

Лекція 4. ПОШУК РІШЕНЬ ІЗ У ПРОСТОРІ СТАНІВ. МЕТОДИ ЕВРИСТИЧНОГО ПОШУКУ

- Алгоритм A^*
- Генетичний алгоритм
- Методи пошуку рішень ІЗ у разі зведення задач до сукупності підзадач (метод редукції)

1. Алгоритм A^*

Алгоритм A^* спершу відвідує ті вершини, які ймовірно ведуть найкоротшим шляхом до мети. Аби розпізнати такі вершини, кожній відомій вершині x співставляється значення $f(x)$, яке дорівнює довжині найкоротшого шляху від початкової вершини до кінцевої, який пролягає через обрану вершину. Вершини з найменшим значенням f обираються в першу чергу. Функція $f(x)$ для вершини x визначається так:

$$f(x) = g(x) + h(x)$$

де $g(x)$ – функція, значення якої дорівнюють вартості шляху від початкової вершини до x ; $h(x)$ евристична функція, оцінює вартість шляху від вершини x до кінцевої вершини. Використана евристика не повинна давати завищену оцінку вартості шляху. Прикладом оцінки може служити пряма лінія: загальний шлях не може бути коротшим за пряму лінію.

Приклад алгоритму A^* . Подорожуючому необхідно потрапити з точки А в точку В, які розділені бар'єром (стіною), як показано на рис. 1.

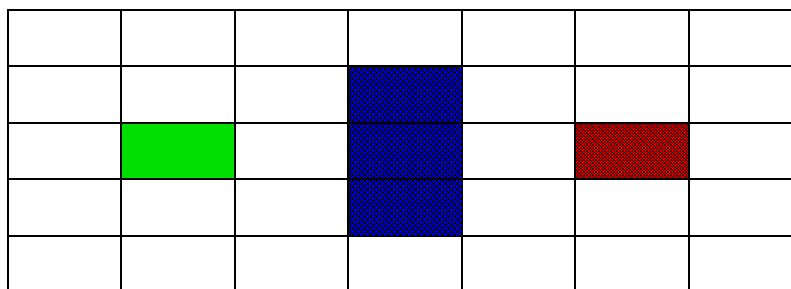


Рис. 1. Постановка задачі алгоритму A^*

Точка А позначена зеленим кольором, точка В – червоним, а стіна – синім.

Першим кроком є розбиття нашої області на підобласті – квадрати. Такий підхід спрощує нашу область пошуку до простого двовимірного масиву. Кожен елемент масиву представляє одну з кліток сітки, а його значенням буде прохідність цієї клітки (прохідна і непрохідна). Для знаходження шляху нам необхідно визначити, які нам потрібні клітки для переміщення з точки А в точку В.

Як тільки шлях буде знайдено, наш подорожній почне рухатися з центру однієї клітки на центр наступної, поки не досягне цільової клітки.

Ці центральні точки називають "вершинами". Коли ви що-небудь читаєте про пошук шляху, то часто можна зіткнутися з обговоренням вершин. Чом би просто не назвати їх клітками? Тому що завжди можливо розділити вашу область пошуку на щось відмінне від квадратів. Наприклад, на прямокутники, шестикутники, трикутники, або будь-яку іншу фігуру. І вершини можуть розташовуватися де завгодно – в центрі, уздовж граней або ще де-небудь. Ми використовуємо цю систему, оскільки вона найпростіша.

Як тільки ми спростили нашу область пошуку до деякого числа вершин, нам потрібно почати пошук для знаходження найкоротшого шляху. Почнемо з точки А, перевіряючи сусідні клітки і рухаючись далі до того часу, поки не зайдемо у цільову точку.

Нам необхідно обробляти два списки: "відкритий" і "закритий".

"Відкритий список" – це список кліток, які потрібно перевірити.

"Закритий список" – це список кліток, які більше не потрібно перевіряти.

Алгоритм:

1. Додаємо стартову точку А у "відкритий список" кліток, які потрібно обробити. Відкритий список це щось на зразок списку покупок. В даний момент є тільки один елемент в списку, але пізніше ми додамо ще. Список містить клітки, які можуть знаходитися уздовж шляху, який ви виберете, а може і ні.

2. Шукаємо доступні або прохідні клітки, що граничать із стартовою точкою, ігноруючи клітки із стінами, водою або іншою непрохідною областю. І також додаємо їх у відкритий список. Для кожної з цих кліток зберігаємо точку А, як "батьківську клітку". Ця батьківська клітка важлива, коли ми простежуватимемо наш шлях.

3. Видаляємо стартову точку А з відкритого списку і додаємо її в "закритий список" кліток. Тепер повинно бути щось схоже на рис. 2.

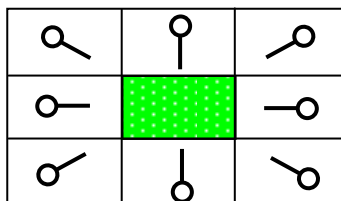


Рис. 2. Стартова точка із покажчиками

На цій ілюстрації зелений квадрат в центрі – ваша стартова точка, яка знаходиться в закритому списку. Всі сусідні клітки в даний момент знаходяться у відкритому списку.

