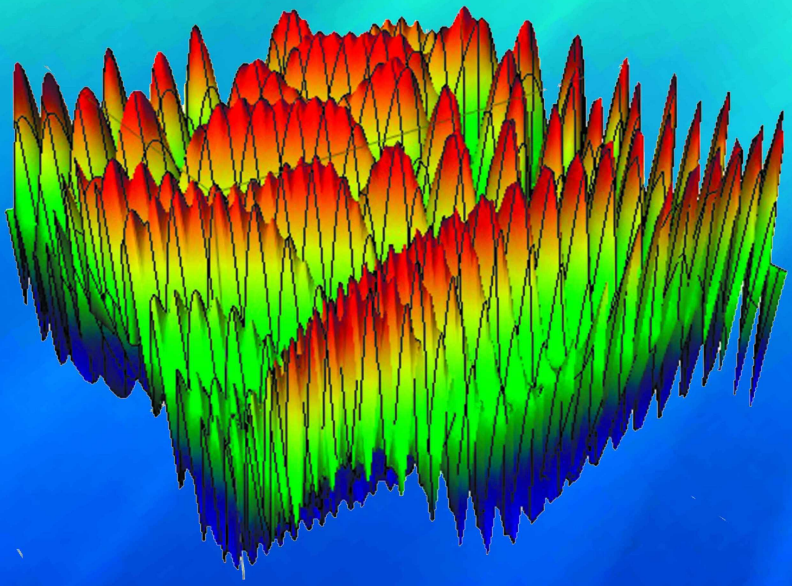
Л. Ф. ГУЛЯНИЦЬКИЙ
О. Ю. МУЛЕСА

ПРИКЛАДНІ МЕТОДИ КОМБІНАТОРНОЇ ОПТИМІЗАЦІЇ



МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
КИЇВСЬКИЙ НАЦІОНАЛЬНИЙ УНІВЕРСИТЕТ ІМЕНІ ТАРАСА ШЕВЧЕНКА

Л. Ф. ГУЛЯНИЦЬКИЙ
О. Ю. МУЛЕСА

ПРИКЛАДНІ МЕТОДИ КОМБІНАТОРНОЇ ОПТИМІЗАЦІЇ

Навчальний посібник

КИЇВСЬКИЙ УНІВЕРСИТЕТ 2016

УДК 519.16(075.8)
ББК 22.176я73
Г94

Рецензенти:
д-р техн. наук, проф. О. Ф. Волошин,
д-р фіз.-мат. наук П. І. Стецюк

Рекомендовано до друку вченою радою факультету кібернетики
(протокол № 5 від 30листопада 2015 року)

Ухвалено науково-методичною радою
Київського національного університету імені Тараса Шевченка
21 квітня 2016 року

Гуляницький Л. Ф.

Г94 Прикладні методи комбінаторної оптимізації : навч. посіб. / Л. Ф. Гуляницький, О. Ю. Мулеса. - К. : Видавничо-поліграфічний центр "Київський університет", 2016. – 142 с.

Розглянуто результати сучасних досліджень із розробки й упровадження прикладних методів комбінаторної оптимізації, питання формалізації, класифікації та оцінювання обчислювальної складності задач комбінаторної оптимізації, сучасні підходи до розв'язування зазначених задач. Основну увагу приділено метаевристичним методам.

Для студентів вищих навчальних закладів, магістрів, аспірантів, викладачів, а також фахівців, що займаються дослідженням і розв'язуванням прикладних задач комбінаторної оптимізації.

УДК 519.16(075.8)
ББК 22.176я73

ПЕРЕДМОВА

Стрімке розширення сфер застосування методів математичного моделювання й оптимізації, що відбувається в останні роки, формує потребу в розв'язуванні нових класів задач підвищеної складності та розмірності. Це зумовлює необхідність подальшого розвитку методів комбінаторної оптимізації та розробки нових математичних моделей оптимізаційних проблем, які можуть використовуватися в сучасних інформаційних технологіях. Показово, що за словами *combinatorial optimization*, *combinatorial optimization problem* у системі *Google Академія* на початку вересня 2015 р. було знайдено більше 3 млн наукових посилань.

У посібнику викладено результати сучасних досліджень із питань розробки й упровадження прикладних методів комбінаторної оптимізації. Розглянуто питання формалізації, класифікації та оцінювання обчислювальної складності задач комбінаторної оптимізації, сучасні підходи до їх розв'язування. Основну увагу приділено метаевристичним методам, які широко використовуються на практиці.

