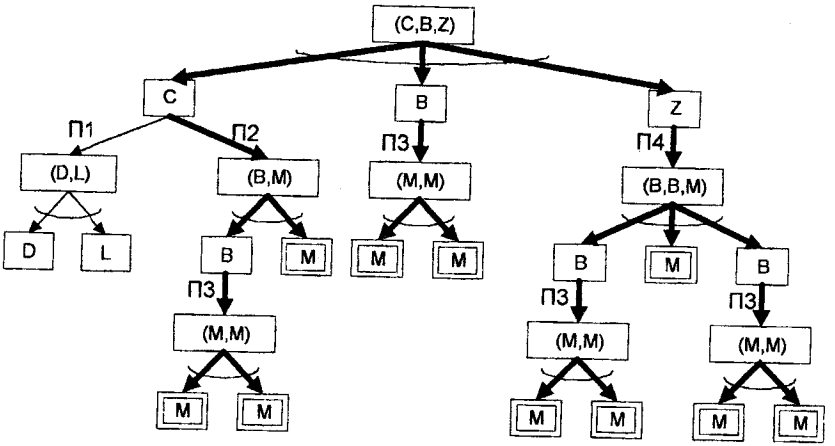


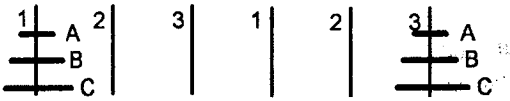
Рисунок 3.4 - Граф І/АБО

3.6 Представлення задач, які можна звести до підзадач

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

Є три кілки 1, 2, 3 та три диски різного розміру: A, B, C. Необхідно перекласти диски з першого кілка 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 мають, відповідно, такий вигляд: (рисунок 3.5)



а) б)

Рисунок 3.5 – Початкова (а) та кінцева (б) конфігурація задачі про ханойську вежу

Таку задачу можна вирішити методами, які використовують простір станів. Граф простору станів для неї буде мати 27 вершин, кожна з яких відображає одну з дозволених конфігурацій розташування дисків на кілках. На рисунку 3.6 стани відображаються трирозрядними векторами, значення першого розряду яких збігається з номером кілка на якому розташований диск С, значення другого розряду – з номером кілка на якому розташований диск В і значення третього розряду – з номером кілка на якому розташований диск А.

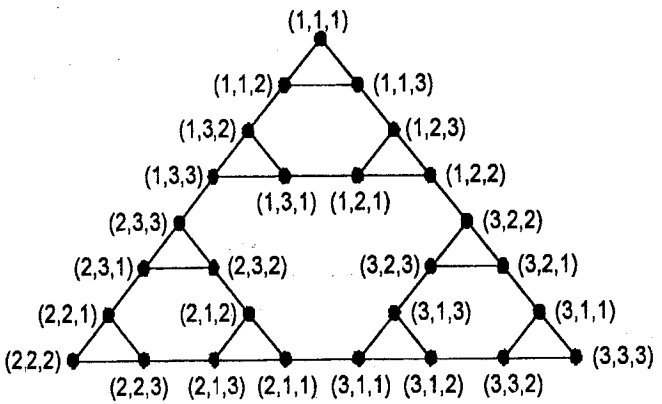


Рисунок 3.6 – Граф простору станів для задачі про ханойську вежу

Цю задачу можна також розглядати як задачу, яка направлена на досягнення трьох підцілей:

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

Але ці підцілі не є незалежними. Наприклад, можна одразу перекласти диск А на кілок 3 і перша підціль буде досягнута. Але в цьому випадку неможливо стане досягнути дві інші підцілі. (якщо не відмінити першу дію). Але в даному прикладі 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 кілок 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 пдціль "диск С на кілок 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 3, потрібно перекласти туди С. При цьому на диску С не повинні знаходитись ніякі інші диски. Тобто, необхідно забезпечити можливість перекласти диск С з кілка 1 на кілок 3.
2. Перекласти диск С з кілка 1 на кілок 3.
3. Досягнути дві підцілі, які залишились (перекласти диски А і В на кілок 3).

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

1. Зняти диски А і В з кілка 1, при цьому розташувати їх не на кілку 3, а на кілку 2.
2. Виконати переміщення диску С з кілка 1 на кілок 3.
3. Перемістити диски А і В з кілка 2 на кілок 3.

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

1. Задача про переміщення двох дисків (А і В) на кілок 2.
2. Задача про переміщення одного диску (С) на кілок 3.
3. Задача про переміщення двох дисків (А і В) на кілок 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 диску С. Використовуючи аналогічний ланцюжок міркувань задачі 2 і 3 також можливо звести до тривіальних, як це зображено на рисунку 3.7 (саме така ж схема редукції задачі може бути застосована і для початкової конфігурації з будь-якою кількістю дисків):

Відмітимо, що наявність двох типів вершин викликає необхідність розробки власних процедур пошуку рішень для І/АБО графів, які відрізняються від процедур пошуку на звичайних графах. На мові І/АБО графів застосування одиночного оператора редукції задачі означає, що спочатку будується проміжна АБО вершина, а потім безпосередньо наступні за нею І вершини. Вершини І/АБО графу, що відповідають описам елементарних задач, називаються заключними, або термінальними вершинами.

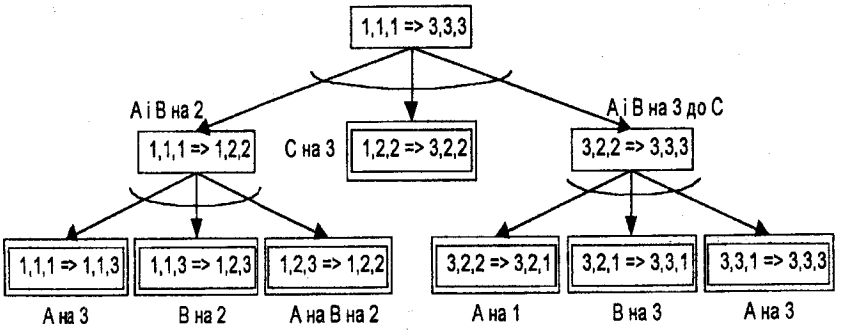


Рисунок 3.7 – Граф зведення задачі про ханойську вежу до підзадач

Метою пошуку на І/АБО графі є доведення розв'язуваності вершини.

Загальне визначення розв'язуваності вершини І/АБО графу рекурсивно формулюється таким чином:

- завершальні вершини розв'язувані (оскільки вони відповідають елементарним задачам);
- вершина типу АБО є розв'язуваною тільки в тому випадку, коли розв'язною є хоча б одна з її дочірніх вершин;
- вершина типу І є розв'язною тільки в тому випадку, коли розв'язними є кожна з її дочірніх вершин.

При цьому граф роз'вязання визначається як підграф з розв'язних вершин, які показують, що початкова вершина є розв'язною.

На рисунку 3.8 показано приклад І/АБО графа, в якому розв'язні вершини відображені у вигляді чорних кіл, а граф розв'язання позначений товстими лініями.

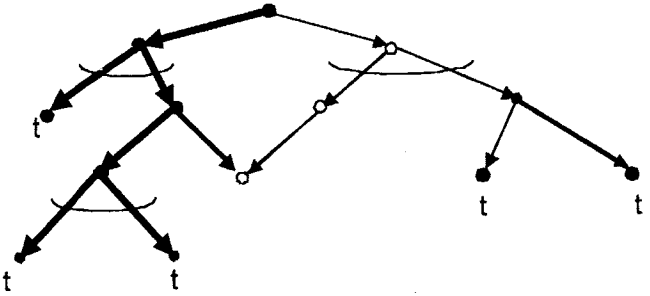


Рисунок 3.8 – Приклад І/АБО графу

Нерозв'язними є такі вершини І/АБО графа, які не є завершальними і, при цьому, за ними відсутні наступні вершини. На рисунку 3.8 такі вершини позначені білими колами. Поява нерозв'язних вершин може означати, що і інші вершини графа, в тому числі і початкова, можуть виявитись нерозв'язними.

Загальне визначення нерозв'язності вершини І/АБО графу рекурсивно формулюється таким чином:

- вершини, що не є завершальними і не мають наступних за ними вершин, є нерозв'язні;
- вершина типу АБО є нерозв'язною тільки в тому випадку, якщо нерозв'язною є кожна з її дочірніх вершин;
- вершина типу І є нерозв'язною тільки в тому випадку, якщо нерозв'язною є хоча б одна з її дочірніх вершин.

Неявне визначення І/АБО графів здійснюється за допомогою опису початкової задачі і операторів редукції задачі. Процес розв'язання задачі полягає в даному випадку в тому, щоб побудувати частину І/АБО графа, достатню для визначення того, чи є початкова вершина розв'язною.

Прикладом використання методу редукції задачі може служити задача знаходження невизначеного інтегралу:

$$F(x) = \int \frac{x^2 - 2x + 1}{x - 1} dx$$

Дерево розв'язання цієї задачі буде мати такий вигляд (рисунок 3.9):

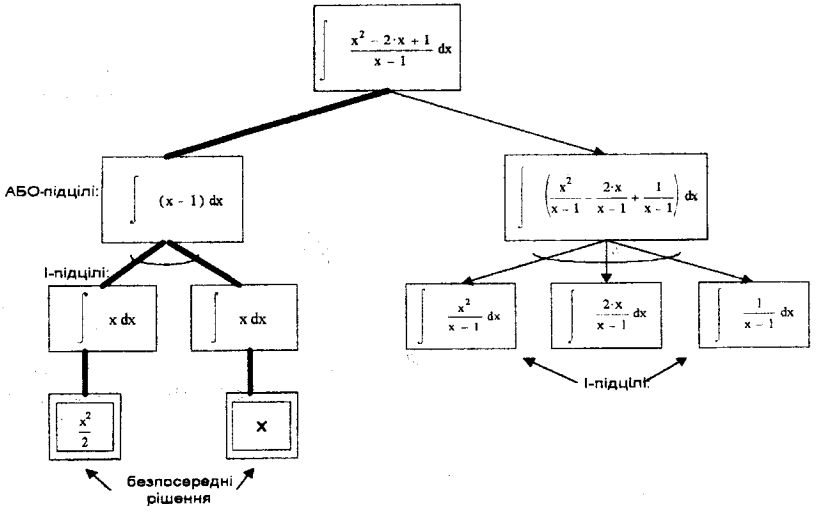


Рисунок 3.9 – Приклад І/АБО графа для обчислення невизначеного інтеграла

3.7 Комутативні системи продукцій

При деяких умовах порядок, в якому множина дозволених правил застосовується до бази даних, буває неістотним. В цьому випадку ефективність роботи системи продукцій можна значно підвищити за рахунок відмови від вивчення надлишкових шляхів вирішення, які збігаються в усьому за винятком послідовності застосування правил.

Розглянемо, наприклад, три правила П1, П2, П3 - які застосовуються до бази даних (рисунок 3.10). Після застосування будь-якого з цих правил, кожне з них залишається дозволеним для застосування до знову отриманої бази даних. Крім того, та ж сама база даних СС створюється? незалежно від послідовності застосування правил з множини {П1, П2, П3}. Система продукцій називається комутативною, якщо вона має такі властивості відносно будь-якої бази даних Б:

1. Кожне з множини правил, які дозволено застосовувати до Б, можна застосовувати також і до будь-якої бази даних, що була отримана з Б шляхом попереднього використання будь-якої послідовності дозволених на Б правил.

2. Якщо цільова умова задовольняється базою Б, то вона також задовольняється будь-якою базою даних, отриманою при використанні будь-якого дозволеного на Б правила.

3. База даних, яка була отримана в результаті застосування до Б будь-якої послідовності дозволених на Б правил, інваріантна відносно будь-яких переміщень у цій послідовності.

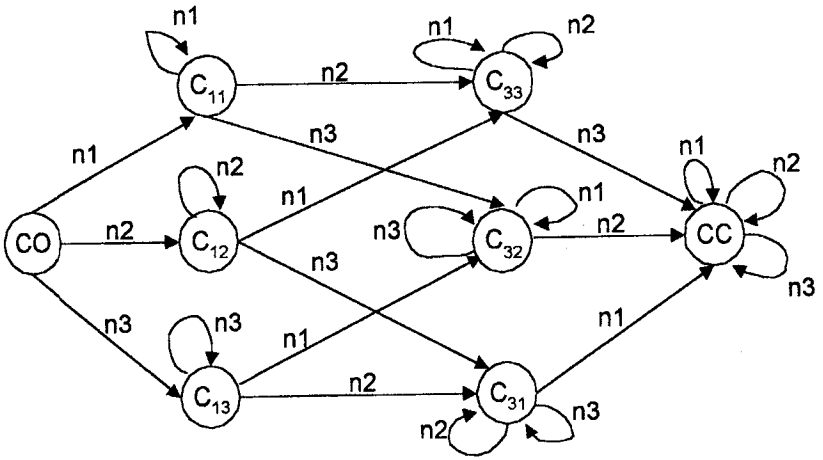


Рисунок 3.10 – Комутативна система продукцій

В прикладі (рисунок 3.9), що розглядається, процес застосування правил {П1, П2, П3} має властивість комутативності. Для отримання бази даних СС достатньо розглянути лише будь-який один з численних шляхів, що наведені у прикладі. Для комутативних систем, таким чином, важливими є методи виключення надлишкових шляхів. Треба підкреслити, що у комутативній системі об'єднати у будь-яку послідовність можна тільки ті правила, що були дозволені в початковій базі даних. Справа полягає у тому, що після застосування деякого правила до бази даних, дозволеними до застосування можуть стати додатково ще деякі правила, які не були дозволені спочатку. На такі правила дозвіл на зміну порядку їх застосування не поширюється.

Важливою властивістю комутативних баз даних є те, що в них завжди можна використовувати беззворотний режим управління, тому що застосування будь-якого правила ніколи не викликає необхідність його скасування. Кожне правило, що було дозволено на одній з попередніх баз даних, буде дозволеним і на існуючій базі. Тут не треба вводити механізм застосування різних послідовностей правил. Невдале застосування правила лише затримує процес розв'язання, але не робить його неможливим. Крім того, після отримання розв'язку виникає можливість видалити зайві правила з послідовності правил, яка описує розв'язання.

Помітимо, що будь-яка система продукцій може бути перетворена в комутативну. Для цього достатньо зробити глобальною базою даних другої системи повне дерево пошуку першої системи.

3.8 Метод "ключових операторів"

При необхідності зведення задачі пошуку у просторі станів, яка визначається трійкою (S, F, G), до сукупності більш простих задач, виникає задача виділення ланцюжка операторів, що переводять систему з початкового стану S до деякого проміжного стану g_i. В загальному випадку така задача є досить складною. Але іноді виділення окремих операторів, що є необхідними для розв'язання задачі (тобто операцій, які обов'язково доведеться виконати в процесі її розв'язування), буває досить простим. Наприклад, в задачі про ханойську вежу таким явно необхідним оператором є оператор "перекласти диск С на кілок 3". Такі оператори отримали назву "ключових". Один з засобів знаходження операторів, що можуть відігравати роль ключових полягає в обчисленні відмінностей для даної задачі (S, F, G). Відмінність - це частковий список причин, з яких елементи множини S не задовольняють ті умови, яким повинні задовольняти цільові стани (множина G). Якщо деякий елемент множини S знаходиться в G, то задача розв'язана і ніяких відмінностей не існує. Якщо ж множина цільових станів G визначається деякою групою умов, частину яких задовольняє елемент $s \in S$, то відмінністю може служити перелік тих умов, які S не задовольняє. Причому такі умови можуть бути розташовані відповідно їх важливості. В даному випадку кожній можливій відмінності ставиться у відповідність деякий оператор (чи множина операторів в просторі станів). Це і є ті оператори, які можуть бути визначені як ключові. При цьому відмінність ставиться у відповідність оператору тільки в тому випадку, якщо його застосування може усунути її.