Кожна з них має покажчик, направлений на батьківську клітку, яка в нашому випадку є стартовою точкою.

Далі ми виберемо одну з сусідніх кліток у відкритому списку і практично повторюємо описаний вище процес. Але яку клітку ми виберемо? Ту, у якої менше вартість $F = f(x) = G + H$, де G = вартість пересування із стартової точки A до даної клітки за знайденим шляхом до цієї клітки; H = приблизна вартість пересування від даної клітки до цільової, тобто точки B . Вона зазвичай є **евристичною функцією**. Причина, чому вона так називається в тому, що це припущення. Ми дійсно не знаємо довжину шляху, поки не знайдемо сам шлях, тому що в процесі пошуку нам може зустрітися безліч перешкод (наприклад, стіни і вода).

4. Обчислюємо вартість F для кожної клітки у відкритому списку.

В даному прикладі ми виберемо вартість 10 для горизонтальних і вертикальних пересувань, а для діагональних – 14. Ми використовуємо ці числа тому, що пройдена по діагоналі відстань приблизно в 1,414 разів (корінь з 2) більше вартості пересування по горизонталі або вертикалі. Для простоти ми використовуємо 10 і 14. Співвідношення дотримується і ми уникаємо обчислення квадратного кореня і десяткового дробу. Це пов'язано з тим, що використання цілих чисел значно прискорює роботу алгоритму.

Отже, **встановлення вартості G** уздовж шляху до поточної точки полягає в тому, щоб узяти G батьківської клітки і додати 10 або 14, залежно від діагонального або ортогонального (не діагонального) розташування поточної клітки щодо батьківської клітки. Необхідність використання цього методу стане очевидною трохи пізніше, коли ми віддалимось від стартової точки більш ніж на одну клітку.

Вартість H може бути обчислена багатьма способами. Метод, який ми використовуємо, називається методом Манхеттену (Manhattan method), згідно з яким рахуємо загальну кількість кліток, необхідних для досягнення цільової точки від поточної, по горизонталі і вертикалі, ігноруючи діагональні переміщення і будь-які перешкоди, які можуть опинитися на шляху. Потім ми перемножуємо загальну кількість отриманих кліток на 10.

Вартість F обчислюється шляхом додавання вартостей G і H . Значення G , H і F записуємо у кожну клітку відкритого списку: G – у лівий нижній кут, H – у нижній правий кут, а F – у верхній лівий кут. Отримаємо наступну ілюстрацію (рис 3).

5. Для продовження пошуку ми просто вибираємо клітку з найменшою вартістю F зі всіх кліток, що знаходяться у відкритому списку. З наших початкових 9 кліток, залишилося 8 у відкритому списку, а стартова клітка була внесена до закритого списку. Клітка з найменшою вартістю F знаходиться прямо праворуч від стартової клітки, і її вартість $F = 40$. Тому ми вибираємо цю клітку як нашу наступну клітку.

74 14	60 10	54 14				
60 10		F 40 G 10				
74 14	60 10	54 14				

Рис. 3. Вартості G , H і F кліток відкритого списку

6. Далі видаляємо клітку з найменшою вартістю $F=40$ з відкритого списку і додаємо в закритий список. Потім ми перевіряємо сусідні з нею клітки. Клітки, відразу праворуч від цієї клітки – стіни, тому ми їх ігноруємо. Клітка, прямо зліва – стартова клітка, вона знаходиться в закритому списку, тому ми її теж ігноруємо.

Залишилися 4 клітки, які вже знаходяться у відкритому списку, тому ми повинні перевірити, чи не коротший шлях по цих клітках, використовуючи поточну клітку. Порівнювати будемо *за вартістю G* . Давайте подивимося на клітку, прямо під нашою вибраною кліткою. Її вартість G рівна 14. Якщо ми рухатимемося по цій клітці, вартість G буде рівна 20 (вартість $G = 10$, щоб добратися до поточної клітки плюс 10 для руху вертикально вгору, до сусідньої клітки). Вартість $G = 20$ більше, ніж $G = 14$, тому це буде не кращий шлях. Це стане зрозумілим, якщо поглянути на рис. 4. Доцільнішим буде рух по діагоналі на одну клітку, чим рух на одну клітку по горизонталі, а потім одну по вертикалі.

Коли ми повторимо цей процес для всіх 4-х сусідніх кліток, які знаходяться у відкритому списку, то дізнаємося, що жоден з шляхів не є кращим при русі по цих клітках через вибрану, тому нічого не міняємо. Таким чином, вибрану клітинку з

вартістю $F = 40$ можемо забрати з відкритого списку та додати в "закритий список" кліток, які більше не потрібно перевіряти.