Теоретичний матеріал засновано на курсах лекцій, які читаються в Київському національному університеті імені Тараса Шевченка та НТУУ "Київський політехнічний інститут", і згруповано у два модулі. У розділах 1–4 висвітлюються питання розробки та аналізу наближених алгоритмів траєкторного типу, до яких належать алгоритми детермінованого та стохастичного локального пошуку. У розділах 5–10 вивчаються популяційні метаевристики, серед яких – еволюційні схеми (генетичні та міметичні алгоритми), метод деформацій, оптимізація мураши-

ними і бджолиними колоніями та інші алгоритми глобального пошуку, засновані на ройовому інтелекті.

Методи комбінаторної оптимізації можуть застосовуватися для розв'язування широкого кола прикладних проблем, що виникають у науці, техніці, біології, економіці, на виробництві тощо, тому автори сподіваються, що посібник буде корисним для фахівців у різних сферах застосування математичних методів оптимізації.

СПИСОК СКОРОЧЕНЬ

АІВ – алгоритм імітаційного відпалу
АКО – алгоритми комбінаторної оптимізації
БОК – багатопроцесорний обчислювальний комплекс
ГА – генетичний алгоритм
ЗБП – задачі (псевдо)булевого програмування
ЗК – задача комівояжера
ЗКО – задача комбінаторної оптимізації
ЗЛЦП – задачі лінійного цілочислового програмування
ЗНЛП – задачі неперервного лінійного програмування
КЗП – квадратична задача про призначення
КО – комбінаторна оптимізація
ЛП – локальний пошук
МА – міметичний алгоритм
МВС – метод вектора спаду
МДБ – метод деформованого багатогранника
МКД – мінімальне кістякове дерево
МС – мурашині системи
ОМК – оптимізація мурашиними колоніями
СЛП – стохастичний локальний пошук

1. МОДЕЛІ КОМБІНАТОРНОЇ ОПТИМІЗАЦІЇ

1.1. Загальна постановка і класифікація задач оптимізації

Довільну задачу оптимізації (не лише комбінаторної) у загальному випадку можна подати кортежем

$$< f\ X, \Pi, D, ext >,$$

де f: $X \to R^1$ – задана цільова функція задачі, R^1 – числова пряма, X – простір розв'язків задачі (простір пошуку), П – предикат, який визначає підмножину $D \subseteq X$ припустимих варіантів розв'язку згідно з наявними обмежувальними умовами, $ext \in \{\min,\ \max\}$ – напрям оптимізації.

У цих позначеннях задачу оптимізації можна переписати у вигляді: необхідно знайти $x_* \in D \subseteq X$ таке, що

$$x_* = \arg \underset{x \in D \subseteq X}{ext} f(x). \qquad (1.1)$$

Під простором розуміємо множину X, у якій вводимо певні співвідношення між елементами (метрики, топологію, сусідство тощо), а предикат П(x) визначає множину припустимих розв'язків:

$$\Pi(x) = \begin{cases} 0, & x \in D, \\ 1, & x \notin D. \end{cases}$$

Значно рідше зустрічаються задачі, у яких необхідно знайти всю множину розв'язків, що відповідають екстремальному значенню цільової функції, – ця множина позначається $Arg \underset{x \in D \subseteq X}{ext} f(x)$.

Варто зазначити, що вкрай рідко в оптимізаційних задачах вимагається знайти власне екстремальне значення цільової функції, тоді задача набуває вигляду:

$$\text{необхідно знайти } \underset{x\in D\subseteq X}{ext} f(x). \qquad (1.2)$$

Хоча (1.1) та (1.2) – дві різні задачі, історично склалося так, що в деяких застосуваннях (наприклад у соціально-економічних дослідженнях) задачу оптимізації записують у вигляді (1.2), але насправді розуміють як проблему пошуку саме аргументу екстремуму (1.1), а не лише відповідного значення цільової функції.

Часто також для опису задач оптимізації вживається вираз $f(x) \to ext$, а обмежувальні умови подаються у вигляді систем рівностей-нерівностей.

Усюди далі, якщо особливо не обумовлене протилежне, будемо для визначеності розглядати задачі на мінімум.

Цільова функція визначається аналітичним (чи іншим) способом задання та конкретним набором числових даних: $f(x) = f(x|c)$, де c – набір даних задачі (вхід задачі).

Означення 1.1. *Індивідуальною задачею оптимізації* називається пара $(f(x|c), X)$.

Тоді під *задачею оптимізації* розуміють сукупність усіх можливих індивідуальних задач.

Залежно від типу простору X розрізняють задачі неперервної, комбінаторної та змішаної оптимізації. Перший клас утворюють задачі, у яких простір X є неперервним (континуальним). Щодо другого класу, то тут існують різні підходи до означення як самого поняття ЗКО, так і його підкласів – власне комбінаторних задач, а також задач дискретного й цілочислового програмування. Детальніше класифікацію подано на рис. 1.1, де ЗНЛП – це задачі неперервного лінійного програмування, ЗБП – задачі (псевдо)булевого програмування, ЗЦЛП – задачі цілочислового лінійного програмування.