Повернемося до задачі про мавпу та банани (див.сторінку 24). Тут нами було визначено чотири оператори:

		підійти (u)
f1:	(w,0,y,z)	----------> (u,0,y,z)
		пересунути (v)
f2:	(w,0,w,z)	----------> (v,0,v,z)
		залізти
f3:	(w,0,w,z)	----------> (w,1,w,z)
		схопити
f4:	(c,1,c,0)	----------> (c,1,c,1),

де w – місце розташування мавпи;
y - місце знаходження ящика;

с - точка, в якій висять банани.

Критерієм досягнення мети є виконання умови z = 1. Початковий стан задано списком (a, 0, в, 0). Початковий стан задачі має при цьому такий вигляд: **({a, 0, b, 0}, F_i, G).** Оскільки множина операторів F_i залишається в задачі незмінною, можна спростити її описання до такого вигляду: ({a, 0, b, 0}, G). Враховуючи вище сказане, процедуру зведення задачі до сукупності підзадач з використанням поняття ключових операторів можна описати таким чином.

Для початкової задачі відмінність від цільової полягає в першу чергу в тому, що останній елемент початкового списку не дорівнює 1. Ключовим оператором, що відповідає за усунення цієї відмінності є оператор f4 "схопити". (Тут ми припускаємо, що зв'язки між можливими відмінностями та їх ключовими операторами задані з самого початку).

Таким чином, використовуючи оператор f4 для редукції початкової задачі, приходимо до пари підзадач:

$$(\{a,0,b,0\},Gf4) \quad \text{і} \quad (\{f4(S1)\},G), \quad S1 \in Gf4.$$

де Gf4 - множина описів всіх станів, до яких можна застосувати оператор f4 (в нашому випадку це єдиний стан {c, 1, c, 0});

Sl - той стан в Gf4, застосування до якого оператора f4 призведе до розв'язання підзадачі (тобто до стану {c, 1, c, 1}).

Таким чином першим етапом редукції задачі буде такий (рисунок 3.11):

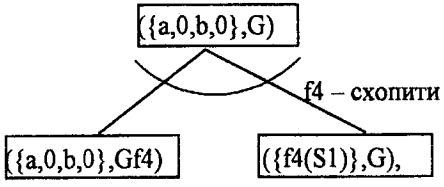


Рисунок 3.11 – Перший етап редукції задачі

Оскільки перш за все треба вирішити першу задачу з цієї пари, виявимо присутні в ній відмінності. Стан {a, 0, b, 0} не входить до Gf4 ({a, 0, b, 0} ≠ {c, 1, c, 0}) перш за все тому, що:

ящик не знаходиться в точці с; $b \neq c$

мавпа не знаходиться в точці с; $a \neq c$

мавпа не знаходиться на ящику. 0 (друга позиція списку атрибутів) ≠1.

Відповідно, можна визначити наступні ключові оператори: f2 - пересунути; f1 - підійти; f3 - залізти. Оскільки в дійсності треба застосувати лише один з цих операторів, альтернатива вибору буде представлена вершиною типу АБО. Розглянемо ту частину отриманого І/АБО графа, яка буде побудована у випадку вибору в якості ключового оператора f2.

Застосування цього оператора призведе до виникнення нової пари задач:

({a, 0, b, 0}), Gf2) і ({f2(S2)}, Gf4), S2 ∈ Gf2,

які показані на рисунку 3.12.

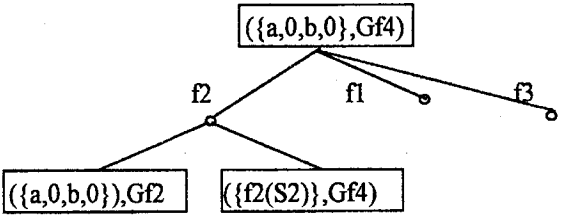


Рисунок 3.12 – Фрагмент І/АБО графа після застосування ключового оператора f2

Для першої з цих задач відмінність ({a, 0, b, 0}, {w, 0, w, z}) полягає в незбіганні першого та третього аргументів, тобто в тому, що мавпа не знаходиться в точці b. Така відмінність може бути усунена тільки використанням ключового оператора f1 - "підійти", який породжує дві підзадачі:

({a, 0, b, 0}, Gf1) і ({f1(S3) }, Gf2), S3 ∈ Gf1.

Перша підзадача означає, що від стану {a, 0, b, 0} треба перейти до такого стану, до якого можна застосувати оператор f1. Але ця задача є тривіальною, її відмінність дорівнює нулю, тому що стан (a, 0, b, 0) знаходиться в області визначення оператора f1. Тобто ця задача може бути розв'язана, результатом чого стане перехід системи до стану S3 = {b, 0, b, 0}, який є наслідком розв'язку першої підзадачі. Тобто, друга задача приймає вигляд: ({f2 (b, 0, b, 0)}, Gf2). Однак цей стан, в свою чергу, знаходиться в області визначення оператора f2, тобто і ця задача є тривіальною і може бути легко розв'язана, результатом чого стане стан Gf2∈S2={c, 0, c, 0}, отриманий внаслідок застосування оператора f2 до стану {b, 0, b, 0}. Таким чином по цій гілці І/АБО графа досягнено кінцевих вершин. Для доведення розв'язуваності всього графа треба довести розв'язність другої задачі І-вершини f2.

Обчислимо відмінність задачі ({f2 (S2)}, Gf4), яка визначається відмінністю станів {c, 0, c, 0} і {c, 1, c, 0} відповідно. Ця відмінність обумовлена різними значеннями в описі станів другого аргументу і означає, що мавпа не залізла на ящик. Таку відмінність повинен усувати оператор f3 - "залізти", який, в свою чергу, породжує два оператори:

({c, 0, c, 0}, Gf3) і ({f3(S4)}, Gf4) , S4 ∈ Gf3.

Перша задача, яка потребує переходу від стану {c, 0, c, 0} до стану, який належить області визначення оператора f3 є тривіальною, бо ця умова

виконується без будь-яких перетворень. Результатом цієї підзадачі є застосування оператора f3 до стану {c, 0, c, 0), що породжує стан f3(S4)=S1={c, 1, c, 0}. При цьому тривіальною стає і друга підзадача, бо отриманий стан належить до області визначення функції f4.

Таким чином вся ліва гілка І/АБО графа, що породжується, є розв'язною. Для повного доведення розв'язності початкової задачі залишилось довести розв'язність правої гілки графа. Тут проблем не виникає, оскільки стан f4(S1) = {c, 1, c, 1}, що отримується внаслідок застосування оператора f4 - "схопити" до ситуації S1 = {c, 1, c, 0} безпосередньо належить множині цільових станів, тобто підзадача є тривіальною, що свідчить про вирішення початкової задачі. Аналізуючи побудований граф не важко визначити послідовність операторів, яка вирішує початкову задачу; {підійти (b), пересунути (c), залізти, схопити}.

Інші дві гілки І/АБО графа, які розпочинаються після його першої АБО вершини будуються за тими ж принципами.

Гілка, що розпочинається з оператора f1 - "підійти".

Застосування цього оператора призведе до виникнення нової пари задач:

({a, 0, b, 0}), Gf1) і ({f1(S5)}, Gf4), S5 ∈ Gf1,

які показані на рисунку 3.13.

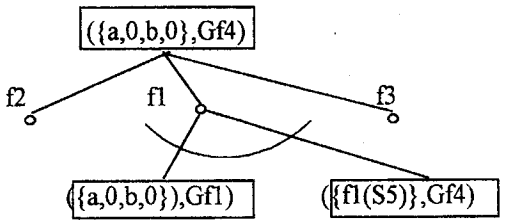


Рисунок 3.13 – Застосування оператора f1

Перша підзадача означає, що від стану {a, 0, b, 0} треба перейти до такого стану, до якого можна застосувати оператор f1. Ця задача є тривіальною, її відмінність дорівнює нулю, тому що стан (a, 0, b, 0) знаходиться в області визначення оператора f1. Тобто ця задача може бути вирішена, результатом чого стане перехід системи до стану S5 ={b, 0, b, 0}, який є наслідком вирішення першої підзадачі. Тобто, друга задача приймає вигляд: ({f1 (b, 0, b, 0)}, Gf4). Відмінність між {b, 0, b, 0} і {c, 1, c, 0} полягає в незбіганні першого і третього, та другого аргументів, тобто в тому, що мавпа з ящиком не знаходиться в точці c, і що мавпа не знаходиться на ящику. Такі відмінності можуть бути усунені використанням ключових операторів f2 - "пересунути", або f3 -"залізти", що породжує нову вершину типу АБО (рисунок 3.14).

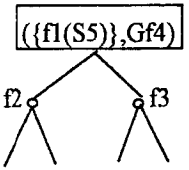


Рисунок 3.14 – Приклад породження нової вершини

Розглянемо ту частину отриманого І/АБО графа, яка буде побудована у випадку вибору в якості ключового оператора f2. Застосування цього оператора призведе до виникнення нової пари задач:

$$(\{b, 0, b, 0\}), Gf2) \text{ і } (\{f2\ (S6)\},Gf4),\ S6 \in Gf2$$

які показані на рисунку 3.15.

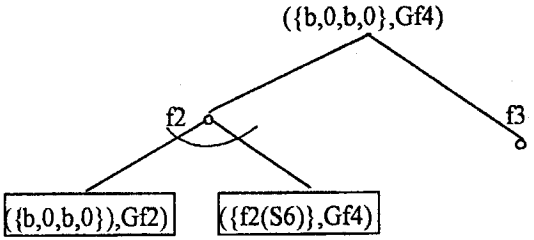


Рисунок 3.15 – Використання ключового оператора f2

Знов перша з підзадач є тривіальною, а друга тепер полягає у переводі стану Gf2 = f2 (S6) = {c, 0, c, 0} у стан, до якого можна застосувати оператор f4, тобто у стан {c, 1, c, 0}. Відмінність в даному випадку обумовлена значенням другого аргументу, а за її усунення відповідає оператор f3. Застосування цього оператора призведе до виникнення такої пари підзадач:

$$(\{c, 0, c, 0\}, Gf3) \text{ і } (\{f3(S7), Gf4),\ S7 \in Gf3.$$

Перша задача, яка потребує переходу від стану {c, 0, c, 0} до стану, який належить області визначення оператора f3 є тривіальною, бо ця умова виконується без будь-яких перетворень. Результатом цієї підзадачі є застосування оператора f3 до стану {c, 0, c, 0), що породжує стан f3 (S7) = S1 = {c, 1, c, 0}. При цьому тривіальною стає і друга підзадача, бо отриманий стан належить до області визначення оператора f4. З урахуванням показаної раніше тривіальності другої підзадачі першого рівня робимо висновок про розв'язання початкової задачі.

Гілка, що розпочинається з оператора f3 - "залізти".

Застосування цього оператора призведе до виникнення нової пари задач:

$$(\{a, 0, b, 0\}), Gf3) \text{ і } (\{f3(S8)\}, Gf4),\ S8 \in Gf3$$

які показані на рисунку 3.16.

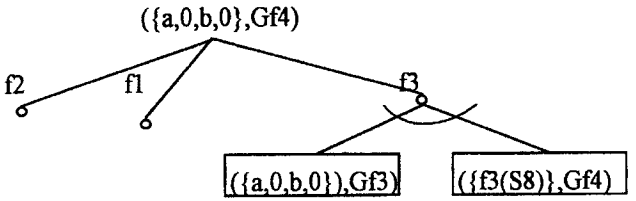


Рисунок 3.16 – Приклад застосування оператора f3

Для першої з цих задач відмінність стану {a, 0, b, 0} від стану {w, 0, w, 0} до якого можна застосувати оператор f4 полягає в незбіганні першого та третього аргументів, тобто в тому, що мавпа і ящик не знаходяться в одній і тій же точці. Така відмінність може бути усунена тільки використанням ключового оператора f1 - "підійти", який породжує дві підзадачі:

({a,0,b,0},Gf1) і ({f1(S9),Gf3), S9 ∈ Gf1.

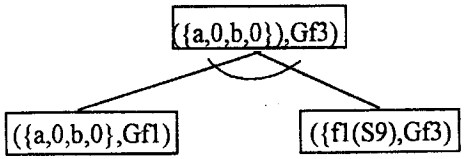


Рисунок 3.17 – Приклад використання ключового оператора f1

Перша задача є тривіальною, і результатом її вирішення є стан Gf1 = f1 (S9) = {b, 0, b, 0}. При цьому тривіальною стає і друга задача, оскільки стан {b, 0, b, 0} попадає до області визначення оператора f3. Але стан f3 (S8) = {b, 1, b, 0} не входить до області визначення оператора f4 - "схопити". Таким чином, права гілка породженого І/АБО графа є нерозв'язною.

Завдання для виконання

1. Використати метод "підняття вгору" для розв'язання ігрової ситуації для гри 8:

2	8	3
1	6	4
7	*	5

2. Використати метод "підняття вгору" для розв'язання ігрової ситуації для гри у "квадрати, що обертаються".

Умова задачі: Необхідно розташувати цифри у порядку зростання (зліва направо, зверху донизу). Кожен хід полягає в повороті будь-якого квадрата, створеного чотирма сусідніми.

7	1	6
6	4	3
8	9	2

3. Використати стратегію управління з повертанням для розвязання ігрової ситуації для гри 8:

2	8	3
1	6	4
7	*	5

Питання для самоперевірки

1. Визначити поняття формальної системи.
2. Визначити основні типи правил виведення в формальних системах.
3. Проаналізувати поняття системи продукцій.
4. Визначити поняття інтерпретації в формальних системах.
5. Проаналізувати представлення систем продукцій недетермінованими алгоритмами.
6. Проаналізувати узагальнений алгоритм управління системами продукцій.
7. Визначити основні компоненти системи продукцій та описати їх взаємодію в процесі функціонування.
8. Проаналізувати основні режими управління системами продукцій.
9. Проаналізувати стратегію "підйому вгору".
10. Даги оцінку стратегії управління із поверненням.
11. Визначити поняття зворотної та двосторонньої системи продукцій.
12. Проаналізувати комутативні системи продукцій та їх властивості.
13. Проаналізувати системи продукцій, які розкладаються.
14. Проаналізувати представлення задач, які можна звести до підзадач.
15. Визначити узагальнений алгоритм функціонування систем продукцій, які розкладаються.
16. Проаналізувати метод ключових операторів.

4.1 Ефективність стратегій управління системами продукцій

Важливою характеристикою обчислювального процесу є об'єм інформації, або "знань" про задачу. При повній відсутності інформації вибір робиться абсолютно довільно, без урахування будь-якої інформації про задачу. В разі наявності повної інформованості стратегія управління має достатньо знань, щоб кожен раз вибирати "найбільш вірне" правило.