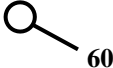

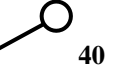
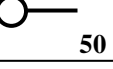
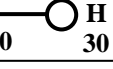
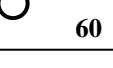

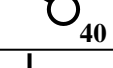
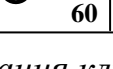

74 14  60	60 10  50	54 14  40	[Closed List]			
60 10  50	[Open List]	F 40 G 10  H 30	[Closed List]		[Closed List]	
74 14  60	60 10  50	54 14  40	[Closed List]			
	88 28  60	74 24  50				

Рис. 4. Додавання кліток у відкритий список для нової стартової точки

7. Тепер ми проходимо весь відкритий список, який зменшився до 7-ми кліток, і вибираємо клітку з найменшою вартістю F . Цікаво, що в цьому випадку існує 2 клітки з вартістю 54. Так яку ми виберемо? Це не має ніякого значення. В цілях збільшення швидкості пошуку можна вибрати останню клітку, яку ми додали у відкритий список. Це попередить пошук у виборі кліток, до яких можна буде звернутися пізніше, коли ми підберемося ближче до мети. Але насправді це не так вже важливо. (От чому дві версії A^* можуть знайти різні шляхи з однаковою довжиною.)

Нехай вибрана клітка прямо вниз, справа від стартової. Призначаємо її новою стартовою точкою.

8. Шукаємо доступні або прохідні клітки, що граничать із стартовою точкою, ігноруючи клітки із стінами, водою або іншою непрохідною областю (рис. 4).

На цей раз, коли ми перевіряємо сусідні клітки, бачимо, що клітка прямо справа – стіна і ми її пропускаємо. Так само поступаємо і з кліткою, яка знаходиться прямо над нею. Так само ми ігноруємо клітку, яка знаходиться прямо під стіною. Чому? Тому, що ви не можете дістатися до тієї клітки без зрізу кута найближчої стіни. Спочатку ви повинні спуститися вниз, а тільки потім рухатися на цю клітку. (Зауваження: Це правило зрізу кутів необов'язкове. Його використання залежить від розташування ваших вершин.)

Залишається ще 5 кліток. 2 клітки, що знаходиться під поточною, ще не у відкритому списку, тому ми їх додаємо у відкритий список і призначаємо поточну клітку їх "батьком". З 3-х інших кліток 2 вже знаходяться в закритому списку (стартова клітка і

клітка, прямо над нею) і ми їх ігноруємо. Остання клітка, яка знаходиться прямо зліва від поточної, перевіряється на довжину шляху по поточній клітці через цю клітку за вартістю G. Ні, шлях буде не коротший. Отже ми тут закінчили і готові перевірити наступну клітку у відкритому списку.

Повторюємо цей процес до того часу, доки не додамо цільову клітку у відкритий список (рис. 5).

94 24	80 20	74 24				
74 14	60 10	54 14				
60 10		F 40 G 10		74 64	68 68	
74 14	60 10	54 14		74 54	68 58	
94 24	80 20	74 24	74 34	74 44	74 54	

Рис. 5. Досягнення цільової клітинки

Основним **недоліком** алгоритму A* є потреба в пам'яті для збереження всіх відомих та досліджених вершин. Через це алгоритм A* непридатний для багатьох задач. Наприклад, повний граф ходів для гри П'ятнашки має $16! = 20\,922\,789\,888\,000$ вершин.

Алгоритм пошуку A* може бути замінений алгоритмом Дейкстри або «жадібним» алгоритмом. Алгоритм Дейкстри не використовує евристичних функцій і обирає наступну вершину в залежності від поточної вартості шляху. Натомість, жадібний алгоритм обирає наступну вершину лише на основі оцінки можливого маршруту через неї. Тому алгоритм Дейкстри можна застосовувати для пошуку шляху, коли кінцева вершина невідома і використання евристичної функції неможливе.

Деякі алгоритми намагаються уникнути потреби у великих обсягах пам'яті:

- IDA* (A* з ітеративним заглибленням), варіант ітеративного пошуку в глибину;
- RBFS (рекурсивний пошук за найкращим співпадінням, англ. Recursive Best-First Search), вимагає лінійну кількість пам'яті в залежності від довжини розв'язку
- MA* (A* з обмеженням пам'яті), SMA* (Simplified MA*), використовують лише наперед виділений обсяг пам'яті.

Ці алгоритми обмежують потребу в пам'яті за рахунок швидкодії. За деяких обставин, не обов'язково зберігати всі вершини в пам'яті. Погані вершини можуть бути забуті і згодом можуть бути наново відтворені. При використанні монотонної евристики та за умови наявності достатньої кількості пам'яті, ці алгоритми оптимальні. При надто суворих обмеженнях пам'яті, оптимальний розв'язок може бути не знайдений.

Алгоритм пошуку D^* (динамічний A^*) є вдосконаленням алгоритму A^* . Цей алгоритм враховує інформацію про структуру графа.

Серед інших алгоритмів можна назвати алгоритм Беллмана-Форда (дозволяє ребра з від'ємними вагами) або алгоритм Флойда-Воршала (знаходить найкоротший шлях між всіма парами вершин).