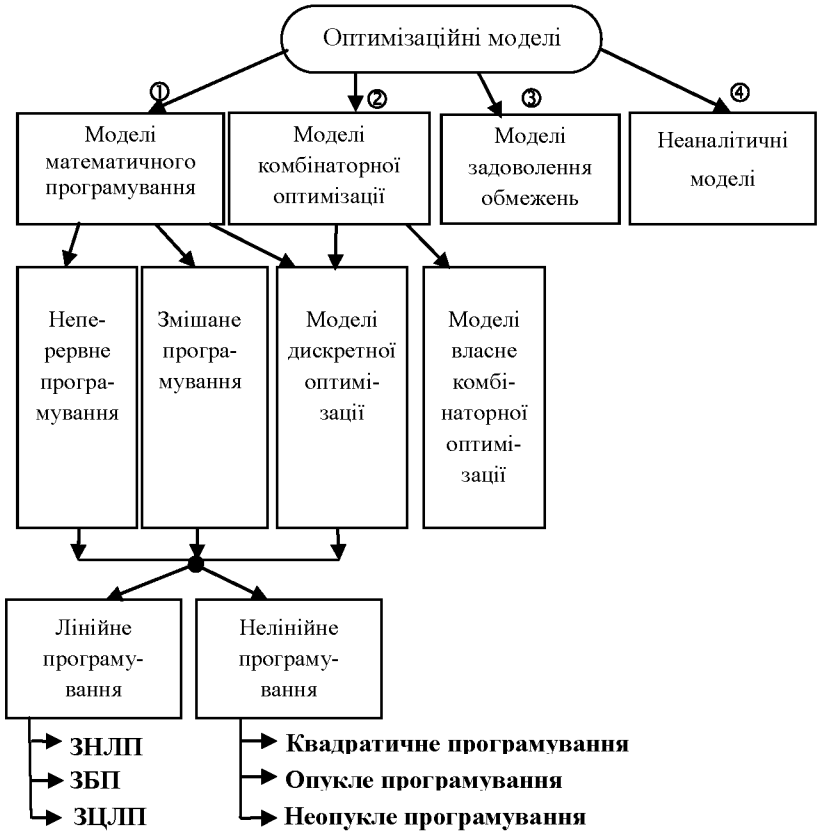


Рис. 1.1. Класифікація оптимізаційних моделей

Часто під *дискретною оптимізацією* розуміють задачу вигляду (1.1), у якій множина розв'язків $X = D_1 \times ... \times D_n$ – це підпростір n-вимірного евклідового простору, тобто $x = (x_1, ..., x_n)$, $x_i \in D_i$, $i = 1, ..., n$, причому хоча б одне із D_i є дискретним підпростором.

1.2. Формалізація задач комбінаторної оптимізації

Перш ніж перейти до формалізації ЗКО, уведемо низку понять. Далі потужність довільної множини X будемо позначати $\|X\|$, довільний окіл точки $x \in X$ позначатимемо $O(x)$, конкретно для метричних околів уживатимемо позначення $L(x)$ чи $L_\rho(x)$, де ρ – заданий радіус.

За досліджуваний простір візьмемо вибрану множину (назвемо її твірною), між елементами якої існують певні співвідношення. Нагадаємо, що булеаном 2^X довільної множини X називається множина всіх її підмножин.

Нехай на множині X тим чи іншим чином уведена система околів O, тобто для довільного $x \in X$ визначена сім'я множин $o^\sigma(x) \subseteq 2^X, \sigma \in I$, де I – множина індексів околів, причому $x \in o^\sigma(x)$ для всіх значень σ, а $O = \{o^\sigma(x) : x \in X, \sigma \in I\}$.

Означення 1.2. *Дискретний простір* – це такий простір, у якому твірна множина X є скінченною ($\|X\| < \infty$) або нескінченною без граничних точок.

Граничні точки ще називають *точками конденсації*, *дотику* або *накопичення*.

Найчастіше використовуються такі види околів: метричні, топологічні, алгоритмічні чи дескриптивні. Перші два види добре відомі в математиці. Утворення алгоритмічних околів описується певною процедурою, наприклад 2-заміни в ЗК. Дескриптивні визначаються шляхом задання в аналітичній чи описовій формі (наприклад шляхом переліку для кожного $x \in X$ елементів, які належать його околам).

Нехай тепер X – метричний простір, тобто на ньому введено метрику $d(x,y)$.