В цілому ефективність стратегії управління залежить від того, в якому положенні між цими двома крайніми випадками знаходиться система управління. Обчислювальні витрати системи продукцій можна розподілити на дві категорії: витрати на застосування правил, та витрати на керування (тобто на вибір правил відповідно до наявної інформації про задачу). Повністю неінформована стратегія несе мінімальні витрати на стратегію управління, оскільки довільний вибір правил не потребує великих обчислень. Але при цьому значними є витрати на застосування правил, тому що для знаходження рішення треба перепробувати велику їх кількість. Введення в систему управління повної інформації про область задачі, яка розглядається, пов'язане з використанням стратегії управління великої вартості, що потребує великого об'єму пам'яті та обчислень для вибору одного з багатьох правил з бази знань. При цьому витрати на використання правил (загальна кількість застосованих правил) тут мінімальні. Таким чином загальна вартість обчислень у системі продукцій складається з вартості застосування правил та вартості стратегії управління:

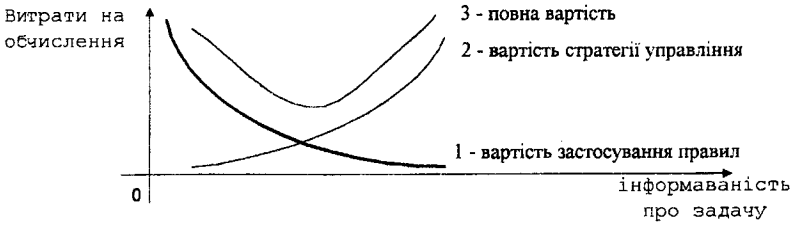


Рисунок 4.1 – Графік ефективності стратегій управління системами продукцій

4.2 Стратегії пошуку із поверненням

Стратегії пошуку з поверненням є досить ефективними для задач з невеликим обсягом пошукової роботи. Порівняно з стратегіями пошуку на графах вони є простішими у реалізації і потребують меншого об'єму пам'яті. Суть такої стратегії можна відобразити простою рекурсивною процедурою BACKTRACK, з єдиним аргументом DATA, який на початку роботи збігається з глобальною базою даних. За умов успішного завершення процедура видає на вихід (за допомогою команди return) перелік правил, послідовне застосування яких до початкової бази даних породжує базу, що задовольняє термінальній умові. Якщо процедура не знайде такого переліку правил, вона дає відповідь FAIL (невдача) [6].

Recursive procedure BACKTRACK

1. DATA ⇐ початковий стан глобальної бази даних;

2. **if** TERM (DATA), **return** NIL; TERM - предикат, що приймає значення "істина" для аргументів, які задовольняють термінальну умову системи продукцій. За умов успішного закінчення роботи на виході видається пустий перелік NIL.

3. **if** DEADEND (DATA), **return** FAIL; DEADEND - предикат, що приймає значення "істина" для аргументів, про які відомо, що вони не зустрічаються на шляху до розв'язку задачі. При цьому на виході видається значення FAIL.

4. RULES ← APPRULES(DATA); APPRULES - функція, що вибирає правила, які можна застосувати до її аргументу та впорядковує їх (або довільно, або відповідно до значення їх евристичної функції).

5. LOOP: **if** NULL(RULES), **return** FAIL; процедура фіксує невдачу, якщо відсутні правила, які можна застосувати

6. R ← FIRST(RULES); вибір "найкращого" з правил, які можна застосувати.

7. RULES ← TAIL(RULES); виключення з переліку правил, які можна застосувати, вибраного правила.

8. RDATA ← R(DATA); застосування правила R і породження нової бази даних.

9. PATH ← BACKTRACK(RDATA); BACKTRACK - рекурсивний виклик для нової бази даних.

10. **if** PATH = FAIL, **go** LOOP; якщо рекурсивний виклик невдалий, то спробувати застосувати інше правило.

11. **return** CONC(R, PATH); видача переліку правил, що розпочинаються з R та забезпечують успішне завершення.

Процедура забезпечує успішне завершення роботи тільки якщо породжується база даних, яка задовольняє термінальну умову (крок 2).

Невдале завершення можливе на кроках 3 та 5, але якщо воно трапиться під час рекурсивного виклику, то процедура повертається на більш високий рівень.

Процедура може зациклюватися, або без кінця породжувати нові бази даних. Обом цим випадкам можна запобігти, якщо ввести обмеження на глибину рекурсії і перелік баз даних, які вже породжувались в процесі пошуку, та перевіряти кожну нову базу даних на відповідність тим, що вже є у переліку.

На крокові 4 правила можна впорядковувати довільно, але більш ефективним є розташування на першому місці найбільш "вірного" правила, це забезпечить відсутність повернень.

Приведена процедура пошуку є загальною. Вона лише ілюструє процес управління і не є найбільш ефективною.

Запобігання зациклювань забезпечує алгоритм BACKTRACK(DATALIST) [6]:

1. DATA ← FIRST(DATALIST); DATALIST - перелік усіх баз даних на зворотному шляху до початкової бази. DATA - остання породжена база даних.

2. **if** MEMBER(DATA,TAIL(DATALIST)), **return** FAIL; невдача, якщо процедура будує базу даних, яка вже зустрічалась раніше.

3. **if** TERM(DATA), **return** NIL; TERM - предикат, що приймає значення "істина" для аргументів, які задовольняють термінальну умову системи продукцій. За умов успішного закінчення роботи на виході видається пустий перелік NIL.

4. **if** DEADEND(DATA), **return** FAIL; DEADEND - предикат, що приймає значення "істина" для аргументів, про які відомо, що вони не зустрічаються на шляху до рішення задачі. При цьому на виході видається значення FAIL.

5. **if** LENGTH(DATALIST) > BOUND, **return** FAIL; невдача, якщо кількість застосованих правил перевищила встановлену межу. BOUND - глобальна змінна, визначена до першого виклику цієї процедури.

6. RULES ← APPRULES(DATA); APPRULES - функція, що вибирає правила, які можна застосувати до її аргументу та впорядковує їх (або довільно, або відповідно до значення їх евристичної функції).

7. LOOP: **if** NULL(RULES), **return** FAIL; процедура фіксує невдачу, якщо відсутні правила, які можна застосувати

8. R ← FIRST(RULES); вибір "найкращого" з правил, які можна застосувати.

9. RULES ← TAIL(RULES); виключення з переліку правил, які можна застосувати, вибраного правила.

10. RDATA ← R(DATA); застосування правила R і породження нової бази даних.

11. RDATALIST ← CONC(RDATA,DATALIST); RDATA додається до переліку баз даних, що вже зустрічались на шляху пошуку рішення.

12. PATH ← BACKTRACK1(RDATALIST); BACKTRACK - рекурсивний виклик для нової бази даних.

13. **if** PATH = FAIL, **go** LOOP; якщо рекурсивний виклик невдалий, то спробувати застосувати інше правило.

14. **return** CONC(R, PATH); видача переліку правил, що розпочинається з R .

Необхідно звернути увагу на те, що цей алгоритм не зберігає у пам'яті всі бази даних, що зустрічалися раніше. Він запам'ятовує тільки ті бази, що знаходяться на поточному зворотному шляху до бази даних.

4.3 Загальна процедура пошуку на графах

На відміну від вищерозглянутої, стратегія пошуку на графах зберігає у пам'яті всі пробні шляхи з тим, щоб кожен з них міг залишатись кандидатом на подальше продовження. Така стратегія є більш гнучкою, але потребує більшого об'єму пам'яті. При цьому на практиці застосовують спеціальні методи стиснення інформації, які дозволяють зберігати в пам'яті тільки початкову базу даних, та записи змін, на основі яких можна швидко обчислити будь-яку необхідну базу даних.

Процес явного породження частки неявно заданого графа можна неформально визначити таким чином [6]:

1. Створити граф G, що включає лише початкову вершину S. Занести S до списку вершин з ім'ям OPEN.

2. Створити список з ім'ям CLOSE, який спочатку пустий.

3. LOOP: якщо OPEN пустий, то невдале закінчення роботи.

4. Вибрати першу вершину з OPEN, видалити її з цього списку, та розмістити у списку CLOSE. Помітити цю вершину через n.

5. Якщо n - цільова вершина - успіх, закінчення роботи з видачею розв'язку, яке отримують переглядом шляху в графові G уздовж покажчиків від n до S (покажчики визначаються на кроці 7).

6. Розкрити вершину n, яка породжує множину M її спадкоємців, що не є предками n. Розмістити в графові G ці елементи множини M як спадкоємців вершини n.

7. Ввести покажчики до n від тих елементів з M, які раніше не зустрічались в G (тобто не зустрічались ні в списку OPEN, ні в списку CLOSE). Додати ці елементи до OPEN. Для кожного елемента з M, які вже були у OPEN чи CLOSE, вирішити, чи є необхідність

покажчика на n. Для кожного елемента з M, який вже є у CLOSE, прийняти рішення відносно кожного з його нащадків у G, чи треба переорієнтувати його покажчик.

8. Перевпорядкувати список OPEN відповідно, чи то з деякою довільною схемою, чи то з евристичною залежністю.

9. Перейти до мітки LOOP.

Ця процедура має досить загальний характер і вміщує в себе велику різноманітність окремих алгоритмів пошуку на графах. Процедура породжує в явному вигляді граф G, який називається графом пошуку, та підмножину T графа G, яка називається деревом пошуку. Кожна вершина з G належить також і до T. Дерево пошуку має покажчики, які встановлюються на крокові 7. Кожна вершина (за винятком S), має в G покажчики, що спрямовані тільки до однієї з батьківських вершин в графі G, яка визначає її єдиного предка в T. Цей граф пошуку впроваджує часткове впорядкування, оскільки ні одна вершина з G не є своїм власним предком. Кожен можливий шлях до будь-якої вершини, відкритий цим алгоритмом, зберігається в явному вигляді у G, і лише один виділений шлях до будь-якої вершини визначається в дереві T. Список OPEN зберігає ті (кінцеві) вершини дерева пошуку, які ще не були вибрані для розкриття. Список CLOSE вміщує або такі кінцеві вершини, які не породжують спадкоємців у графі пошуку, або некінцеві вершини дерева пошуку.

Шлях розв'язання від початкової вершини до цільової можна відбудувати проглядаючи (в зворотному порядку) покажчики від цільової вершини до S. Процес закінчується невдало, якщо на дереві пошуку не залишається вершин, які ще не були обрані для розкриття. Невдача означає, що цільову вершину не можна досягнути з початкової.

Дивлячись на крок 7 можна помітити таку деталь. Якщо б неявно заданий граф, на якому відбувається пошук був деревом, то можна було б бути впевненим, що ніякий із спадкоємців, що породжується на крокові 6, не породжувався раніше, тому що кожна вершина дерева завжди є спадкоємцем лише однієї вершини.

Якщо ж цей граф не є деревом, то можливо, що деякі елементи з M вже породжувались раніше, тобто вони вже можуть знаходитися у списках OPEN чи CLOSE. Тому процедура GRAPHSEARCH перевіряє всі знов розкриті вершини на ідентичність тим вершинам, які були розкриті раніше.

Коли процес пошуку породжує вершину, що вже виникала раніше, він знаходить до неї новий шлях (можливо кращий) порівняно з тим, що вже зафіксовано у дереві пошуку. Бажано, щоб дерево пошуку зберігало той шлях, вартість якого є мінімальною (вартість шляху складається з суми вартостей всіх дуг, що належать цьому шляху). Якщо знов знайдений шлях дешевший за попередній, дерево пошуку перетворюється - батьківські функції затверджуються за останнім предком знов одержаної вершини.

Якщо у T було змінено предка вершини n, що знаходився у списку CLOSE, тобто знайдено шлях до n, який має меншу вартість, ніж попередній, то можлива зміна в порядку батьківства в підмножині T спадкоємців n у G.

Розглянемо приклад.

Припустимо, що процес пошуку породив граф та дерево пошуку, які показано на рисунку 4.2. Стрілки тут відображають покажчики, що визначають батьківські вершини в дереві пошуку. Чорні вершини знаходяться в списку CLOSE, білі - в списку OPEN в той момент, коли алгоритм вибирає для розкриття вершину 1. При розкритті вершини 1 породжується її єдиний спадкоємець - вершина 2. Але вершина 2, як і її предок на дереві пошуку - вершина 3, вже породжувалась у процесі пошуку і знаходиться у списку CLOSE зі своїми вершинами спадкоємцями 4 та 5. Однак, предком вершини 4 є вершина 6, оскільки через неї проходить шлях меншої вартості від S до вершини 4. Тепер знайдено новий шлях до вершини 2 через вершину 1, вартість якого нижча за вартість попереднього шляху через вершину 3. Тому предком вершини 2 у дереві пошуку стає замість вершини 3 вершина 1. Вартість шляхів до нащадків вершини 2 на дереві пошуку перераховується. Ця вартість менша за попередню, тому предком вершини 4 замість вершини 6 стає вершина 2. Скореговане дерево пошуку визначено покажчиками дуг у графі пошуку.

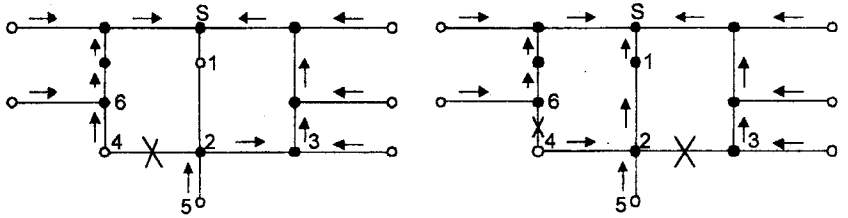


Рисунок 4.2 – Приклад графа і побудови дерева процесу пошуку

Алгоритм GRAPHSEARCH породжує одразу усіх спадкоємців вершин. Можна змінити його так, щоб в кожен момент розкривалася лише одна вершина. Але це зробить алгоритм більш складним для розуміння, тому в подальшому будемо користуватись тим варіантом алгоритму, що породжує усі вершини спадкоємці одночасно.

4.4 Неінформовані процедури пошуку на графах

Якщо для впорядкування вершин у списку OPEN на кроці 8 алгоритму GRAPHSEARCH не використовується ніяка евристична

то процедура пошуку називається неінформованою. В системах штучного інтелекту такі процедури як правило не застосовуються, але для порівняння розглянемо два основних типи: пошук у глибину та пошук у ширину.

Перший тип пошуку розташовує вершини у переліку OPEN в порядку зменшення їх глибини у дереві пошуку. Вершини, що розташовані на одній глибині, впорядковуються довільно. На першому місці в списку розташовуються найбільш глибокі вершини. Оскільки при цьому для розкриття завжди вибирається найбільш глибока вершина, він називається пошуком у глибину. Для попередження безмежного прямування процесу пошуку по безперспективному шляху задають обмеження на глибину пошуку. Таким чином процес пошуку заглиблюється вздовж одного шляху доки він не досягне встановленої межі, після чого розглядаються інші шляхи тієї ж чи меншої глибини, що відрізняються одним кроком, потім розглядаються шляхи, що відрізняються двома кроками і т.д.

Другий тип неінформованої стратегії пошуку розміщує вершини в списку OPEN в порядку зростання їх глибини в дереві пошуку. Він називається пошуком в ширину, тому що розкриття вершин у дереві здійснюється вздовж "рівня" однієї глибини. Пошук в ширину гарантує знаходження найкоротшого шляху до цільової вершини при умові, що вона існує.

4.5 Використання функцій оцінювання

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. Інформація, яка дозволяє сформулювати такі правила називається евристичною, а методи пошуку, що її використовують - методами евристичного пошуку. Суть цих методів полягає в тому, що вони дозволяють не розкривати більшість таких вершин графа пошуку, які не ведуть до цільової вершини. Така можливість з'являється за рахунок того, що на кожному крокові алгоритму пошуку на графі згідно деякої умови виконують перевпорядкування вершин в списку OPEN. Міра, яка в процесі перевпорядкування дозволяє здійснювати оцінки "перспективності" вершин називається функцією оцінювання.

Частіше за все знайдені евристики зменшують витрати на перебір за рахунок втрати гарантованої можливості знаходження шляху мінімальної вартості. Але, як правило, користувача цікавлять такі методи перебору, в яких мінімізується деяка комбінація з вартості шляху та вартості перебору усереднена по всіх задачах, що можуть йому зустрітись. Чим меншою є

така усереднена комбінаційна вартість методу перебору, тим більшу евристичну силу він має. На практиці обчислити евристичну силу алгоритму часто буває неможливо, і рішення про вибір конкретного алгоритму приймається на основі інтуїції експерта та на основі результатів експериментів з конкретними методами.