2. Генетичний алгоритм

Генетичні алгоритми – адаптивні методи пошуку, які останнім часом часто використовуються для вирішення завдань оптимізації (пошуку оптимального рішення). Вони засновані на генетичних процесах біологічних організмів: біологічні популяції розвиваються протягом декількох поколінь, підкоряючись законам природного відбору і за принципом "виживає найбільш пристосований", відкритому Чарльзом Дарвіном. Наслідуючи цьому процесу генетичні алгоритми здатні "розвивати" рішення реальних завдань, якщо ті відповідним чином закодовані. Наприклад, ГА можуть використовуватися, щоб проектувати структури моста, для пошуку максимального відношення міцності/ваги, або визначати найменш марнотратне розміщення для нарізки форм з тканини. Вони можуть також використовуватися для інтерактивного управління процесом, наприклад на хімічному заводі, або балансуванні завантаження на багатопроцесорному комп'ютері.

Дуже важливо зрозуміти, за рахунок чого ГА на декілька порядків перевершує по швидкості випадковий пошук в багатьох завданнях? Справа тут мабуть в тому, що більшість систем мають досить незалежні підсистеми. Внаслідок цього, при обміні генетичним матеріалом часто може зустрітися ситуація, коли від кожного з батьків беруться гени, відповідні найбільш вдалому варіанту певної підсистеми (решта "виродків" поступово вимирає). Іншими словами, ГА дозволяє накопичувати вдалі рішення для систем, що складаються з щодо незалежних підсистем (більшість сучасних складних технічних систем, і всі відомі живі організми). Відповідно можна передбачити і коли ГА швидше за

все дасть збій (або, принаймні, не покаже особливих переваг перед методом Монте-Карло) – системи, які складно розбити на підсистеми (вузли, модулі), а так само у разі невдалого порядку розташування генів.

У природі особини в популяції конкурують один з одним за різні ресурси, такі, наприклад, як їжа або вода. Крім того, члени популяції одного вигляду часто конкурують за залучення шлюбного партнера. Ті особини, які найбільш пристосовані до навколишніх умов, матимуть відносно більше шансів відтворити нащадків. Слабо пристосовані особини або зовсім не породять потомства, або їх потомство буде дуже нечисленним. Це означає, що гени від високо адаптованих або пристосованих особин будуть розповсюджатися в кількості нащадків, що збільшується, на кожному подальшому поколінні. Комбінація хороших характеристик від різних батьків іноді може приводити до появи "суперпристосованого" нащадка, чия пристосованість більша, ніж пристосованість будь-якого з його батька. Таким чином, вигляд розвивається, краще і краще пристосовуючись до місця існування.

Є багато способів реалізації ідеї біологічної еволюції в рамках ГА. Традиційним вважається ГА, представлений на рис. 6.