Означення 1.3. Під (замкненим) метричним околом із заданим радіусом $\rho > 0$ розуміють множину

$$L_\rho(x) = \{y \in X : d(x,y) \le \rho\}.$$

Для більшості дискретних просторів $\rho \in \{1,2,3,\ldots,diam\,X\}$, де діаметр простору X – це відстань між найвіддаленішими точками,

$$diam\,X = \max_{\forall x,y \in X} \{d(x,y)\}.$$

Приклад. Нехай $X \subseteq \{0,1\}^n$ – n-вимірний метричний простір векторів, компонентами яких є булеві змінні, $x = (x_1,\ldots,x_n) \in X$, $x_i \in \{0,1\}$, $i = 1,\ldots,n$, а $d(x,y) = \sum_{i=1}^{n} |x_i - y_i|$, $x, y \in X$ – функція відстані на X. Тоді метричний окіл з одиничним радіусом $L_1(x)$ визначається так: $L_1(x) = \{y \in X : d(x,y) \le 1\}$, тобто це множина булевих векторів, які відрізняються не більше ніж на одну компоненту.

На основі введених позначень означення 1.2 можна перефразувати.

Означення 1.4. Простір (X,O) називається *дискретним*, якщо

$$\forall x \in X \;\; \exists\, o^{\sigma}(x) \in O : o^{\sigma}(x) = \{x\}.$$

Таким чином, для кожної точки дискретного простору існує окіл, який складається лише із цієї точки – саме в такому разі точку називають *ізольованою*. Тому інколи використовують таке означення.

Означення 1.5. Простір X називається *дискретним*, якщо він складається лише із ізольованих точок.

Неважко показати еквівалентність усіх трьох наведених означень дискретного простору.

Нехай X – це простір з околами: (X, O), а множина $S \subseteq X$.

Варіант розв'язку $x_* \in S$ називається *субоптимальним розв'язком* задачі (1.1), якщо

$$f(x_*) \le f(y), \;\; \forall y \in S. \qquad (1.4)$$

Якщо S збігається з (метричним) околом точки x, тобто $S = o^{\sigma}(x)$, де $o^{\sigma}(x)$ – деякий окіл точки x, то такий субоптимальний розв'язок називається *локально оптимальним,* або *локальним, розв'язком* задачі (1.1).

Уведені означення дозволяють формально визначити поняття глобального екстремуму. Якщо умова (1.4) виконується для $S = X$, то точка x_* називається *глобальним,* або *точним, розв'язком* задачі (1.1).

Якщо система околів така, що будь-який локально оптимальний розв'язок задачі (1.1) глобальний, то кажуть, що така система околів є *точною.*

Часто під ЗКО розуміють проблему пошуку екстремумів заданої цільової функції вигляду (1.1), коли X – комбінаторний простір. Комбінаторним простором вважають сукупність комбінаторних об'єктів певного типу, утворених із елементів заданої скінченної множини (твірна множина), хоча при цьому формального означення не наводять.

Означення 1.6. *Базисними околами* довільної точки $x \in X$ будемо називати множину

$$B_x = \{o^\tau(x) \in O: \|o^\tau(x)\| > 1 \ \& \ \nexists \gamma: 1 < \|o^\gamma(x)\| < \|o^\tau(x)\|\}.$$

Отже, базисні околи – це такі нетривіальні (неодноточкові) околи точки, що мають найменшу потужність із усіх околів цієї точки.

Неважко бачити, що всі базисні околи однієї точки мають однакову потужність, а для деяких точок простору таких околів може й не існувати.

Означення 1.7. Простір X називається *локально скінченним* (у комбінаторному розумінні), якщо всі базисні околи його точок скінченні:

$$\forall x \in X, o^\tau(x) \in B_x \Rightarrow \| o^\tau(x) \| < \infty.$$

Означення 1.8. *Комбінаторним простором* назвемо дискретний локально скінченний у комбінаторному розумінні простір, який має не більш ніж зліченну кількість елементів.

Повертаючись до оптимізаційної задачі (1.1), дамо таке означення.

Означення 1.9. Задача (1.1) називається *ЗКО*, якщо простір її розв'язків X є комбінаторним.

1.3. Обчислювальна складність задач комбінаторної оптимізації

При дослідженні складності алгоритмів задач оптимізації останні розглядаються як *задачі розпізнавання* вигляду:

для даної індивідуальної ЗКО вигляду (1.1) *і цілого числа* M *визначити, чи існує такий припустимий розв'язок* $x \in D \subseteq X$, *що* $f(x) \le M$

Зрозуміло, що для задачі максимізації нерівність мала б вигляд $f(x) \ge M$.

Можна показати, що ця форма постановки задач має таку саму трудомісткість, як і відповідний оптимізаційний варіант.

Розрізняють два класи проблем: *P* та *NP*. Ці класи можна визначити точніше за допомогою будь-якого формального означення алгоритмів, такого, наприклад, як машина Тьюрінга. Не заглиблюючись у більш формальний опис, надалі нам буде достатньо означити їх так.