Розглянемо головні принципи використання евристичних функцій. Візьмемо деяку довільну функцію оцінювання f, яка приймає на вершинах n значення f(n). Пізніше ми припустимо, що її значення збігається з оцінкою вартості того з шляхів, що веде від початкової до цільової вершини через вершину n, вартість якого є мінімальною. Будемо використовувати алгоритм впорядкованого перебору, згідно з яким вершини в списку OPEN, що призначені для відкриття на кроковi 8 алгоритму GRAPHSEARCH, будемо розташовувати в порядку зростання відповідних їм значень функції f (тобто від меншого значення до більшого). У випадку збігу значень f, впорядкування виконується довільно, але цільовим вершинам завжди надається перевага.

Використаємо для гри 8 просту функцію оцінювання $f(n) = \tilde{g}(n)+W(n)$, де $\tilde{g}(n)$ - довжина шляху в дереві пошуку від початкової вершини до вершини n, а W(n) - кількість фішок, що знаходяться не на своєму місці в описанні ситуації, пов'язаній з вершиною n. За припущенням, на оптимальному шляху з більшою ймовірністю знаходиться та вершина, що має мінімальну оцінку. В цьому випадку використання функції оцінювання для гри 8 (початкова ситуація зображена в прикладі на рисунку 4.3), дасть такий результат, показаний на рисунку 4.3.

Використання функції оцінювання дозволило скоротити кількість розкритих вершин до 6 (замість 28-ми), при цьому побудовано було лише 14 вершин (замість 49-ти). Якщо використати правило вибору будь-якої вершини n із значенням $f(n) \leq f(n+1)$, то кількість побудованих вершин може зменшитися ще на 3 - 4.

Треба враховувати, що вибір функції оцінювання дуже сильно впливає на результат перебору. Вибір функції, яка не враховує дійсної перспективності деяких вершин може привести до вибору шляхів, що не є шляхами мінімальної вартості. З іншого боку, якщо функція переоцінює (завищує оцінку) перспективність деяких вершин, то буде розкрито занадто багато зайвих вершин, тому треба провести дослідження щодо використання функції оцінювання, яка забезпечить розкриття мінімальної кількості вершин і гарантує при цьому знаходження шляху мінімальної вартості.

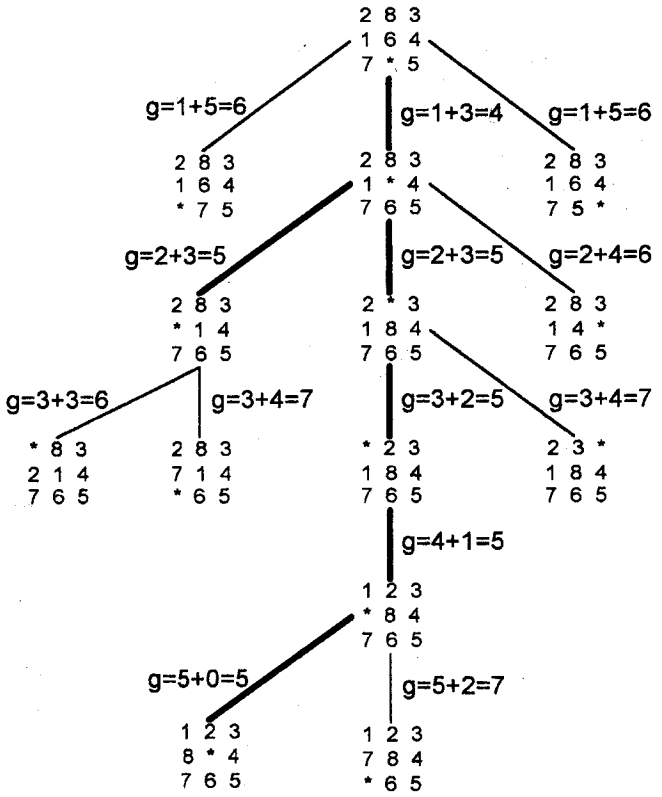


Рисунок 4.3 – Приклад використання функції оцінювання для гри 8

4.6 Алгоритм A*

Визначимо функцію оцінювання таким чином, щоб її значення f(n) для будь-якої вершини n оцінювало суму вартості шляхів мінімальної вартості (ШМВ) від початкової вершини s до вершини n і вартості аналогічного шляху від вершини n до цільової вершини. Це буде означати, що f(n) є оцінкою вартості ШМВ при умові, що він проходить через вершину n. В зв'язку з цим, вершина з списку OPEN, яка має мінімальне значення f(n) вважається такою, що розташована на ШМВ і з цієї причини вона повинна бути розкрита в першу чергу.

Введемо ряд позначень. Нехай функція $k(n_i, n_j)$ дає дійсну вартість ШМВ між двома довільними вершинами n_i, n_j (вона не визначена для тих

пар вершин, між якими не існує шляху). Якщо T = {t_i} множина цільових вершин, то вартість ШМВ від вершини n_i до мети можна визначити так:

$$h(n_i) = \min_{n_j \in T}(n_i, n_j)$$

Функція h буде не визначеною для тих вершин n, з яких не може бути досягнуто ні одну з цільових вершин. Можна побачити, що будь-який шлях від n_i до t_i зі значенням h(n) буде оптимальним.

Введемо також нову функцію g, що буде визначати вартість оптимального шляху від початкової вершини s до деякої довільної вершини n: g(n) = k(s, n), для усіх n, які можна досягнути з s.

Тепер можемо записати функцію f таким чином, щоб її значення f(n) для будь-якої вершини n складало:

$$f(n) = g(n) + h(n),$$

тобто визначало суму оптимального шляху при умові, що він проходить через вершину n (зауважимо, що f(s) = h(s) представляє дійсну вартість оптимального шляху від початкової вершини s до мети, без будь-яких обмежень).

Визначимо тепер функцію оцінювання $\tilde{f}$, яка буде оцінкою функції f. Будемо вважати, що необхідна функція оцінювання задається виразом: $\tilde{f}(n) = \tilde{g}(n) + \tilde{h}(n)$, де $\tilde{g}(n)$ - оцінка для g, $\tilde{h}$ - оцінка для h.

Зрозуміло, що в якості $\tilde{g}(n)$ можна обрати вартість ШМВ в дереві перебору від s до n, що знайдений алгоритмом на даний момент часу, і може бути отриманий додаванням вартостей дуг, які розташовані на тих шляхах від s до n, що позначені покажчиками. З визначення випливає, що $\tilde{g}(n) \geq g(n)$, тобто значення $\tilde{g}(n)$ для деяких вершин можуть зменшуватись при зміні дерева пошуку на крокові 7 алгоритму GRAPHSEARCH.

При визначенні оцінки $\tilde{h}(n)$ для h(n) будемо виходити з будь-якої евристичної інформації, пов'язаної з самою задачею. Назвемо $\tilde{h}$ евристичною функцією.

Алгоритм впорядкованого перебору, який використовує функцію оцінювання $\tilde{f}$ називають алгоритмом A*. Помітимо, що для $\tilde{h} = 0$ і $\tilde{g} = d$, алгоритм A* співпадає з алгоритмом пошуку в ширину. Раніше стверджувалося без доведення, що алгоритм пошуку в ширину гарантує знаходження шляху мінімальної вартості. Покажемо тепер. що якщо $\tilde{h}$ є нижчою межею для h (тобто виключається переоцінка вартості переходу від n до цільової вершини, або завжди виконується умова: $\tilde{h}(n) \leq h(n)$), то алгоритм A* також завжди знайде оптимальний шлях до мети. Врахуємо, що оскільки значення $\tilde{h} = 0$ безумовно є нижчою межею для h, то той факт, що алгоритм пошуку в ширину дозволяє знаходити оптимальні шляхи, буде випливати як окремий випадок цього більш загального результату для A*.

4.7 Спроможність алгоритму A*

Будемо стверджувати, що алгоритм пошуку спроможний, якщо для будь-якого графа він закінчує роботу знаходженням оптимального шляху від s до цільової вершини, при умові, що такий шлях існує. Розглянемо неформальне доведення спроможності алгоритму A*.

1. Для підтвердження спроможності алгоритму A* покажемо спочатку, що він завжди завершує роботу, якщо цільову вершину можна досягти з початкової вершини. Ми пам'ятаємо, що алгоритм GRAPHSEARCH зупиняється (якщо це взагалі трапиться) на кроках 3 або 5. Помітимо, що при повторенні кожного циклу алгоритму з списку OPEN видаляється одна вершина і тільки кінцева кількість спадкоємців додається в цей список. Для кінцевих графів список рано чи пізно буде вичерпано. Тобто, якщо алгоритм і не знайде цільову вершину (тобто не зупиниться на крокові 5), то він зупиниться на крокові 3, вичерпавши кінець кінцем список OPEN.

Висновок 1: процедура GRAPHSEARCH завжди закінчує роботу на кінцевих графах.

2. Покажемо тепер, що якщо шлях від s до цільової вершини існує, то алгоритм A* завершить свою роботу навіть на нескінченому графові. Для цього припустимо зворотне, що алгоритм A* ніколи не зупиниться. Це може статись, якщо нові вершини постійно будуть додаватись до списку OPEN. Але, можна показати, що в цьому випадку навіть мінімальні значення f в списку OPEN повинні зростати до занадто великих величин.

Нехай d(n) - довжина самого короткого шляху в неявному графові, що розглядається, від s до будь-якої вершини n на дереві пошуку, що породжується алгоритмом A*. Оскільки вартість кожної дуги є хоча б яким малим, але додатним числом e, то буде виконуватись співвідношення $g(n) \geq d(n) \cdot e$. (Нагадаємо, що g(n) - вартість оптимального шляху від s до n, а $\tilde{g}(n)$ - вартість шляху від s до n на дереві пошуку.) Зрозуміло, що при цьому має місце співвідношення $\tilde{g}(n) \geq g(n)$. Тобто і $\tilde{g}(n) \geq d(n) \cdot e$. Якщо $\tilde{h}(n) \geq 0$, (що будемо припускати і в подальшому), то $\tilde{f}(n) \geq \tilde{g}(n)$, тобто і $\tilde{f}(n) \geq d(n) \cdot e$. Таким чином, для кожної вершини з списку OPEN її функція $\tilde{f}$ дорівнює, як мінімум, $d(n) \cdot e$. І хоча алгоритм A* вибирає для розкриття ту вершину з списку OPEN для якої значення $\tilde{f}$ мінімальне, ця вершина, тим не менш, буде кінець кінцем мати яке завгодно велике значення d, а відповідно, і значення $\tilde{f}$, якщо алгоритм A* не зупиниться.

Покажемо тепер, що до моменту зупинки алгоритму A* в списку OPEN завжди є деяка вершина n, для якої $\tilde{f}(n) \leq f(s)$. Нехай впорядкована послідовність $s = (n_0, n_1, \ldots n_k)$ буде оптимальним шляхом від s до цільової

вершини n_k. Тоді в будь-який момент до завершення роботи алгоритму A* будемо вважати, що n′ - перша вершина з цієї послідовності, що знаходиться в списку OPEN (там має бути хоча б одна така вершина, тому що останньою закривається вершина n_k, але при цьому алгоритм завершує свою роботу). Згідно з визначенням $\tilde{f}$ для A* маємо: $\tilde{f}(n') = \tilde{g}(n') + \tilde{h}(n')$. Відомо, що алгоритм A* вже знайшов оптимальний шлях до вершини n', оскільки n′ розташована на оптимальному шляху до мети і всі її попередники на цьому шляху закриті. Отже $\tilde{g}(n') = g(n)$ і $\tilde{f}(n') = g(n') + \tilde{h}(n')$. З урахуванням припущення, що $\tilde{h}(n') \le h(n')$, можемо записати:

$$\tilde{f}(n') \le g(n') + h(n') = f(n').$$

Але значення f для будь-якої вершини на оптимальному шляху дорівнює просто f(s), тобто мінімальній вартості, тому $\tilde{f}(n') \le f(s)$, що і потрібно було довести.

Висновок 2: В будь-який момент часу до завершення роботи алгоритму A* в списку OPEN існує вершина n′ така, що вона знаходиться на оптимальному шляху від s до цільової вершини, і при цьому виконується співвідношення $\tilde{f}(n') \le f(s)$

Порівнявши отриманий результат з попереднім твердженням про те, що навіть самі малі значення f для вершин в списку OPEN у випадку безупинного алгоритму A* стають необмеженими зверху, бачимо, що алгоритм A* повинен завершити роботу навіть для нескінченних графів.

Висновок 3: Якщо шлях від s до цільової вершини існує, то алгоритм A* завершує роботу.

Наслідком цього результату є те, що будь-яка вершина n з списку OPEN, для якої $\tilde{f}(n) < f(s)$, раніше чи пізніше вибрана алгоритмом A* для розкриття.

3. Відповідно до висновку 2, якщо існує шлях від s до цільової вершини, то в списку OPEN і на оптимальному шляху завжди є деяка вершина, тобто алгоритм не має змоги зупинитись на кроковi 3 алгоритму, не вичерпавши список OPEN. Покажемо тепер, що він закінчує свою роботу тільки при виявленні оптимального шляху до цільової вершини. Уявимо собі, що алгоритм A* зупинився, знайшовши деяку цільову вершину t, але не знайшовши при цьому оптимального шляху, тобто $\tilde{f}(t) = \tilde{g}(t) > f(s)$. Але відповідно до висновку 2 безпосередньо перед завершенням роботи алгоритму існує вершина n′ в списку OPEN і на оптимальному шляху така, що $\tilde{f}(n') \le f(s) < \tilde{f}(t)$. Тобто, на цьому етапі алгоритм A* обов'язково відібрав би для розкриття n′, а не t, що не відповідає зробленому нами припущенню про завершення роботи алгоритму A*. Отже, в кінцевому підсумку отримаємо:

Висновок 4: Алгоритм A* спроможний, тобто якщо існує шлях від s до цільової вершини, то алгоритм A* завершить роботу тільки знайшовши оптимальний шлях.

Помітимо, що кожна вершина, яку вибирає для розкриття алгоритм A* має цікаву властивість, що є безпосереднім наслідком з висновку 2: значення функції $\tilde{f}$ для неї ніколи не перевищує вартості f(s) оптимального шляху. Для доведення цього позначимо як n довільну вершину, що вибрана алгоритмом A* для розкриття. Якщо n - цільова вершина, то відповідно до висновку 4 $\tilde{f}(n) = f(s)$, тому вважаємо, що n нецільова вершина, оскільки алгоритм A* вибирає n до завершення роботи, то (відповідно до висновку 2) в списку OPEN є деяка вершина n′, що розташована на оптимальному шляху від s до мети. Для цієї вершини виконується співвідношення $\tilde{f}(n') < f(s)$. Якщо $n = n'$, то отримаємо наш результат. Інакше, відомо, що алгоритм A* вибере для розкриття не n′, а n, отже це має бути випадок, коли $\tilde{f}(n) \le \tilde{f}(n') \le f(s)$.

Висновок 5: Для будь-якої вершини n, вибір якої здійснив алгоритм A* для розкриття, виконується співвідношення $\tilde{f}(n) < f(s)$

4.8 Оптимальність алгоритму A*

Точність евристичної функції $\tilde{h}$ залежить від того, наскільки велике евристичне знання про область задачі закладено в неї. Зрозуміло, що використання значення $\tilde{h}(n) = 0$ свідчить про повну відсутність будь-якої евристичної інформації про задачу, незважаючи на те, що така оцінка є нижньою межею h(n) і, таким чином, приведе до оптимального алгоритму.

Порівняємо два варіанти алгоритму A*: A1* та A2*, які використовують відповідні функції оцінювання:

$$\tilde{f}_1(n) = \tilde{g}_1(n) + \tilde{h}_1(n) \text{ і } \tilde{f}_2(n) = \tilde{g}_2(n) + \tilde{h}_2(n),$$

де $\tilde{h}_1$ і $\tilde{h}_2$ - нижні межі для h_1 і h_2, відповідно. Будемо вважати, що алгоритм A2* є більш інформованим ніж алгоритм A1*, якщо для всіх нецільових вершин n виконується умова $\tilde{h}_2(n) > \tilde{h}_1(n)$. Таке припущення здається інтуїтивно розумним, оскільки $\tilde{h}$ обмежено зверху h, що забезпечує спроможність алгоритму, і можна чекати, що використання більших значень $\tilde{h}$ (і відповідно більш наближених до значень h), потребує більш точної евристичної інформації.