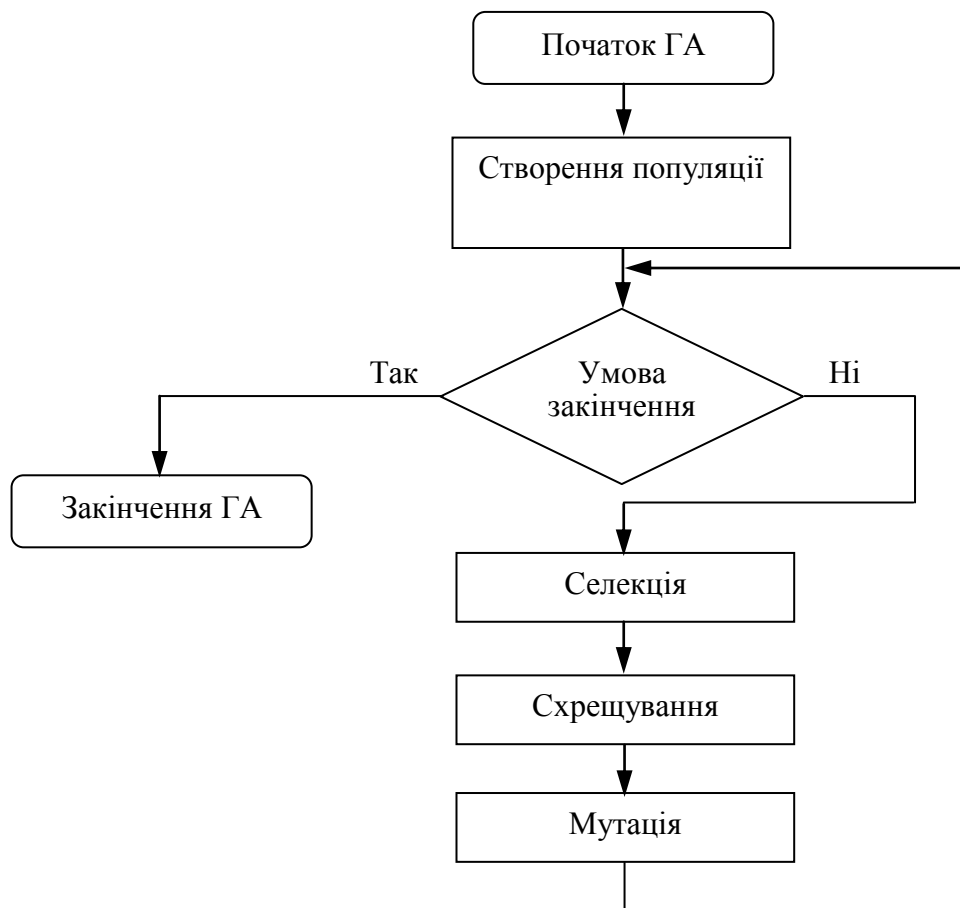


Рис. 6. Традиційний генетичний алгоритм

На першому кроці алгоритму **формується початкова популяція** особин, де кожна особина популяції є рішенням задачі, інакше кажучи, є кандидатом на рішення. Формування початкової популяції, як правило, відбувається з використанням якого-небудь випадкового закону, у ряді випадків початкова популяція може бути також результатом роботи іншого алгоритму. Необхідно відзначити, що особини популяції можуть складатися як з однієї, так і з декількох хромосом. Кожна хромосома особини у свою чергу складається з генів, причому кількість генів в хромосомі визначається кількістю варійованих параметрів вирішуваного завдання (аргументів цільової функції).

Для застосування генетичних операторів значення цих параметрів повинні бути представлені у вигляді двійкової послідовності, тобто двійкового рядка, що складається з декількох битий. Кількість битий при кодуванні гена (хромосоми) залежить від необхідної точності вирішуваного завдання. Оптимальним вважається такий вибір, коли виконується співвідношення

$$n \geq \log_2 \left(\frac{x_{\max} - x_{\min}}{\Delta} \right),$$

де x_{\max} і x_{\min} – максимальне і мінімальне значення аргументу x цільової функції;

Δ – похибка вирішення задачі;

n – кількість біт, що використовуються для кодування значення аргументу.

Кількість особин M в популяції визначається, як правило, емпіричним шляхом, бажано з інтервалу $n \leq M \leq 2n$.

На другому кроці роботи генетичного алгоритму відбувається перевірка **умови закінчення алгоритму**, адже є ймовірність, що під час формування початкової популяції утворено особу, яка є розв'язком задачі.

Як **критерії останову роботи генетичного алгоритму** прийнято розглядати наступні умови :

- отримано розв'язок задачі;
- сформовано задане число поколінь;
- популяція досягла заданого рівня якості (наприклад, 80% особин мають однакову генетичну структуру або однакове значення функції придатності) ;
- досягнутий деякий рівень збіжності, при якому поліпшення популяції не відбувається.

На третьому кроці роботи генетичного алгоритму відбувається відбір або **селекція** найбільш пристосовані особини, що мають найбільш переважні значення функції придатності в порівнянні з рештою особин.

Спочатку, пропорційний відбір призначає кожній структурі ймовірність $Ps(i)$ рівну відношенню її пристосованості до сумарної пристосованості популяції:

$$Ps(i) = \frac{f(i)}{\sum_{i=1}^n f(i)} .$$

Потім відбувається відбір (із заміщенням) всіх n особин для подальшої генетичної обробки, згідно величині $Ps(i)$. Простий пропорційний відбір – рулетка – відбирає особини за допомогою n "запусків" рулетки. Колесо рулетки містить по одному сектору для кожного члена популяції. Розмір i -го сектора пропорційний відповідній величині $Ps(i)$. При такому відборі члени популяції з вищою пристосованістю з більшою ймовірністю частіше вибиратимуться, чим особини з низькою пристосованістю.

У класичному генетичному алгоритмі відбір найбільш пристосованих особин здійснюється випадковим чином за допомогою різних методів генерації дискретних випадкових величин, що мають різні закони розподілу. Серед даних методів слід згадати відбір методом “колеса рулетки, пропорційний відбір, відбір з витісненням, рівноймовірний відбір і так далі. Кожен з перерахованих методів має як свої переваги, так і недоліки, наприклад, загальним недоліком вказаних методів є те, що при їх використанні в деякому поколінні якнайкращі особини популяції можуть бути втрачені. Одним із способів подолання цього недоліку є використання елітного відбору, який передбачає збереження “якнайкращої” особини в популяції (“краща” особина завжди переходить в наступне покоління).

На четвертому етапі реалізується операція **схрещування** особин. Сенс даної процедури зводиться до знаходження нових комбінацій генетичного коду хромосом шляхом обміну випадковим чином ділянок генетичного коду у двох особин, що пройшли відбір.

Після відбору, n вибраних особин піддаються **кросоверу** (іноді званому рекомбінацією) із заданою ймовірністю Pc .

n рядків випадковим чином розбиваються на $n/2$ пари. Для кожної пари з вірогідністю Pc може застосовуватися кросинговер. Відповідно з вірогідністю $1-Pc$ кросинговер

не відбувається і незмінні особини переходять на стадію мутації. Якщо кросинговер відбувається, отримані нащадки замінюють собою батьків і переходять до мутації.

Одноточковий кросинговер працює таким чином: просто беруться дві хромосоми, і перерізуються у випадково вибраній точці. Результируюча хромосома виходить з початку однієї і кінця іншої батьківських хромосом.