Клас *P* складають задачі, для розв'язування яких відомі алгоритми з поліноміальною складністю (polynomial-time algorithms). До поліноміальних відносять алгоритми, складність (трудомісткість) яких обмежена поліномом, степінь якого залежить від розміру входу ЗКО. Отже, це клас відносно простих задач, для яких існують ефективні алгоритми.

До **класу *NP*** (nondeterministic polynomial), який здається ширшим, належать задачі, що можуть бути розв'язані недетермінованим поліноміальним алгоритмом. Такі алгоритми мають дві фази: на першій знаходять припустимий варіант розв'язку, а на другій цей варіант перевіряється детермінованим поліноміальним алгоритмом верифікації.

Означення 1.10. ЗКО π у варіанті розпізнавання є *NP-повною* (*NP*-complete), якщо виконуються дві умови:

1) $\pi \in NP$;

2) усі задачі з *NP* можуть бути зведені до π за допомогою поліноміального алгоритму.

Зауважимо, що на практиці для верифікації п. 2 означення 1.10 зазвичай лише показують, що деяку відому *NP*-повну задачу можна поліноміально перетворити на досліджувану задачу.

Інколи можна показати, що всі задачі з *NP* поліноміально зводяться до деякої задачі π, але не вдається довести, що $\pi \in NP$. Такі задачі відносять до *NP*-складних.

Означення 1.11. Задача належить до класу *NP-складних* (*NP*-hard), якщо до неї поліноміально зводяться всі задачі з класу *NP*.

Для потреб практики важливо, щоб трудомісткість алгоритмів була обмежена поліномом від вхідних даних задачі. Нині предметом дискусій є питання, чи $P = NP$. У разі негативної відповіді теоретично не існує поліноміального алгоритму, тому при розв'язуванні таких задач одразу слід зосередитися на розробці потужних наближених алгоритмів, які, хоча й не гарантують отримання оптимального розв'язку, здатні знаходити оптимальний чи близький до нього розв'язок за прийнятний час. Зауважимо, що ця проблема Математичним інститутом Клея з Кембриджа у 2000 р. визначена як одна із семи проблем тисячоліття й за її розв'язання обіцяна премія у 1000 дол. США. Утім, згідно з опитуванням, яке було проведене серед 100 вчених у 2002 р., 61 дослідник вважає, що ці класи не збігаються, 9 – що збігаються, 22 вагалися з відповіддю, а 8 вважають, що гіпотеза не виводиться з наявної системи аксіом і, таким чином, не може бути доведена чи спростована.

Насамкінець зазначимо, що викладене справедливе не лише для ЗКО, які розглядалися у формі розпізнавання, а й для низки інших проблем, що формулюються відразу як задачі розпізнавання (наприклад задача виконуваності булевої форми чи задача про гамільтонів цикл).

1.4. Приклади моделей комбінаторної оптимізації

1) Задача комівояжера (ЗК). Дано множину n міст, відстані між якими відомі. Необхідно знайти оптимальний маршрут, що проходить через задані міста рівно по одному разу.

Математична постановка задачі є такою: дано граф $G=(V,E)$, де V – множина вершин, які ототожнюються з містами, E – множина ребер, d_{ij} – вага ребра $e_{ij}\in E$. Необхідно знайти гамільтонів цикл мінімальної ваги, тобто замкнений цикл у графі, що містить усі вершини та передбачає відвідування кожної вершини лише один раз.

Нехай $p=(p_1,\ldots,p_n)$ – довільна перестановка всіх n елементів множини $\{1,\ldots,n\}$; будемо вважати, що елемент p_i, $i=1,\ldots,n$, задає номер міста, яке відвідується на i-му етапі, а $p_{n+1}=p_1$.

У комбінаторній постановці шукаємо таку перестановку, для якої досягається

$$\min\sum_{i=1}^{n} d_{p_i p_{i+1}}.$$

2) Задача (псевдо)булевого лінійного програмування. Необхідно знайти такий булевий вектор, для якого досягається

$$\max\sum_{i=1}^{n} c_i x_i$$

за обмежень

$$\sum_{j=1}^{n} a_{ij} x_j \le b_i,$$

де $x=(x_1,\ldots,x_n)$, $x_i\in\{0,1\}$, – булевий вектор довжиною n, а $c=(c_1,\ldots c_n)$, $b=(b_1,\ldots b_n)$, $(a_{ij})_{n\times n}$ – задані числові вектори та матриця.

Строго кажучи, це є задача псевдобулевої оптимізації, оскільки цільова функція не булева, але історично вона дістала назву *задачі булевого програмування*.