Наприклад, для гри 8 розумно припустити, що алгоритм A* з $\tilde{h}(n) = W(n)$ є більш інформованим, ніж пошук у ширину, для якого $\tilde{h} = 0$ (відмітимо, що W(n) є нижньою межею кількості кроків, які залишилися до досягнення мети).

Інтуїтивно можна очікувати, що більш інформованому алгоритму для знаходження шляху мінімальної вартості знадобиться розкрити меншу кількість вершин. Це ще не означає, що при цьому він буде більш ефективним, тому що більш складні обчислення в більш інформованому алгоритмі можуть знизити його ефективність порівняно з менш інформованими алгоритмами. Тим не менш, кількість розкритих вершин є одним з важливих факторів, що визначають ефективність алгоритмів, який до того ж дозволяє виконувати прості порівняння.

Припустимо, що обидва алгоритми A1* і A2* при пошуку на явно заданому графі, що має шлях від вершини s до цільової вершини, завершать роботу знайшовши оптимальний шлях. Покажемо, що на момент завершення роботи, всяка вершина n в G, яку розкрив алгоритм A2*, обов'язково буде розкрита і алгоритмом A1*. Тобто, менш інформований алгоритм A1* завжди розкриває у будь-якому випадку не меншу кількість вершин, ніж більш інформований алгоритм A2*.

Доведемо цей результат з використанням індукції по глибині вершини на дереві пошуку після закінчення роботи алгоритму A2*. Спочатку покажемо, що якщо алгоритм A2* розкриває вершину n, що має нульову глибину в дереві пошуку, то це зробить і алгоритм A1*. Але в даному випадку n = s. Якщо s є цільовою вершиною, то ні один з алгоритмів не розкриває жодної вершини. Якщо s не є цільовою вершиною, то її розкриють обидва алгоритми. Продовжуючи доведення за індукцією припустимо (гіпотеза індукції), що алгоритм A1* розкриває всі ті вершини, які розкриває алгоритм A2* і які мають на дереві пошуку алгоритму A2* глибину k чи меншу. Тоді треба довести, що будь-яка вершина n, що розкрита алгоритмом A2* і має глибину k буде також розкрита і алгоритмом A1*. Згідно з гіпотезою індукції будь-який нащадок n в дереві пошуку алгоритму A2* буде розкритий також алгоритмом A1*. Отже вершина n знаходиться на дереві пошуку алгоритму A1* і існує шлях від s до n в цьому дереві пошуку , причому вартість шляху не перевищує вартість шляху від s до n в дереві пошуку алгоритму A2*, тобто $\tilde{g}_1(n) \le \tilde{g}_2(n)$.

Припустимо протилежне тому, що ми вирішили довести, тобто що алгоритм A1* не розкриває вершину n, яку розкрив алгоритм A2*. Після завершення роботи алгоритму A1* вершина n повинна знаходитись в такому випадку в його списку OPEN, оскільки алгоритм A1* розкрив її батьківську вершину. Оскльки алгоритм A1* завершив роботу, знайшовши шлях мінімальної вартості і не розкривши при цьому вершину n, то повинно виконуватися співвідношення $\tilde{f}(n) \ge f(s)$, отже

$$\tilde{g}_1(n) + \tilde{h}_1(n) \ge f(s).$$

Оскільки було показано, що $\tilde{g}_1(n) \le \tilde{g}_2(n)$, то $\tilde{h}_1(n) \ge f(s) - \tilde{g}_2(n)$

Але якщо згідно з висновком 5 алгоритм A2* розкрив вершину n_i, то отримаємо:

$\tilde{f}_2(n) \le f(s)$, або $\tilde{g}_2(n)+\tilde{h}_2(n) \le f(s)$, або $\tilde{h}_2(n) \le f(s) - \tilde{g}_2(n)$.

Порівняння нерівностей отриманих для $\tilde{h}_2(n)$ і для $\tilde{h}_1(n)$ показує, що в будь-якому випадку в вершині n $\tilde{h}_1$ повинно дорівнювати $\tilde{h}_2$, однак це суперечить припущенню, що алгоритм A2* є більш інформованим ніж алгоритм A1*. Таким чином приходимо до висновку:

Висновок 6: Якщо алгоритми A1* і A2* є двома варіантами алгоритму A* такими, що алгоритм A2* є більш інформованим ніж алгоритм A1*, то до закінчення процесів пошуку на будь-якому графі, що має шлях від вершини s до цільової вершини, кожна вершина, яка була розкрита алгоритмом A2*, буде розкрита і алгоритмом A1*. Тобто алгоритм A1* розкриває не менше вершин ніж алгоритм A2*.

4.9 Монотонні обмеження

Роздивляючись процедуру GRAPHSEARCH ми відзначали, що при розкритті деякої вершини, її спадкоємці можуть вже знаходитись в списках OPEN або CLOSE. Необхідне при цьому коректування дерева пошуку, а також сама процедура перевірки всіх вершин двох списків при розкритті кожної нової вершини, потребують великих обчислювальних витрат.

Але, виявляється, що існують достатньо слабкі обмеження на значення емпіричної функції $\tilde{h}$, виконання яких дозволяє алгоритму A* знаходити оптимальний шлях до вершини безпосередньо в процесі вибору її для розкриття. Тобто, задоволення цих обмежень виключає необхідність виконувати в алгоритмі A* перевірку, 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.

Кажуть, що евристична функція $\tilde{h}$ задовольняє монотонному обмеженню, якщо для будь-яких вершин n_i, n_j, таких що n_j є спадкоємцем n_i, виконується умова:

$$\tilde{h}(n_i) - \tilde{h}(n_j) \le k(n_i,n_j), \text{ причому } \tilde{h}(t)=0.$$

Ця нерівність встановлює, що різниця між оцінками вартості шляхів від будь-якої пари вершин n_i, n_j до цільової вершини, повинна бути нижньою межею вартості оптимального шляху від n_i до n_j (тобто оцінка оптимальної вартості шляху від n_i до мети не повинна перевищувати суму вартості дуги від n_i до n_j, і оцінки оптимальної вартості шляху від n_j до мети).

На прикладі гри 8 легко перевірити, що $\tilde{h}(n) = W(n)$ задовольняє монотонному обмеженню. Якщо ж функція h якимось чином змінюється в ході процесу пошуку, то монотонне обмеження може і не виконуватись.

Покажемо, що при такому обмеженні алгоритм A* відкриваючи вершину, дійсно знаходить оптимальний шлях до неї. Нехай n - будь-яка з вершин, яку алгоритм A* вибрав для розкриття. Якщо n = s, то знаходження оптимального шляху є тривіальним, тому вважаємо що n не збігається з s. Нехай послідовність $P = (s = n_0, n_1, n_2, \ldots n_k = n)$ є оптимальним шляхом від і до n. Нехай вершина n_l - остання з цієї послідовності, яка знаходиться в списку CLOSE в той час, коли A* вибирає n для розкриття (відзначимо, що s вже знаходиться у списку CLOSE, а n_k - не знаходиться, тому що безпосередньо вона вибирається для розкриття). Отже, вершина $n_l + 1$ з послідовності P знаходиться в списку OPEN в той час, як A* вибирає вершину n.

Використовуючи монотонне обмеження знаходимо:

$$g(n_i) + \tilde{h}(n_i) \le g(n_i) + k(n_i, n_{i+1}) + \tilde{h}(n_{i+1}).$$

Оскільки n_i і n_{i+1} знаходяться на оптимальному шляху, то

$$g(n_{i+1}) = g(n_i) + k(n_i, n_{i+1}) \text{ і } \{g(n_i) + \tilde{h}(n_i)\} \le \{g(n_{i+1}) + \tilde{h}(n_{i+1})\}.$$

В наслідок транзитивності отримуємо:

$$g(n_{i+1}) + \tilde{h}(n_{i+1}) \le g(n_k) + \tilde{h}(n_k), \text{ або } \tilde{f}(n_{i+1}) \le g(n) + \tilde{h}(n).$$

Отож, в той час, як алгоритм A* обирає вершину n, віддаючи їй перевагу перед вершиною n_{i+1}, повинна виконуватись умова $\tilde{g}(n) \le g(n)$, інакше $\tilde{f}(n)$ була б більшою за $\tilde{f}(n_{i+1})$. Оскільки $\tilde{g}(m) \ge g(m)$ для будь-яких вершин m на дереві пошуку, отримуємо:

Висновок 7. Якщо задовольняється монотонне обмеження, то алгоритм A* автоматично виявляє оптимальний шлях до будь-якої вершини, яку він обирає для розкриття. Тобто, якщо алгоритм A* обирає n для розкриття і якщо задовольняється монотонне обмеження, то має місце рівність $\tilde{g}(n) = g(n)$.

Іншим цікавим результатом, що пов'язаний із монотонним обмеженням є той, що значення функції $\tilde{f}$ на послідовності вершин, які розкриває алгоритм A*, створюють послідовність, що не зменшується. Нехай n_2 розкривається одразу після n_1. Якщо n_2 знаходиться в списку OPEN в той час, коли розкривається n_1, отримуємо (тривіально), що $\tilde{f}(n_1) \le \tilde{f}(n_2)$. Нехай n_2 не знаходиться в OPEN, коли розкривається n_1. (вершина n_2 не знаходиться і в CLOSE, бо ми припускаємо, що вона ще не розкрита). Тоді, якщо n_2 розкривається одразу після n_1, то вона повинна додаватися до списку *OPEN в процесі розкриття вершини* n_1. *В зв'язку з цим* n_2 є спадкоємцем n_1. При цьому в момент вибору n_2 для розкриття, маємо:

$$\tilde{f}(n_2) = \tilde{g}(n_2)+\tilde{h}(n_2) = g(n_2)+\tilde{h}(n_2) = g(n_1)+k(n_1,n_2)+\tilde{h}(n_2)$$
$$=\tilde{g}(n_1)+k(n_1,n_2)+\tilde{h}(n_2).$$

Оскільки монотонне обмеження означає, що $k(n_1, n_2)+\tilde{h}(n_2) \geq \tilde{h}(n_1)$, отримуємо:

$$\tilde{f}(n_2) \geq \tilde{g}(n_1)+\tilde{h}(n_1) = \tilde{f}(n_1).$$

Оскільки цей факт має місце для будь-якої пари сусідніх вершин в послідовності вершин, що розкриває алгоритм A*, отримуємо:

Висновок 8. Якщо задовольняється монотонне обмеження, то значення $\tilde{f}$ для послідовності вершин, що розкриті алгоритмом A*, не зменшується.

4.10. Евристична сила функцій оцінювання

При визначенні евристичної сили алгоритму впорядкованого пошуку, головну роль відіграє вибір функції $\tilde{h}$.

Вибір $\tilde{h} = 0$ гарантує спроможність, але веде до сліпого перебору. Вибір в якості $\tilde{h}$ найбільшої з можливих нижніх меж для h забезпечує розкриття мінімальної кількості вершин, зберігає спроможність. Але досить часто евристичну силу алгоритму можна підвищити за рахунок відмови від спроможності, шляхом використання як $\tilde{h}$ деякої функції, яка не є нижньою межею для h. Така додаткова евристична сила дозволяє вирішувати набагато більш складні задачі. Наприклад, для гри 8 функція $\tilde{h}$ = W(n) (де W(n) - кількість фішок, які знаходяться не на своїх місцях), є нижньою межею для h, але така функція оцінювання не забезпечує якісної оцінки труднощів даного розташування фішок (у розумінні кількості кроків, що відділяють дану позицію від мети). Кращу оцінку надає функція $\tilde{h}(n) = P(n)$, де P(n) - сума відстаней кожної фішки від "свого" місця. Але і ця функція є досить грубою, тому що не враховує належним чином труднощі обміну місцями двох сусідніх фішок.

Достатньо якісною оцінкою для гри 8 є така: $\tilde{h}(n) = P(n)+3S(n)$, де S(n) - кількість очок, яка враховує порядок розташування фішок. Для її обчислення треба послідовно проглянути всі нецентральні фішки і за кожну фішку, за якою розташована не та фішка, що має бути в цільовій ситуації, додати два бали, в іншому випадку - 0 балів. За фішку, яка знаходиться на центральному полі, надається 1 бал. Нагадаємо, що така функція $\tilde{h}$ не надає нижньої межі для h, але успішно використовується для знаходження рішень у набагато складніших ситуаціях ніж ті, що були розглянуті нами у прикладах. Наприклад, для початкової ситуації

використання функції забезпечує вирішення задачі за 18 кроків, кількість розкритих вершин - 21, кількість побудованих вершин - 44.

Другим фактором, що визначає евристичну силу алгоритму є обсяг зусиль по обчисленню значень $\tilde{h}$. Найкращою була б функція $\tilde{h}$ = h, яка забезпечила б абсолютний мінімум розкритих вершин. Але іноді набагато легше буває обчислити деяку функцію $\tilde{h}$, що відрізняється від нижньої межі h. При цьому евристична сила може зрости з двох причин: зниження загальної кількості вершин, що розкриваються (за рахунок відмови від спроможності) та зменшення обсягу обчислень.

Іноді евристичну силу $\tilde{h}$ можна підвищити множенням її на деяку константу більшу за одиницю. Якщо цей множник буде достатньо великим, отримаємо ситуацію аналогічну ситуації $\tilde{g}$(n) = 0. Такий вибір надасть безумовно недосконалий, але здатний задовільно працювати, алгоритм.

Таким чином на евристичну силу алгоритму впорядкованого пошуку впливають три важливих фактори: 1) вартість шляху; 2) кількість вершин, які розкриваються в процесі пошуку шляху; 3) обсяг обчислень для підрахунків значень $\tilde{h}$. Евристична сила алгоритму максимізується пошуком відповідного компромісу.

Існують випадки, коли необхідно знайти будь-який шлях до цільової вершини, при цьому вартість шляху значення не має (хоча нас турбує трудомісткість пошуку, необхідного для знаходження шляху). При цьому можна вважати, що значення $\tilde{g}$ можуть зовсім не враховуватися, оскільки на будь-якому крокові перебору нас не турбують вартості вже побудованих шляхів. Нас цікавлять лише ті витрати, що знадобляться для знаходження цільової вершини. Ці значення можливо і залежать від значень $\tilde{h}$ для розкритих вершин, але свідомо не залежать від значень $\tilde{g}$ для цих розкритих вершин. Тобто, в таких випадках можна використовувати як функцію оцінювання f = h. Але для гарантування того, що хоч який-небудь шлях буде кінець -кінцем знайдений, функція $\tilde{g}$ повинна бути включена до f навіть у випадках, коли нас не цікавить знаходження шляху з мінімальною вартістю. Такою впевненістю треба заручатись завжди, коли $\tilde{h}$ є недостатньо якісною оцінкою для h, оскільки може трапитись таке, що в процесі перебору весь час будуть розкриватися помилкові вершини, а цільової вершини так ніколи й не буде досягнуто. Включення в функцію оцінювання компоненти g додає в процес пошуку складову пошуку в ширину, завдяки чому гарантується, що ніяка частка неявно заданого графу не буде постійно залишатися недослідженою.

Відносну вагу функцій g та h можна змінювати, використовуючи оцінку $\tilde{f} = \tilde{g} + w\tilde{h}$, де w - додатне число. Дуже велике значення w надає евристичній компоненті, дуже маленьке

Дослідження показують, що ефективність пошуку часто зростає при зміні w в зворотній пропорції відносно глибини вершини на дереві пошуку. При невеликих глибинах пошук направляється в основному евристичною компонентою, при великих - йому все більше надають характеру пошуку в ширину, щоб гарантувати, що будь-який шлях буде кінець-кінцем знайдений.