10010 11000	----->	1001011100
01000 11100		

Це сприяє “отриманню” додаткового числа нових особин, серед яких можуть опинитися як більш пристосовані, так і менш пристосовані особини. Це явище пояснюється тим, що точка схрещування (локус) вибирається випадковим чином.

На п'ятому етапі до особин популяції, отриманих в результаті схрещування, застосовується операція **мутації**.

Мутація є випадковою зміною хромосоми (зазвичай простою зміною стану одного з бітів на протилежне значення).

10010110	----->	10011110
----------	--------	----------

За допомогою даної операції можна отримувати принципово нові генотипи і фенотипи особини, що приводить до ще більшої різноманітності особин в даній популяції. Внесення таких випадкових змін дозволяє істотно розширити простір пошуку прийнятних рішень задачі цілочисельної оптимізації.

В процесі роботи алгоритму всі вказані оператори застосовуються багато разів і ведуть до поступової зміни початкової популяції у напрямі поліпшення значення функції придатності.

Перехід до п.2.

Приклад. Знайти максимум функції $f(x) = 2x^2 + 1$ (1) для цілочисельної змінної x , що приймає значення від 0 до 31.

Рішення:

1. Використовуючи двійкову систему числення, закодуємо числа від 0 до 31. Тоді хромосоми придбають вид двійкових послідовностей, що складаються з 5 бітів (0 як 00000, 31 як 11111)

В ролі функції пристосованості виступатиме функція $f(x) = 2x^2 + 1$. Тоді пристосованість хромосоми ch_i , $i = 1, 2, \dots, N$, буде визначатися значенням функції $f(x)$

для x , рівного фенотипу, відповідному генотипу ch_i . Позначимо ці фенотипи як ch_i^* . Тоді значення функції пристосованості хромосоми ch_i буде рівне $f(ch_i^*)$.

Виберемо випадковим чином **початкову популяцію**, що складається з 6 кодових послідовностей ($N=6$). Хай вибрані хромосоми:

$$ch_1 = [10011], ch_2 = [00011], ch_3 = [00111]$$

$$ch_4 = [10101], ch_5 = [01000], ch_6 = [11101]$$

Відповідні їм фенотипи – це числа від 0 до 31:

$$ch_1^* = 19, ch_2^* = 3, ch_3^* = 7, ch_4^* = 21, ch_5^* = 8, ch_6^* = 29$$

За формулою (1) розрахуємо значення функції пристосованості для кожної хромосоми в популяції, отримаємо: $f(ch_1) = 2x^2 + 1 = 2 * 19^2 + 1 = 723$,

$$f(ch_2) = 19, f(ch_3) = 99, f(ch_4) = 883, f(ch_5) = 129, f(ch_6) = 1683.$$

Сума всіх функцій пристосованості $\Sigma f(ch_i) = 723 + 19 + \dots + 1683 = 3536$.

2. Перевірка умови зупинки алгоритму. Число 29 дає на цьому кроці найкращий результат (максимум), можливо це розв'язок, якщо на наступному кроці результат не покращиться.

3. Селекція хромосом здійснюється методом рулетки.

Використовуючи формулу $Ps(i) = \frac{f(i)}{\sum_{i=1}^n f(i)}$ отримаємо, що:

$$V(ch_1) = f(ch_1) / \Sigma f(ch_i) = 723 / 3536 = 20,45; V(ch_2) = 0,54; V(ch_3) = 2,80;$$

$$V(ch_4) = 24,97; V(ch_5) = 3,65; V(ch_6) = 47,60.$$

Метод рулетки – найпростіший пропорційний відбір: рулетка відбирає особини з допомогою n "запусків" рулетки. Колесо рулетки містить по одному сектору для кожного члена популяції. Розмір i -ого сектора пропорційний відповідній величині $Ps(i)$. При такому відборі члени популяції з більш високою пристосованістю з більшою ймовірністю будуть частіше вибиратися, чим особини з низькою пристосованістю.

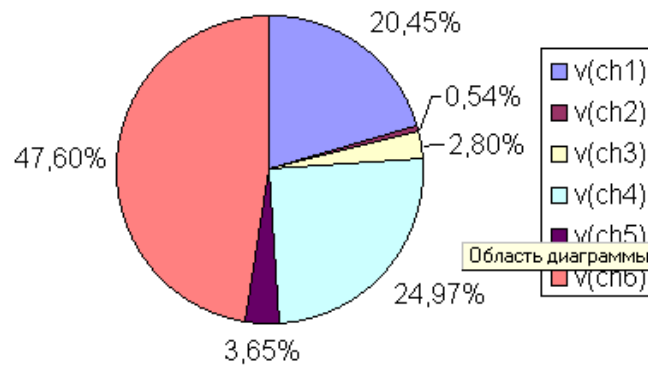


Рис 7. Колесо рулетки імовірнісного вибору хромосоми

Розиграш за допомогою колеса рулетки зводиться до випадкового вибору n чисел з інтервалу $[0, 100]$, яке вказує на відповідний сектор на колесі, тобто на конкретну хромосому (рис 7).