3) Лінійна задача про призначення. Дано n виконавців та n робіт. Будь-який виконавець може бути призначений до виконання будь-якої роботи. Виконання роботи пов'язане з витратами, які залежать від того, який виконавець виконує роботу. Необхідно виконати всі роботи, призначивши для кожної лише одного виконавця, тобто знайти таке призначення робіт для виконавців, щоб загальні витати були мінімальними.

Математична постановка задачі: знайти призначення, яке оптимізує задану цільову функцію з урахуванням обмежень:

$$\sum_{i=1}^{n}\sum_{j=1}^{n} c_{ij}x_{ij} \to \min,$$

$$\sum_{i=1}^{n} x_{ij} = 1,\ j = 1,...,n,$$

$$\sum_{j=1}^{n} x_{ij} = 1,\ i = 1,...,n,$$

де c_{ij} – вартість призначення i-го виконавця на j-ту роботу $(i, j = 1,...,n)$,

$$x_{ij} = \begin{cases} 1, \text{ якщо виконавець } i \text{ призначається на роботу } j, \\ 0 \text{ в іншому разі.} \end{cases}$$

Матрицю, що задовольняє умови (1.5), називають *перестановочною*.

Нехай $p = (p_1,...,p_n)$ – довільна перестановка всіх n елементів множини $\{1, ..., n\}$, $C = (c_{ij})_{n\times n}$. Будемо вважати, що значення i-го елемента перестановки p_i містить номер роботи, на яку призначений i-й виконавець, тобто якщо $p_i = 3$, то це означає, що i-го виконавця призначено на третю роботу.

Тоді в термінах перестановок цільова функція матиме вигляд: знайти таку перестановку p_*, щоб

$$f(p) = \sum_{i=1}^{n} c_{ip_i} \to \min .$$

Перестановка дозволяє описувати всі можливі варіанти розв'язків цієї задачі, тому при її використанні немає явних обмежень.

4) Квадратична задача про призначення. Нехай дано множину з n об'єктів і множину з n їх місць призначення. Відомі відстані між місцями призначення та інтенсивність потоків ресурсів між об'єктами. Задача полягає в розміщенні всіх об'єктів

у різних місцях призначення таким чином, щоб сума відстаней, помножених на відповідні потоки, була мінімальною.

Математична постановка задачі є такою. Задано матрицю $C=(c_{ij})_{n\times n}$, $c_{ij}\geq 0$ – інтенсивність потоків між об'єктами (виконавцями, що призначаються), $D=(d_{st})_{n\times n}$, $d_{st}\geq 0$ – відстань між місцями призначення (роботами). Необхідно розмістити n об'єктів на n місцях так, щоб виконувалися умови:

$$\sum_{i=1}^{n}\sum_{j=1}^{n}\sum_{s=1}^{n}\sum_{t=1}^{n}c_{ij}d_{st}x_{is}x_{jt}\to\min,$$

$$\sum_{s=1}^{n}x_{is}=\sum_{t=1}^{n}x_{jt}=1,\ \ i,j=1,...,n,$$

$$x_{kl}=\begin{cases}1, & \text{якщо } k-\text{й об'єкт розміщений на } l-\text{му місці призначення;}\\ 0 & \text{у протилежному випадку.}\end{cases}$$

У термінах перестановок цільова функція надає задачі вигляду

$$f(p)=\sum_{i=1}^{n}\sum_{j=1}^{n}c_{ij}d_{p_ip_j}\to\min,$$

де $p=(p_1,...,p_n)$ – перестановка з n елементів над множиною $\{1,...,n\}$ така, що p_i містить номер місця призначення, на якому розміщений i -й об'єкт.

У різних постановках квадратичної задачі про призначення можливі обмежувальні умови: наприклад, зафіксовані чи заборонені позиції деяких елементів.

5) Мінімальне кістякове дерево (МКД). Нагадаємо, що кістяковим деревом для заданого графа G називається його ациклічний підграф, який містить усі вершини графа G. Задача про МКД полягає в пошуку такого кістякового дерева, щоб сума ваг ребер, які входять у дерево, була мінімальною.

Математична постановка задачі така: для заданої множини вершин V, ребер E та їх ваг $D=\left(d_{ij}\right)$, $(i,j)\in E$, необхідно побу-

дувати кістякове дерево з множиною вершин V та підмножиною ребер $U \subseteq E$, яке б мінімізувало сумарну вагу.

Цільова функція має вигляд

$$\sum_{(i,j)\in U} d_{ij} \to \min .$$

Додатковою умовою тут може бути те, що локальний степінь вершини не має перевищувати деяке задане число.

Більшість ЗКО – це задачі на скінченних множинах. Однак навіть у цьому разі, не кажучи вже про пошук розв'язку в нескінченних просторах, виникають проблеми із застосуванням точних методів, що й визначає актуальність розробки наближених алгоритмів комбінаторної оптимізації.