4.11 Інші евристики, що знайшли широке застосування

1) Двонаправлений пошук. Перебор одночасно здійснюється з початкової та цільової множини вершин і завершується при зустрічі двох фронтів. Він розкриває значно меншу кількість вершин, але якщо евристичні функції будуть хоча б незначно не точними, два фронти можуть розійтись не перетнувшись. В такому випадку двонаправлений пошук може розкрити вдвічі більше вершин ніж однонаправлений.

2) Обмеження кількості дочірніх вершин. Використовує "інформований" оператор, який не породжує занадто багато зайвих вершин. Один з засобів скорочення перебору полягає в тому, щоб одразу після розкриття вершини відкинути майже усі дочірні вершини, залишивши лише декілька з них, що мають найменші значення f. Щоправда, відкинуті вершини можуть бути розташовані на значно кращих
шляхах (а то і на єдино можливому шляху), так що придатність цього підходу в конкретних випадках може показати тільки експеримент.

3). Перебор етапами . Іноді графи бувають такими великими, що пам'ять вичерпується раніше, ніж буде знайдено задовільний шлях. В таких випадках частину гілок побудованого на даний момент графу відсікають, звільняючи таким чином шлях до подальшого перебору. Такий процес здійснюється етапами, що розподіляться операціями відсічення дерева, необхідними для звільнення пам'яті. Наприкінці кожного етапу зберігається лише деяка підмножина розкритих вершин, наприклад, з найменшими

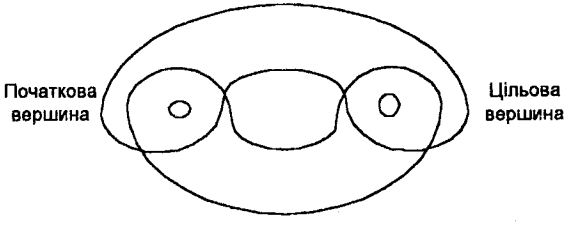


Рисунок 4.4 – Можливі шляхи розкриття вершин при двонаправленому пошукові

значеннями f. Найкращі шляхи до цих вершин зберігаються, а інша частка дерева відкидається. Цей процес продовжується доки не буде знайдено цільову вершину, чи не будуть вичерпані всі ресурси.

4). Поетапна побудова дочірніх вершин . Існують випадки, коли застосування всіх дозволених операторів до кожної вершини є занадто марнотратним. В цьому випадку можна використовувати більш інформований оператор, який виділяв би декілька найбільш перспективних операторів і будував би тільки ті наступні вершини, які виникають внаслідок їх застосування. Однак, треба залишати можливість побудувати в подальшому і інші дочірні вершини. При такому підході в алгоритм впорядкованого перебору треба внести відповідні зміни.

4.12 Критерії якості роботи алгоритмів перебору

Евристична сила методу перебору в значній мірі залежить від характеру задачі і визначення її є скоріше питанням досвіду, ніж обчислень. Однак, існують деякі критерії, корисні при порівнянні різних методів перебору.

а) Цілеспрямованість (направленість). Направленість пошуку P - це ступінь того, наскільки пошук є сфокусованим у напрямку мети, а не блукаючим по безперспективних напрямках:

$$P = L / T,$$

де L - довжина знайденого шляху до мети; T - загальна кількість вершин, побудованих в процесі перебору (за винятком початкової). Направленість показує наскільки дерево перебору витягнуте, а не кущоподібне. Вона зворотно-пропорційно залежить від довжини знайденого шляху (T як правило зростає швидше за L).

б) Показник ефективності розгалуження B. Він є більш незалежним від довжини оптимального шляху. Основою для його обчислення є дерево, глибина якого дорівнює його довжині і загальна кількість вершин дорівнює кількості вершин побудованих в процесі перебору, причому у кожної вершини такого дерева є в точності B дочірніх вершин. Тобто, B пов'язане з довжиною шляху L і загальною кількістю побудованих вершин такими співвідношеннями:

$$B + B^2 + \ldots + B^L = T; \quad B \cdot (B^L - 1)/(B - 1) = T.$$

Значення B неможливо обчислити як явну функцію від L та T, тому вона надається у вигляді діаграм B = f(T) при різних значеннях L. Менші значення B відповідають більш цілеспрямованому пошуку.

4.13 Пошук на І/АБО графах

Нагадаємо, що помітка І чи АБО приписується вершині дерева типу І/АБО. Вершиною типу І називають батьківську вершину, що відображає складену базу даних і має множину дочірніх вершин, кожна з яких відображає одну з її складових баз даних. Вершина АБО є батьківською вершиною, що відображає деяку складову базу даних і має множину дочірніх вершин, кожна з яких відображає одну з баз даних, які отримують при застосуванні альтернативних правил до її "батьківської" бази.

Оскільки однакові бази даних можуть бути породжені різними послідовностями застосування правил, то розглянемо І/АБО графи, а не їх окремий випадок - дерева. Наприклад, деяка база даних може бути отримана як в наслідок декомпозиції складеної бази даних, так і в наслідок застосування одного з правил до деякої складової бази даних. В цьому випадку вершина буде мати водночас і тип І, як складена база даних, і тип АБО, як складова іншої бази даних. В зв'язку з цим, не будемо, як правило, називати вершини графа вершинами типу І чи АБО, однак будемо називати самі структури І/АБО графами, а при розгляді дерев типу І/АБО будемо використовувати поняття вершин типу І та АБО.

Визначимо І/АБО графи, як гіперграфи, а дуги, які з'єднують пари вершин будемо називати гіпердугами. Гіпердуги, які з'єднують батьківську вершину з множиною дочірніх вершин будемо називати k-зв'язками (Якщо всі зв'язки є 1-зв'язками, то як окремий випадок отримаємо звичайний граф). Розглянемо приклад І/АБО графа (рисунок 4.5).

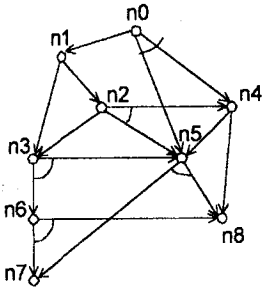


Рисунок 4.5 – Приклад І/АБО графа

Помітимо, що вершина n_0 має 1-зв'язку, що спрямована до спадкоємця n_1 (при цьому її можна розглядати як АБО вершину), і 2-зв'язку, яка направлена до множини спадкоємців n_4 і n_5 (при цьому її можна розглядати як вершини типу І). При k>1, k-зв'язки позначаються за допомогою перетинок між дугами, які йдуть з вершини-предка до елементів множини спадкоємців. Помітимо також, що вершина n_8 один з пращурів n_5 є вершиною типу І, а інший пращур n_4 є вершиною типу АБО. На дереві

І/АБО кожна вершина має лише одного пращура. Вершина графа або дерева, яка не має пращура, називається кореневою. Вершина, яка не має спадкоємців - називається кінцевою. Система продукцій, що розкладається, задає неявний граф І/АБО. Кожна продукція відповідає зв'язці в цьому неявному графові. Вершини, в які входить ця зв'язка відображають складові бази даних, що утворюються в наслідок застосування правила з наступним розкладанням на компоненти. В неявному графі існує набір термінальних вершин, що відповідають базам даних, які задовольняють умову зупинки для системи продукцій. Можна вважати, що задача системи продукцій полягає в тому, щоб простежити граф розв'язання від початкової до термінальних вершин.

Граф розв'язання, який зв'язує вершину n з множиною N термінальних (розв'язних) вершин в графі І/АБО, аналогічний шляху в звичайному графі. Але на кожному крокові створення графа розв'язання треба обирати не дугу, а зв'язку, що виходить з батьківської вершини доки кожна отримана дочірня вершина не буде належати множині N. Наприклад, для розглянутого І/АБО графа можна побудувати два різних графа розв'язання (рисунок 4.6), які зв'язують вершину n_0 з множиною вершин $\{n_7, n_8\}$ (відмітимо, що можна побудувати і інші графи розв'язання). Припустивши, що І/АБО граф є ациклічним, а це гарантує завершення процедури, можна надати точне рекурсивне визначення графа розв'язання. Позначимо через G' граф розв'язання, який зв'язує вершину n з множиною вершин N графа G типу І/АБО. Відмітимо що G' є підграфом G. Якщо n - елемент з N, то G' складається з однієї вершини n. Якщо n не належить N і якщо з n виходить k-зв'язка, яка направлена до вершин $\{n_1, ..., n_k\}$, причому існує граф розв'язання, що іде до N з кожної n_i, (i = 1, ..., k), то G' складається з вершини n, k-зв'язки та вершин $\{ n_1, ..., n_k \}$. Якщо не витримані обидві умови, то графа розв'язання, який зв'язує n з N, не існує.

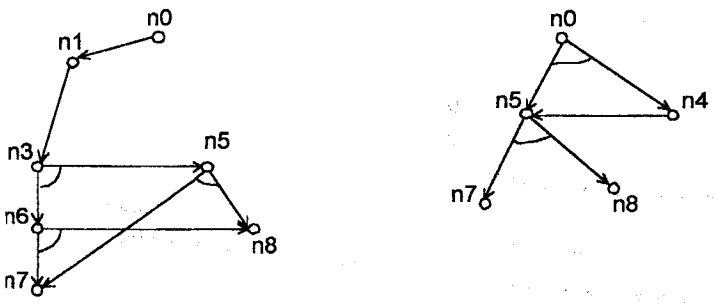


Рисунок 4.6 – Приклад графа розв'язання

Позначимо вартість графа розв'язання, який зв'язує деяку вершину n з N через k(n, N). Тоді вартість k(n, N) може бути обчислена рекурсивно

таким чином: 1) якщо n - елемент N, то k(n, N) = 0; 2) якщо n не належить N, з n виходить зв'язка, яка спрямована на множину дочірніх вершин $[n_1, ..., n_j\}$ в графові розв'язання. Нехай вартість такої ззв'язки дорівнює C_n. Тоді $k(n, N) = C_n + k(n, N) + ... + k(n_j, N)$. Таким чином, вартість графа G', який зв'язує n з N, дорівнює вартості зв'язки, що виходить з n (у межах G'), плюс суми вартостей графів розв'язань, що зв'язують нащадків n (у межах G') з N. Таке рекурсивне визначення коректне лише для ациклічних графів.

При такому визначенні в процесі підраховування вартості самого графа розв'язання, вартості деяких його зв'язок можуть враховуватись більш ніж один раз. В загальному випадку при підраховуванні вартості графа розв'язання, який зв'язує n з N, вартість зв'язки, що виходить з деякої вершини m, враховується стільки разів, скільки шляхів веде з n в m в графі розв'язання. Таким чином, вартості двох графів розв'язань, що були розглянуті в наших прикладах, складають 8 і 7 одиниць відповідно, при умові, що вартість кожної k-зв'язки складає k одиниць.

Граф розв'язання називають оптимальним, якщо він має мінімальну вартість, яка позначається як h(n).

4.14 Евристична процедура AO* пошуку на І/АБО графах

Розглянемо процедуру пошуку з використанням евристичної функції $\tilde{h}(n)$, що є оцінкою функції h(n) - вартості оптимального графа розв'язання. Як і в процедурі Graphsearch, процедура спрощується, якщо $\tilde{h}$ (n) відповідає певним умовам.

Будемо вважати, що $\tilde{h}(n)$ задовольняє умову монотонності, тобто що в неявно заданому графові для кожної ззв'язки, що спрямована з вершини n до її спадкоємців $n_1,...n_k$ має місце нерівність:

$$\tilde{h}(n_0) \le c + \tilde{h}(n) + .. + \tilde{h}(n_k), \text{ де c - вартість зв'язки.}$$

Сформулюємо тепер евристичну процедуру пошуку на графах І/АБО [6]:

1. Побудувати граф пошуку G з однієї початкової вершини s. З вершиною s зв'язати вартість $q(s)= \tilde{h}(s)$. Якщо s - термінальна вершина помітити її як SOLVED.

2. **until** s не буде помічено як SOLVED **do**:

3. **begin**

4. Обчислити частковий граф розв'язання G' в G, спускаючись в межах G по помічених зв'язках, що йдуть з s (ззв'язки в G будуть помічені на наступному крокові).

5. **select** деяку нетермінальну вершину n в G (її вибір буде обговорено пізніше).

5. **select** деяку нетермінальну вершину n в G (її вибір буде обговорено пізніше).

6. Розкрити n, породивши всіх її спадкоємців і призначити їх дочірніми вершинами n в G. З кожним нащадком n_j, який ще не з'являвся в G, зв'язати вартість $q(n_j) = \tilde{h}(n_j)$. Дочірні вершини, які є термінальними, позначити як SOLVED (випадок, коли у вершини n відсутні спадкоємці обговоримо пізніше).

7. Створити одноелементну множину S, яка вміщує єдину вершину n.

8. **until S** стане пустим, **do.**

9. **begin.**

10. Видалити з S вершину m, у якій в G немає спадкоємців, що належать S.

11. Переглянути вартість q(m) вершини m, для чого для кожної ззв'язки, що виходить з m в множину вершин $[n_{1i}, ..., n_{ki}]$, обчислити $q_i(m) = c_i + q(n_{1i}) + ... + q(n_{ki})$. (Значення $q(n_{ji})$ або вже обчислені під час попереднього проходу через цей внутрішній цикл, або, якщо цей прохід перший, беруться з кроку 6). Знайти мінімальне значення q(m) на множині всіх зв'язок, що виходять з m і надати це мінімальне значення q(m), а зв'язку на якій досягається мінімум помітити, після цього необхідно анулювати стару помітку, якщо вона відрізняється від нової. Якщо вершина, на яку ця зв'язка вказує, і всі її дочірні вершини помічені як SOLVED, помітити як SOLVED і вершину m.

12. Якщо m помічена як SOLVED або якщо нова вартість m відрізняється від попередньої, то включити до множини S всі ті батьківські вершини m, для яких m є одним з нащадків в гілці, що починається з поміченої ззв'язки.

13. **end**

14. **end**

В процедурі АО* можна виділити дві основні операції. Перша (кроки 4-6) спрямована зверху до низу (в напрямку зростання графа) і простежує граф за поміченими зв'язками, визначаючи найкращий частковий граф розв'язання. Помітки на зв'язках, що були нанесені раніше, в кожній вершині графа пошуку виділяють найкращий в даний момент часу частковий граф розв'язання (граф розв'язання називають частковим, якщо не всі його вершини є термінальними). Одна з кінцевих вершин найкращого часткового графа розв'язання, що не є термінальною, розкривається і її спадкоємцям приписується деяка вартість.

Друга основна операція в процедурі АО* спрямована знизу вверх і є процедурою перегляду вартостей, поміток зв'язок, присвоєння помітки SOLVED (кроки 7-12). Починаючи з останньої розкритої вершини, ця процедура змінює її вартість (використовуючи при цьому заново обчислені вартості її нащадків) і помічає зв'язки, що виходять [illegible]

який оцінюється як "найкращий". Така змінена оцінка переноситься вверх по графові (мова йде про ациклічні графи, в яких відсутні цикли). Змінена вартість q(n) є більш точною оцінкою вартості оптимального графа розв'язання, що зв'язує n з множиною термінальних вершин. Вартості можуть змінюватись тільки в тих вершинах, що є попередніми для вершин, вартість яких змінилася. Оскільки прийнята умова монотонності, то значення вартості можуть змінюватись тільки в бік зростання. Тому змінювати вартість треба не для всіх дочірніх вершин, а лише для тих, чиї найкращі часткові графи розв'язання вміщують нащадків з зміненою вартістю (звідки випливає і крок 12).