Допустимо, що вибрані числа: 97, 26, 54, 13, 31, 88.

Це означає вибір хромосом $ch_6, ch_4, ch_6, ch_1, ch_4, ch_6$:

$ch_6 = [11101], ch_4 = [10101], ch_6 = [11101],$

$ch_1 = [10011], ch_4 = [10101], ch_6 = [11101].$

4. Допустимо, що **схрещування** виконується з ймовірністю $p_c = 1$ і що для схрещування сформовані пари ch_1 і ch_4, ch_4 і ch_6, ch_6 і ch_6 . Крім того, випадковим чином вибрана точка схрещування, рівна 3 для хромосом ch_1 і ch_4 , а також точка схрещування, рівна 2 для хромосом ch_4 і ch_6 . Точка схрещування для пари однакових хромосом ch_6 і ch_6 ніяк не впливає на результат, тому і не вказується.

$ch_1 = [10011]$ і $ch_4 = [10101]$ – результат $ch_1 = [10001], ch_2 = [10111],$

$ch_4 = [10101]$ і $ch_6 = [11101]$ – результат $ch_3 = [10101], ch_4 = [11101].$

5. При умові, що ймовірність **мутації** $P_m = 0$, в нову популяцію включаються хромосоми

$ch_1 = [10001], ch_2 = [10111], ch_3 = [10101],$

$ch_4 = [11101], ch_5 = [11101], ch_6 = [11101].$

Декодувавши отримані послідовності, обчислимо відповідні фенотипи даних хромосом:

$ch_1^* = 17, ch_2^* = 23, ch_3^* = 21, ch_4^* = 29, ch_5^* = 29, ch_6^* = 29.$

Перехід на п. 2. Отримано знову число 29, але кількість особин з даним фенотипом збільшилась. Оскільки відбулося покращення популяції, продовжуємо роботу алгоритму.

Продовжуючи даний процес, вже на наступній ітерації може бути отримане хромосома [11111], з фенотипом рівним 31, значення функції пристосованості якої буде найбільшою (рівна 1923).

Налаштування параметрів генетичного алгоритму.

В даний час дослідники ГА пропонують багато інших операторів відбору, кросинговера і мутації. Ось лише найбільш поширені з них.

Турнірний відбір. Турнірний відбір реалізує n турнірів, щоб вибрати n особини. Кожен турнір побудований на вибірці до елементів з популяції, і вибору кращої особини серед них. Найбільш поширений турнірний відбір з $k=2$.

Елітні методи відбору гарантують, що при відборі обов'язково виживатимуть кращий або кращі члени популяції сукупності. При цьому частина самих кращих особин без яких-небудь змін переходить в наступне покоління.

Двоточковий кросовер і рівномірний кросовер.

У двоточковому кросинговері вибираються дві точки розриву, і батьківські хромосоми обмінюються ділянкою генетичної коди, яка знаходиться між двома цими крапками. У рівномірному кросинговері, кожен біт першого батька успадковується першим нащадком із заданою вірогідністю; інакше цей біт передається другому нащадкові. І навпаки.

Розмір популяції. Розмір популяції є вельми важливим елементом ГА. Якщо популяція дуже мала, генетичного матеріалу може не вистачити для вирішення завдання. Розмір популяції також впливає на коефіцієнт застосування операторів схрещування і мутації.

Області застосування генетичних алгоритмів.

Генетичний алгоритм використовується для вирішення багатьох завдань оптимізації:

- складання розкладів (виробничих, учбових і так далі);
- завдання розкрою-упакування;
- апроксимації і так далі.

3. Методи пошуку рішень ІЗ у разі зведення задач до сукупності підзадач (метод редуkcії)

Функціонування багатьох ІС носить цілеспрямований характер (прикладом можуть служити автономні інтелектуальні роботи). Типовим актом такого функціонування є рішення завдання планування шляху досягнення потрібної мети з деякої фіксованої початкової ситуації. Результатом рішення завдання повинен бути план дій – із сукупності дій. Такий план нагадує сценарій, у якому в якості відносини між вершинами виступають відносини типу: "ціль-підціль" "цілі-дія", "дія-результат" і т.п. Будь-який шлях у цьому сценарії, який веде від вершини, що відповідає поточній ситуації, у кожен із цільових вершин, визначає план дій.

Пошук плану дій виникає в ІС лише тоді, коли вона зіштовхується з нестандартною ситуацією, для якої немає заздалегідь відомого набору дій, що приводять до потрібної мети. Всі завдання побудови плану дій можна розбити на два типи, яким відповідають різні моделі: *планування в просторі станів* (SS-проблема) і *планування в просторі завдань* (PR-проблема).