2. КЛАСИФІКАЦІЯ МЕТОДІВ КОМБІНАТОРНОЇ ОПТИМІЗАЦІЇ

2.1. Класифікація алгоритмів комбінаторної оптимізації

Класифікація алгоритмів комбінаторної оптимізації (АКО) за отримуваним розв'язком. Для простоти викладу поки вважатимемо, що $D \equiv X$, тобто розглянемо ЗКО без обмежень: необхідно знайти елемент $x_* \in X$ такий, що

$$x_* = \arg \underset{x \in X}{ext} f(x). \qquad (2.1)$$

Будемо вважати, що АКО – певна процедура A, яка переводить задану підмножину $Z \subset X$ (початкові наближення) у множину $X_* \subset X$:

$$AZ = X_*,$$

тобто X_* – це множина знайдених розв'язків задачі (2.1).

Існує багато підходів до класифікації АКО – за точністю, типом використаних просторів, структурою обчислювальної схеми тощо (рис. 2.1).

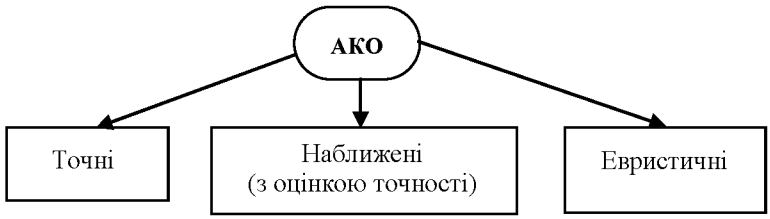


Рис. 2.1. Класифікація АКО за точністю

Точні АКО – це такі методи, які знаходять глобальний розв'язок, тобто для них $X_* \subseteq Arg\,ext$.

Наближені АКО діляться на алгоритми з *апріорною* та *апостеріорною оцінками точності*.

Евристичні алгоритми будуються на основі правдоподібних міркувань (наприклад, алгоритм "іди в найближче місто", що використовується для розв'язання ЗК), але не в змозі оцінити точність знайденого розв'язку.

Часто дослідники називають наближеними як алгоритми з оцінкою точності, так і евристичні; ми теж далі будемо розглядати такий об'єднаний клас прикладних алгоритмів.

Класифікація АКО за типом обчислювальної схеми. Наближені алгоритми за типом обчислювальної схеми прийнято ділити на конструктивні та ітераційні.

Нехай маємо певну множину $Y \supseteq X$.

Конструктивні алгоритми (інші назви – *прямі*, *послідовні*) – це такі алгоритми, у яких $Y \supset X$ ($Y \neq X$), тобто вони оперують у просторі, що є розширенням простору розв'язків X.

Починаючи "з нуля" чи якогось фрагмента розв'язку, вони поступово формують повний розв'язок.

Приклади конструктивних алгоритмів для різних ЗКО:

1) ЗК – алгоритм "іди в найближче місто", який на кожному наступному кроці додає до маршруту вершину, відстань до якої є найменшою з усіх можливих.

2) КЗП – евристика, відповідно до якої об'єкти (елементи) з інтенсивнішими потоками розміщуються в першу чергу.

3) МКД – евристика, відповідно до якої на кожному наступному кроці в побудований фрагмент кістякового дерева включається та вершина, яка мінімально збільшує його сумарну довжину й не утворює підцикл.

Ітераційні алгоритми – це такі алгоритми, які на кожному кроці опрацьовують "повні" розв'язки, тобто для них простір пошуку $Y \equiv X$.

Починаючи з деякого $x^0 \in X$, ітераційні алгоритми намагаються його поліпшувати покроково:

$$x^{(h+1)} = A^{(h)} x^{(h)},\ h = 0,1,\ldots,$$

де $A^{(h)}$ – ітераційна процедура, яка у більшості випадків не залежить від кроку h, тобто $A^{(h)} = A$.

Ітераційні методи, які оперують на кожному кроці одним (поточним) розв'язком, називаються *траєкторними*; інколи в зарубіжній літературі такі методи називаються *базованими на одному розв'язку чи стані* (Single-Solution Based/Single-State Methods), а під траєкторними розуміють такий їх підклас, який породжує послідовність сусідніх розв'язків – траєкторію у просторі пошуку. Уникаючи певних термінологічних ускладнень, будемо всі такі методи називати траєкторними.

Алгоритми, які опрацьовують на кожній ітерації не один, а кілька розв'язків одночасно, називаються *популяційними* (Population-Based Methods).

Отже, для траєкторних алгоритмів $||Z|| = || X_* || = 1$, а для популяційних – $||Z|| > 1$, $|| X_* || > 1$.