Щоб не ускладнювати алгоритм АО* на крокові 6 нехтують можливістю того, що обрана для розкриття вершина може не мати жодного спадкоємця. Такий випадок може бути задоволений на крокові 11, шляхом присудження занадто великої вартості q будь-якій вершині m (або, в більш загальному випадку, будь-якій вершині, що не належить ніякому графу розв'язання). Тоді друга операція процедури перенесе цю велику вартість вверх по графу, за рахунок чого буде виключена будь-яка можливість вибору графа, який вміщує цю вершину, в якості передбачуваного графа розв'язання.

Якщо задовольняється умова монотонності і граф розв'язання існує, то алгоритм АО* забезпечує знаходження оптимального графа розв'язання з вартістю q(s).

Розглянемо роботу алгоритму АО* на прикладі раніше розглянутого графа (рисунок 4.5), для якого введемо такі значення оцінок: $\tilde{h}(n_0)=0$, $\tilde{h}(n_1)=2$, $\tilde{h}(n_2)=4$, $\tilde{h}(n_3)=4$, $\tilde{h}(n_4)=1$, $\tilde{h}(n_5)=1$, $\tilde{h}(n_6)=2$, $\tilde{h}(n_7)=0$, $\tilde{h}(n_8)=0$.

Нехай термінальними будуть вершини n_7, n_8, а вартість кожної k-зв'язки дорівнює k одиниць. Помітимо, що в даному прикладі функція h* є нижньою межею для h і задовольняє умову монотонності. Побудуємо графи пошуку, які будуть отримані після кожного з проходів по зовнішньому циклу алгоритму АО*.

Тут біля кожної вершини показано змінені значення q_i. Помітки зв'язок позначені товстими стрілками. Вершини, що помічені як SOLVED позначені чорними колами. На першому проході розкривається вершина n_0, потім n_1, потім n_5 і n_4.

На крокові 5 вибір нетермінальної вершини для розкриття намагаються зробити так, щоб з найбільшою ймовірністю змінити оцінку найкращого часткового графа розв'язання. Тобто може статися так, що розкриття кінцевої вершини з максимальним значенням h скоріше за все призведе до зміни оцінки (якщо вже є така дочірня вершина, яка "приховує" піддерево максимальної вартості, то краще знайти його як найшвидше, бо раніше чи пізніше, вона все одно буде відкрита).

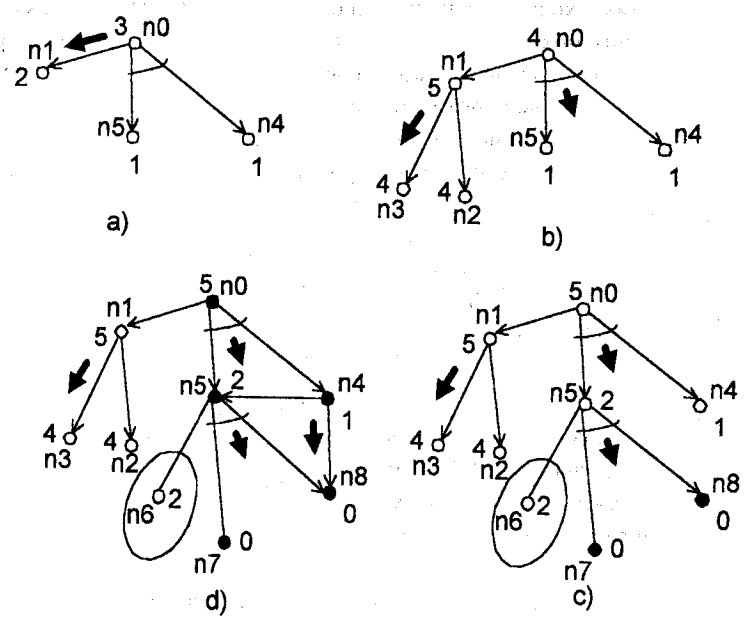


Рисунок 4.7 – Графи пошуку згідно з алгоритмом АО*:
а) після першого циклу;
б) після другого циклу;
в) після третього циклу;
г) після четвертого циклу

До алгоритму АО* можна вносити зміни, щоб в конкретних ситуаціях він був більш ефективним. Наприклад, можна не знаходити новий найкращий частковий граф розв'язання після кожного розкриття вершин, а робити це після розкриття декількох кінцевих вершин і деякої кількості їх нащадків. Це зменшує затрати на обчислення графа розв'язання, але може привести до розкриття "зайвих" вершин, що йому не належать.

4.15 Пошук на ігрових деревах

Методики пошуку, подібні до раніше розглянутих, можна використовувати при знаходженні ігрових стратегій для деяких видів ігр, а саме: для гри двох осіб з повною інформацією, в якій гравці роблять ходи по черзі. Кожен з них знає який хід в кожній з попередніх ситуацій зробив його суперник і який хід він міг обрати. Результатом гри може бути виграш

одного з гравців, або нічия. Тут не розглядуються такі ігри, в яких результат, хоча б частково визначався випадковістю.

Для аналізу таких ігор можна використовувати системи, які багато в чому схожі на системи продукцій. Наприклад, в шахах глобальна база даних може вміщувати інформацію про позиції кожної з фігур на дошці, а правила виведення, моделювати дозволені ходи. Застосування цих правил до початкової і поточних баз даних призведе до побудови ігрового графа.

Розглянемо приклад застосування ігрових графів для такої гри. Є купка предметів. Гравці роблять ходи по черзі. Кожний хід повинен розбити будь-яку з купок на дві, обов'язково з різною кількістю предметів. Програє той хто не зможе зробити черговий хід, тому що всі купки будуть вміщувати по одному чи два предмети. Будемо називати гравців MIN і MAX. Нехай першим ходить MIN.

Почнемо гру з купки, яка має сім предметів.

Для такої гри база даних вміщує невпорядковану послідовність чисел, яка відображає кількість предметів в різних купках, а також ознаку того, хто повинен зробити наступний хід. При цьому описом початкової ситуації буде такий: (7) MIN. MIN може зробити один з трьох ходів, які призведуть до ситуацій: (6,1, MAX), (5,2, MAX), (4,3, MAX). Повний граф буде мати вигляд, приведений на рисунку 4.8. Всі кінцеві позиції є програшними для гравця, який робить наступний хід.

З графа можна побачити, що незалежно від поведінки MIN у MAX завжди існує можливість перемогти (виграшна стратегія позначена товстими стрілками). Для кожної вершини MIN треба показати, що гравець MAX може перемогти при будь-якому ході гравця MIN. Для вершини, з якої ходить MAX треба показати, що MAX має можливість досягти переможної ситуації виходячи хоча б з однієї з тих позицій, в яку він може перевести гру, зробивши черговий хід.

Відмітимо подібність виграшної стратегії гравця MAX з графом розв'язання в графові І/АБО. Вершини MIN подібні до вершин 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 перемога повинна бути здобута хоча б з однієї з дочірніх вершин. Термінальні вершини в даному випадку, це вершини, які відповідають перемозі гравця MAX.

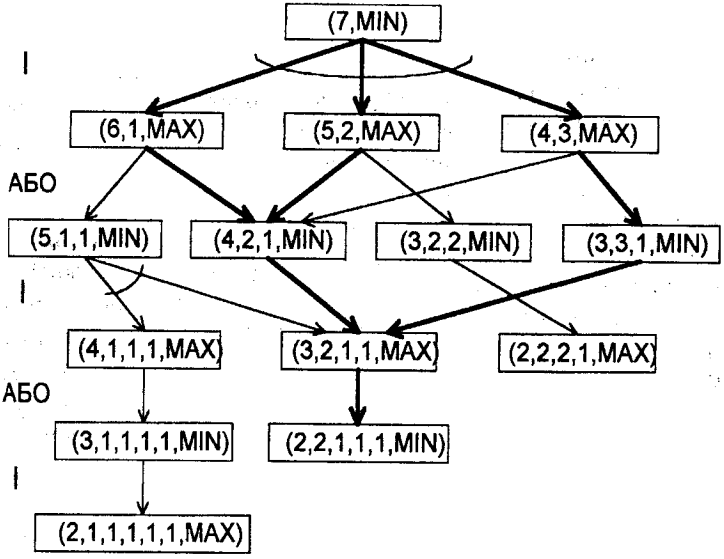


Рисунок 4.8 – Приклад графа пошуку на ігровому дереві

4.16 Мінімаксна процедура при переборі на ігрових деревах

При використанні графів розв'язання для ігрових ситуацій, вони представляють повну ігрову стратегію. При цьому груба оцінка навіть для гри у хрестики-нулики показує, що початкова вершина має 9 спадкоємців, кожна з них має по 8 спадкоємців і т. д. Це дає загальну кількість вершин 362880. Щоправда багато шляхів в цьому дереві закінчуються на термінальних вершинах більш високого рівня, і розмір дерева можна зменшити з урахуванням симетричних позицій гри. Тим не менш, для гри в шашки ігрове дерево містить приблизно 10^{40} вершин, а для гри в шахи - 10^{120}. Навіть якщо породжувати 3 вершини кожну наносекунду, то для породження ігрового дерева для гри в шахи буде потрібно 10^{21} століть. Методи евристичного пошуку не знижують коефіцієнт розгалуження так радикально, щоб отримати суттєві зміни. Тому для складних ігор доводиться погодитись з відсутністю можливості перебору, тобто доводиться відмовитись від бажання довести можливість досягнення виграшу або нічиєї (виключаючи можливі ендшпілі). Весь період існування Сонячної системи, тобто 4,6 млрд. років, відповідає часу, необхідному для аналізу шахової позиції всього на 12 ходів вперед, що потребує здійснення аналізу 40^{12} позицій.

Однак, мета пошуку може полягати

першого ходу в кожній з ігрових ситуацій, що виникають. Можна застосовувати пошук у ширину, глибину або евристичний пошук, треба лише змінити умови зупинки, обмежити час та об'єм пам'яті, чи найбільшу глибину вершин і т. ін. По закінчені пошуку можна виділити на графі пошуку претендента на "найкращий" перший хід, застосувавши до кінцевих вершин графа пошуку статичну функцію оцінювання, яка вимірює "цінність" позиції, що представлена кінцевою вершиною. Для кожної гри обчислення цінності засновано на своїх критеріях. При аналізі ігрових дерев як правило приймають угоду згідно з якою вигідній для гравця MAX позиції надають додатні значення функції оцінювання, а вигідній для гравця MIN - від'ємні. Значення функції оцінювання близькі до нуля відповідають позиціям, які не мають великої переваги для жодного з гравців.

Вибір першого ходу можна здійснити за допомогою процедури, яку в 1945 році запропонували Оскар Монгерштерн і Джон фон Нейман і яка отримала назву мінімаксної. Будемо вважати, що якщо б вибір на кінцевих вершинах робив гравець MAX, то він обрав би ту вершину, на якій значення функції оцінювання було б максимальним. Отже вершині типу MAX приписується обернена величина (backed-up value), рівна максимальній з оцінок її дочірніх вершин. З іншого боку, гравець MIN обрав би ту вершину, на якій значення функції оцінювання є мінімальним (тобто є найбільшим за модулем від'ємним числом). Отже, вершині типу MIN приписується повернена величина, що дорівнює мінімальній з оцінок її дочірніх вершин. Після присвоєння повернених значень всім кінцевим вершинам, будуються повернені значення для наступного рівня, причому вважається, що гравець MAX завжди обирає значення з найбільшими, а гравець MIN - значення з найменшими поверненими значеннями. Таке повернення в зворотному напрямку продовжується з рівня на рівень, доки повернені значення не будуть присвоєні безпосереднім спадкоємцям початкової вершини. Оскільки прийнята домовленість, що першим ходить гравець MAX, то при першому ході він повинен обрати дочірню вершину з найбільшим поверненим значенням. Тому що повернені значення? грунтуються на оцінці тих позицій, які розташовані ближче до цільової, вони є більш точними, ніж ті оцінки, які можна було б отримати в початковій ситуації. Тобто, повернені значення відповідають стратегії перегляду вперед на декілька ходів.

Розглянемо найпростіший приклад (рисунок 4.9) застосування мінімаксної процедури. Нехай гравці MAX і MIN ходять по черзі. Оцінки MAX вершин найнижчого рівня визначаються за допомогою деякої функції оцінювання. Оцінки всіх внутрішніх вершин визначають просуваючись знизу-вверх 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 показує "мінімаксно- оптимальну" гру суперників.

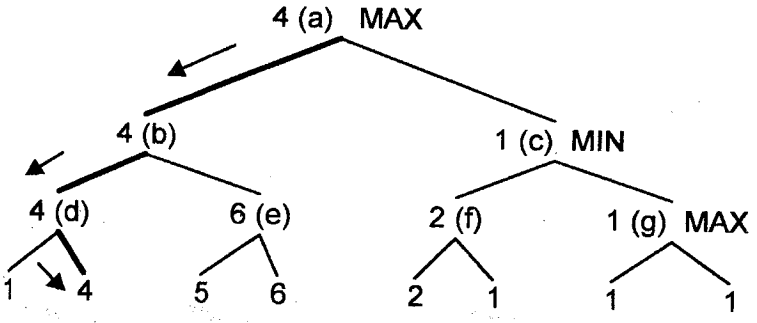


Рисунок 4.9 – Приклад застосування мінімаксної процедури

4.17 Альфа-бета процедура

Одна з найбільш складних проблем серед тих, що виникають при створенні ігрових програм полягає в тому, щоб уникнути 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 тут же обчислювати її статичну оцінку. Повернене значення також повинно приписуватись будь-якій позиції зразу ж, 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, можливо прийняти вірне рішення не уточнюючи, наскільки саме один з ходів є кращим за інший.

Розглянемо використання цього принципу на наведеному раніше прикладі (рисунок 4.9). Розпочинаючи пошук з позиції (a) (рисунок 4.10) 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. 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 привело 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).

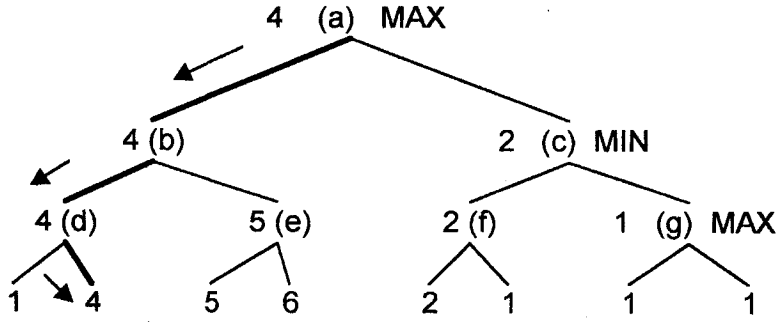


Рисунок 4.10 – Приклад реалізації альфа-бета процедури

Головна ідея алгоритму полягає в визначені двох граничних умов, позначених як альфа й бета, між якими має знаходитися оцінка кореневої вершини. Тобто, 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):

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 того ж самого 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-відсіченням, а згідно другого – бета-відсіченням (рисунок 4.10). Ще раз зупинимося на головному принципі процедури відсічення. Припустимо, що вершина S відповідає позиції, в якій хід належить гравцю MAX. При цьому він може зробити кілька ходів, два з яких показано на рисунку 4.11.

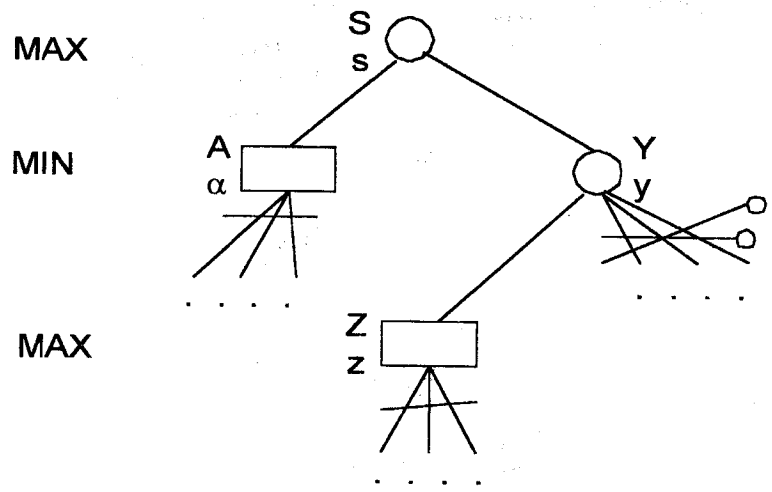


Рисунок 4.11 – Приклад застосування відсічень

В наслідок 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 методу 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 чи дорівнює α і y, то маємо $y \leq z \leq \alpha \leq s$. Тоді невідоме значення оцінки y 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 $z \geq \beta$ (бета-відсічення), де через β позначимо в даному випадку оцінку вершини A.