У першому випадку вважається заданим деякий простір ситуацій. Опис ситуацій включає стан зовнішнього середовища й стан ІС, яка характеризується рядом параметрів. Ситуації утворюють деякі узагальнені стани, а дії ІС або зміни в зовнішньому середовищі приводять до зміни актуалізованих у цей момент станів. Серед узагальнених станів виділені початкові стани (зазвичай один) і кінцеві (цільові) стани. SS-проблема складається в пошуку шляху, що веде з початкового стану в один з кінцевих. Якщо, наприклад, ІС призначена для гри в шахи, то узагальненими станами будуть позиції, що складаються на шахівниці. Як початковий стан може розглядатися позиція, що зафіксована в цей момент гри, а як цільові позиції – множина нічийних позицій. Відзначимо, що у випадку шахів пряме перерахування цільових позицій неможливо. Матові й нічийні позиції описані мовою, яка відмінна від мови опису станів, і характеризуються розташуванням фігур на полях дошки. Саме це утрудняє пошук плану дій у шаховій грі.

При плануванні в просторі завдань ситуація трохи інша. Простір утворюється в результаті введення на множині завдань відносини типу: "частина - ціле", "завдання - підзадача", "загальний випадок - окремий випадок" і т.п. Інакше кажучи, простір завдань відбиває декомпозицію завдань на підзадачі (мети на підціль). PR-проблема складається в пошуку декомпозиції вихідного завдання на підзадачі, що приводить до завдань, рішення

яких системі відомо. Наприклад, ІС відомо, як обчислюються значення $\sin x$ й $\cos x$ для будь-якого значення аргументу і як виробляється операція розподілу. Якщо ІС необхідно обчислити $\operatorname{tg} x$, то рішенням РР-проблеми буде подання цього завдання у вигляді декомпозиції $\operatorname{tg} x = \sin x / \cos x$ (крім $x = p/2 + kp$).

Пошук планування в просторі завдань полягає в послідовному перетворенні вихідного завдання до усе більше простого доти, поки не будуть отримані тільки елементарні завдання. Частково впорядкована сукупність таких завдань складе рішення вихідного завдання. Розчленовування завдання на альтернативні множини підзадач зручно представляти у вигляді І/АБО-графа. У такому графі всяка вершина, крім кінцевий, має або кон'юнктивно зв'язані дочірні вершини (І-вершина), або диз'юнктивно зв'язані (АБО-вершина). В окремому випадку, при відсутності І-вершин, має місце граф простору станів. Кінцеві вершини є або заключними (їм відповідають елементарні завдання), або тупиковими. Початкова вершина (корінь І/АБО-графа) представляє вхідне завдання. Ціль пошуку на І/АБО-графі – показати, що початкова вершина розв'язна. Розв'язними є заключні вершини (І-вершини), у яких розв'язні всі дочірні вершини, і АБО-вершини, у яких розв'язна хоча б одна дочірня вершина. Розв'язний граф складається з розв'язних вершин і вказує спосіб можливості розв'язання початкової вершини. Наявність тупикових вершин приводить до нерозв'язних вершин. Нерозв'язними є тупикові вершини, І-вершини, у яких нерозв'язна хоча б одна дочірня вершина, і АБО-вершини, у яких нерозв'язна кожна дочірня вершина.

Алгоритм Ченга й Слейгла. Заснований на перетворенні довільного І/АБО-графа в спеціальний АБО-граф, кожна АБО-гілка якого має І-вершини тільки наприкінці. Перетворення використовує подання довільного І/АБО-графа як довільної формули логіки висловлень із подальшим перетворенням цієї довільної формули в диз'юнктивну нормальну форму. Подібне перетворення дозволяє далі використати алгоритм Харта, Нільсона й Рафаеля.

Метод ключових операторів. Нехай задане завдання $\langle A, B \rangle$ і відомо, що оператор f обов'язково повинен входити в рішення цього завдання. Такий оператор називається ключовим. Нехай для застосування f необхідний стан C , а результат його застосування є $I(c)$. Тоді І-вершина породжує три дочірні вершини, з яких середня є елементарним завданням. До завдань $\langle A, C \rangle$ й $\langle f(c), B \rangle$ також підбираються ключові оператори, і зазначена процедура редукування повторюється доти, поки це можливо. У підсумку

вихідне завдання $\langle A, B \rangle$ розбивається на впорядковану сукупність підзадач, кожна з яких вирішується методом планування в просторі станів.

Можливі альтернативи на вибір ключових операторів, так що в загальному випадку буде мати місце І/АБО-граф. У більшості завдань вдається не виділити ключовий оператор, а тільки вказати множину, яка його містить. У цьому випадку для завдання $\langle A, B \rangle$ обчислюється розходження між A й B , якому ставиться у відповідність оператор, що усуває це розходження. Останній й i є ключовим.

Метод редукції приводить до гарних результатів тому, що часто рішення завдань має ієрархічну структуру. Однак не обов'язково вимагати, щоб основне завдання й всі її підзадачі вирішувалися однаковими методами. Редукція корисна для подання глобальних аспектів завдання, а при вирішенні більш специфічних задач кращий метод планування за станами. Метод планування за станами можна розглядати як окремий випадок методу планування за допомогою редукцій, тому що кожне застосування оператора в просторі станів означає відомість вихідного завдання до двох більше простих, з яких одна є елементарною. У загальному випадку редукція вихідного завдання не зводиться до формування таких двох підзадач, з яких хоча б одна була елементарною.