Сказане виконується і для вершин, що розташовані на суттєво різних глибинах дерева, при умові, що вони належать рівням одного і того ж самого гравця. 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 $y \leq z$, 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.

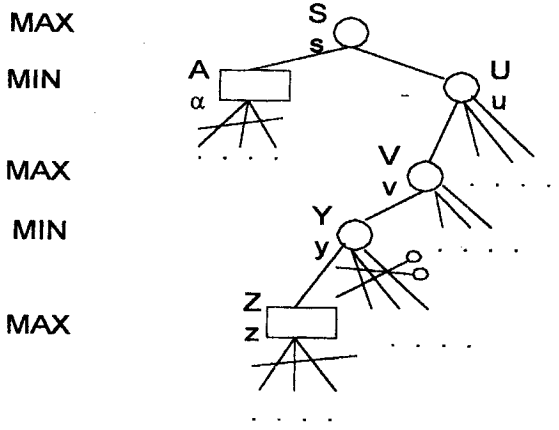


Рисунок 4.12 – Глибинне альфа-бета відсічення

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 $y \leq z$, а значення y в наслідок аналізу може лише зменшуватись. В другому випадку з $y = v$ маємо: $v = y \leq z \leq \alpha$. 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-відсічення). Аналогічно можна розмірковувати і для позицій, з яких перший хід робить MIN. В цьому випадку β-відсічення можна робити кожен раз, коли оцінка позиції, що виникає після ходу MIN, перевищує значення β.

Процедура відсічення може застосовуватись тільки з того моменту, коли отримані оцінки не менше ніж для двох вершин.

4.18 Ефективність пошуку за допомогою альфа-бета процедури

Для проведення альфа-бета відсічення хоча б частинка дерева повинна бути побудована на мінімальну глибину, оскільки обчислення значень альфа і бета засновані 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 з 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 вершина (за винятком кінцевих) 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 поверненими значеннями, то кількість відсічень буде максимальною. Так, при кількості кінцевих вершин ND буде виконуватися таке співвідношення:

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

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 тих самих об'ємах пам'яті. В такому ідеальному випадку (який неможливо досягнути на практиці) вершини на кожному рівні розташовуються згідно із значеннями оцінок, що монотонно зменшуються. В зв'язку з цим, для наближення до ідеального випадку в деяких програмах, раніше ніж застосувати алгоритм до глибини, припустимо 6, спочатку використовують цей алгоритм до глибини 2, після чого здійснюють впорядкування вершин згідно з отриманими оцінками. Далі виконують аналіз, наприклад до глибини 3 і знов перевпорядковують вершини і лише після цього, алгоритм застосовують до глибини 6. Два перших ходи, за які обробляється невелика кількість вершин, виконуються дуже швидко і дозволяють помітно наблизитись до ідеального впорядкування вершин для рівня 6. За рахунок цього, час обчислень зменшується порівняно з випадком застосування аналізу одразу на всю глибину без попередньої підготовки. Ігрові програми, засновані 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 (Кре3, 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 напівходів вперед. 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.

2. Ефект горизонту. В наслідок 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.

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

Завдання для виконання

1. Запропонувати евристичну функцію і побудувати дерево пошуку, за допомогою алгоритму А*, задачі про фішки, що знаходяться у такій початковій ситуації:

Б	Б	Б	Ч	Ч	Ч	

Дозволені ходи: - у сусідню пусту клітинку (вартість ходу - 1);
- стрибок через одну або дві фішки (вартість ходу дорівнює, відповідно, 1 або 2).
Метою гри є розташування білих фішок праворуч від чорних (місце розташування пустої клітинки значення не має).

2. Запропонуйте не менш як три оціночні функції і покажіть результати їх застосування в алгоритмі А* для вирішення задачі про комівояжера з такими початковими даними: загальна кількість міст - 5 (A, B, C, E, H); довжини шляхів - AB=2, AC=4, AE=4, BC=2, BH=3, CH=2, CE=2, EH=1.

3. Намалюйте дерево рішення для гри в якій беруть участь два гравці. Гравці по черзі беруть по одній, дві або три монетки з купки, яка на початку вміщує дев'ять монеток. Надайте просте описання виграшної стратегії.

4. Намалюйте дерево розв'язання для гри в якій беруть участь два гравці. Є купка монеток, яка вміщує на початку гри сім монеток. Гравці кожним ходом повинні розбивати початкову, або одну з створених купок на дві, кількість монеток в яких не повинна бути рівною. Програє той, хто не зможе зробити черговий хід, оскільки всі купки будуть вміщувати по одній або дві монетки. Надайте простий опис виграшної стратегії.

5. Побудувати дерево І/АБО для такої системи продукції, що розкладається:

Початкова база даних БД(C,B,Z).

Правила продукцій: П1: C -> (D,L); П2: C -> (B,M);

П3: B -> (M,M); П4: Z -> (B,M),

Термінальна умова: наявність у базі даних тільки символів M. Навести дерево розв'язання задачі.

6. Побудувати дерево І/АБО для такої системи продукції, що розкладається:

Початкова база даних БД(C).

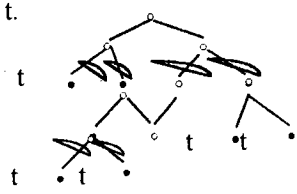
Правила продукцій: П1: C -> (B,G); П2: C -> (D,E); П3: C ->(E,F,F);

П4: B -> (D,F); П5: D->(F,F); П6: D -> (E,G);

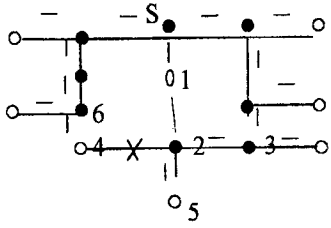
П7: E -> (F,G); П8: F->(G,G).

Термінальна умова: наявність у базі даних тільки символів G. Навести дерево розв'язання задачі.

7. Визначити, чи є розв'язним граф, який приведено на рисунку. Розв'язні вершини на рисунку позначені символами t.



8. Визначте, яким чином зміниться орієнтація показників після того, як при



розкритті вершини 1 з'ясується, що одним з її спадкоємців є вершина 2.

9. Використати двонаправлений пошук з застосуванням функції оцінювання для вирішення ігрової ситуації для гри у вісім.

2	8	3
1	6	4
7	*	5

Запропонувати не менш як дві оціночні функції і порівняти отримані результати.

10. Використати двонаправлений пошук з застосуванням функції оцінювання для вирішення ігрової ситуації у грі "квадрати, що обертаються".

Умова задачі. Необхідно розташувати цифри у порядку зростання (зліва направо, зверху до низу). Кожен хід полягає в повороті будь-якого квадрата, створеного чотирма сусідніми клітинками дошки на одну клітинку проти годинникової стрілки

7	1	6
6	4	3
8	9	2

Запропонувати не менше двох оціночних функцій і порівняти отримані результати.

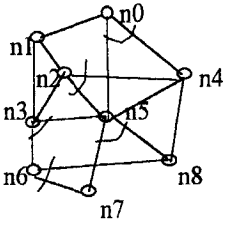
11. Використати двонаправлений пошук з застосуванням функції оцінювання для вирішення ігрової ситуації у грі "квадрати, що обертаються".

Умови задачі. Необхідно розташувати цифри у порядку зростання (зліва направо, зверху до низу). Кожен хід полягає в повороті будь-якого квадрата, створеного чотирма сусідніми клітинками дошки на одну клітинку проти годинникової стрілки.

7	9	1
4	5	3
8	6	2

Запропонувати не менше двох оціночних функцій і порівняти отримані результати.

12. За допомогою алгоритму АО* побудувати граф розв'язань для початкового І/АБО графу, що приведений на рисунку:

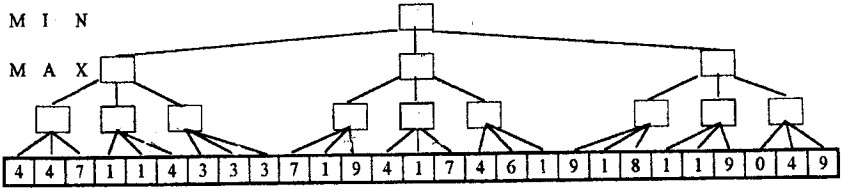


Значення оцінок вершин: $\tilde{h}(n_0)=0$, $\tilde{h}(n_1)=2$, $\tilde{h}(n_2)=4$, $\tilde{h}(n_3)=4$, $\tilde{h}(n_4)=1$, $\tilde{h}(n_5)=1$, $\tilde{h}(n_6)=2$, $\tilde{h}(n_7)=0$, $\tilde{h}(n_8)=0$. Термінальні вершини - n_7, n_8. Вартість кожної k-зв'язки дорівнює k одиниць.

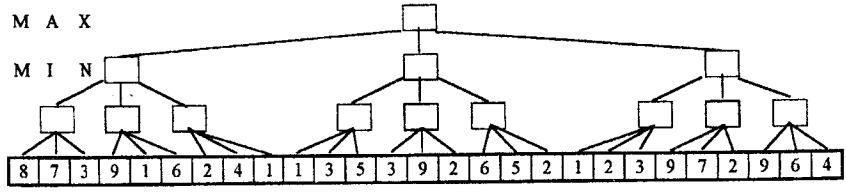
13. За допомогою алгоритму АО* пошуку на І/АБО графах перетворіть цифру 7 в рядок одиниць, використовуючи для заміни однієї цифри на рядок цифр такі правила
7 → 6,1; 7 → 4,3; 7 → 5,2; 6 → 4,2; 6 → 5,1; 5 → 3,2; 4 → 3,1; 3 → 2,1; 2 →1,1. Вважайте, що вартість k-зв'язки дорівнює k одиниць, і що функція 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'язання. Визначте вартість графу розв'язання.

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 → 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'язання. Визначте вартість графу розв'язання.

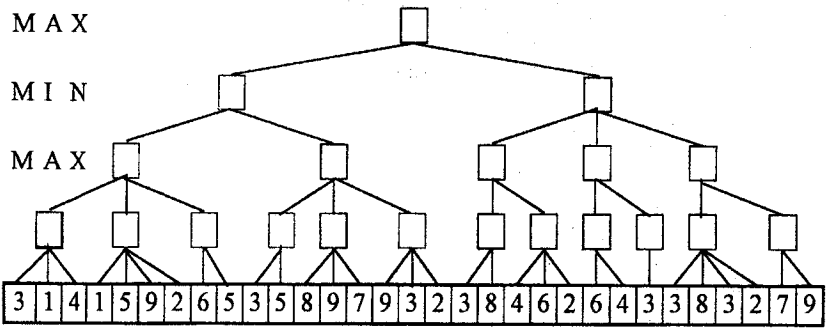
15. Здійсніть альфа-бета перебір на приведеному на рисунку дереві:



16. Здійсніть альфа-бета перебір на приведеному на рисунку дереві:



17. Здійсніть альфа-бета перебір на приведеному на рисунку дереві:



Питання для самоперевірки

1. Довести спроможність алгоритму А*.
2. Визначити поняття інформованості алгоритму А*. Проаналізувати два алгоритми А* різного ступеня інформованості.
3. Визначити поняття монотонного обмеження. Проаналізувати властивості алгоритму А*, який задовольняє монотонне обмеження.
4. Проаналізувати глибинне альфа-відсічення і бета-відсічення.
5. Проаналізувати евристичну силу функції оцінювання.
6. Проаналізувати пошук на ігрових деревах.
7. Проаналізувати пошук на І/АБО графах. Поняття графа розв'язання. Вартість графа розв'язання.
8. Проаналізувати приклади поширених евристик.
9. Проаналізувати альфа-бета процедуру, її переваги і недоліки.
10. Визначити двонаправлений пошук з використанням функції оцінювання.
11. Проаналізувати евристичні процедури пошуку на графі.
12. Проаналізувати евристичну процедуру АО* пошуку на І\АБО графах.
13. Проаналізувати мінімаксну процедуру перебору на ігрових деревах.
14. Визначити алгоритм А*.
15. Проаналізувати загальну процедуру пошуку на графах.
16. Визначити ефективність пошуку з використанням альфа-бета процедури.
17. Визначити поняття дерева пошуку. Проаналізувати процедуру формування дерева пошуку.
18. Проаналізувати основні властивості функції оцінювання.
19. Проаналізувати методи підвищення ефективності альфа-бета процедури.
20. Дати оцінку неінформованим процедурам пошуку на графах.
21. Визначити поняття розв'язуваності вершин І/АБО графів.
22. Проаналізувати вплив двох складових функції оцінювання для алгоритму А* на характер процесу пошуку.
23. Визначити аналогію ігрових дерев з І\АБО графами.
24. Дати оцінку використання І/АБО-графів для подання задач.
25. Проаналізувати правила альфа-відсічення і бета-відсічення.
26. Дати оцінку стратегії пошуку з поверненням.
27. Проаналізувати ефективність стратегій управління системами продукцій з точки зору кількості наявної інформації про задачу.
28. Визначити узагальнене поняття І/АБО - графів.

Перелік використаної літератури:

1. Гладун В.П. Эвристический поиск в сложных средах. - К.:Техніка, 1979.
2. Искусственный интеллект. - В 3-х кн.:Справочник/ Под ред. Э.В.Попова - Кн.1., Д.А. Поспелова - Кн.2., В.Н.Захарова - Кн.3. - М.: Радио и связь, 1990.
3. Компьютер обретает разум: Пер. с англ./Под ред. В.Л.Стефаника. - М.: Мир, 1990.
4. Логический подход к искусственному интеллекту: от классической логики к классическому программированию: Пер. с фр./ Тейз А., Грибомон П., Луи Ж. и др. - М.: Мир, 1990.
5. Лорьер Ж.-Л. Системы искусственного интеллекта: Пер.с фр. - М.: Радио и связь, 1991.
6. Нильсон.Н. Проблемы искусственного интеллекта: Пер. с англ. - М.: Радио и связь, 1985.
7. Осуга С. Обработка знаний: Пер. с япон. - М.: Мир, 1989.
8. Поспелов Д.А. Моделирование рассуждений. Опыт анализа мыслительных актов. - М.: Радио и связь, 1989.
9. Слейгл Дж. Искусственный интеллект: Пер. с англ. - М.: Мир, 1979.
10. Уинстон П. Искусственный интеллект: Пер.с англ. - М.: Мир, 1980.
11. Хант Э. Искусственный интеллект: Пер. с англ. - М.: Мир, 1978.
12. Эндрю А. Искусственный интеллект: Пер. с англ. - М.: Мир, 1985.

Міністерство освіти і науки України
Вінницький державний технічний університет

Навчальне видання

Володимир Іванович Месюра
Любов Михайлівна Ваховська

Основи проектування систем штучного інтелекту

Навчальний посібник

Вінниця ВДТУ 2000

Редактор С.А. Малішевська

Формат 29.7x42 ¼
Гарнітура Times New Roman
Друк різографічний
Зам. № 2000 - 010С
Тир. 39 прим.

Віддруковано в комп'ютерному інформаційно-видавничому центрі ВДТУ
м. Вінниця, Хмельницьке шосе, 95, ВДТУ, ГНК, 9-й поверх
Тел. (0432) 44-01-59