За *складністю структури* АКО можна виділити:

- прості алгоритми;
- комбіновані алгоритми;
- метаевристики;
- гібридні метаевристики;
- гіперевристики.

Комбіновані алгоритми утворюються шляхом послідовного застосування двох чи більше ітераційних алгоритмів з передаванням розв'язків від одного до іншого. У метаевристиках, про що піде далі, здійснюється вкладення одного алгоритму/процедури в іншу стратегію.

Гіперевристикою (*гіперевристичним алгоритмом*) називають метод пошуку, орієнтований на автоматизацію процесів вибору, комбінування або адаптації чи налаштування кількох простіших алгоритмів (евристик або метаевристик) для ефективного розв'язання ЗКО чи їх класів. Це може досягатися як вибором наявних евристик чи їх фрагментів, так і генеруванням нових.

Таким чином, якщо метаевристики та інші алгоритми здійснюють переважно пошук у просторі розв'язків ЗКО, то простором пошуку для гіперевристик є множина евристик (простіших алгоритмів чи їх частин).

За *впливом на ландшафт* пошуку більшість АКО можна віднести до таких, що залишають його незмінним. Проте є алгоритми, які модифікують цей ландшафт шляхом:

- зміни простору розв'язків (наприклад, послідовні алгоритми);
- зміни цільової чи оцінкової функції (алгоритми керованого локального пошуку);
- варіації системи околів, що використовується при пошуку (алгоритми локального пошуку (ЛП) зі змінними околами, метод вектора спаду з пульсуючими околами).

Якщо робота алгоритму базується на безпосередніх даних ЗКО, то такі АКО належать до *задаче-орієнтованих алгоритмів.*

У деяких нових АКО використовуються не стільки прямі дані ЗКО, скільки спеціальна модель задачі, що розв'язується (наприклад феромонна матриця та матриця маршрутів у ОМК), – такі алгоритми отримали назву *моделе-орієнтованих*.

Точні алгоритми можна поділити на загальні методи та спеціальні алгоритми.

Загальні методи:

- повний перебір (вичерпний пошук);
- метод гілок і меж (МГіМ);
- метод гілок і відтинань;
- послідовний аналіз варіантів (ПАВ, "київський вінник");
- динамічне програмування (метод Беллмана).

Спеціальні алгоритми будуються на основі врахування специфіки конкретної задачі оптимізації, що розв'язується, тому мають вузький спектр застосування.

Приклад – метод Балаша (угорський метод) для розв'язання лінійної задачі про призначення. Один з небагатьох прикладів поліноміального алгоритму для розв'язання ЗКО. Однак при додаванні однієї чи кількох обмежувальних умов виникають подібні задачі, але зазначений алгоритм уже не може бути застосованим чи його модифікація уже не має поліноміальної складності.

2.2. Про найуживаніші наближені алгоритми

Необхідність розробки ефективних наближених АКО, які застосовуються у переважній більшості випадків на практиці, визначається низкою обставин:

1) практично всі важливі задачі належать до *NP-складних*, тож точне їх розв'язання дуже проблематичне навіть із використанням сучасних і перспективних комп'ютерів;

2) їх цільові функції мають зазвичай велику кількість *локальних екстремумів*;

3) у багатьох прикладних проблемах *дані задаються з певними похибками*, що робить недоцільними ті істотні обчислювальні затрати, які необхідні для знаходження їх точного розв'язку;

4) покладені в основу розробки наближених обчислювальних схем ідеї (метаевристики) дозволяють створювати алгоритми, які можуть розв'язувати *не одну*, *а цілий клас* близьких за постановкою оптимізаційних задач;

5) важливий клас оптимізаційних задач породжується проблемами з директивним терміном, тобто їх розв'язок має бути знайдений до зазначеного апріорі строку;

6) у деяких задачах значення цільової функції можуть бути доступними лише в процесі розв'язання задачі або змінюватися з часом – цей клас утворюють *динамічні*, або *on-line задачі*.

Найуживаніші на практиці наближені алгоритми можна поділити на сім класів (рис. 2.2). Тут МГіМ – це метод гілок і меж, ПАВ – послідовний аналіз варіантів; ці точні алгоритми використовуються в даному контексті для породження наближених обчислювальних схем.

Зрозуміло, що сукупність усіх розроблених донині АКО значно перевищує перелік наведених на цьому рисунку. Проте, як свідчить аналіз наукових публікацій, саме зазначені алгоритми та їх модифікації є основним інструментом для розв'язання практичних ЗКО.

Левову частку алгоритмів із класів 2 – 7 становлять ітераційні методи, багато з яких базуються на використанні процедур локального пошуку. Крім того, для розв'язання задач із підвищеною точністю значного поширення набули метаевристики – саме таким алгоритмам і буде приділено основну увагу надалі.

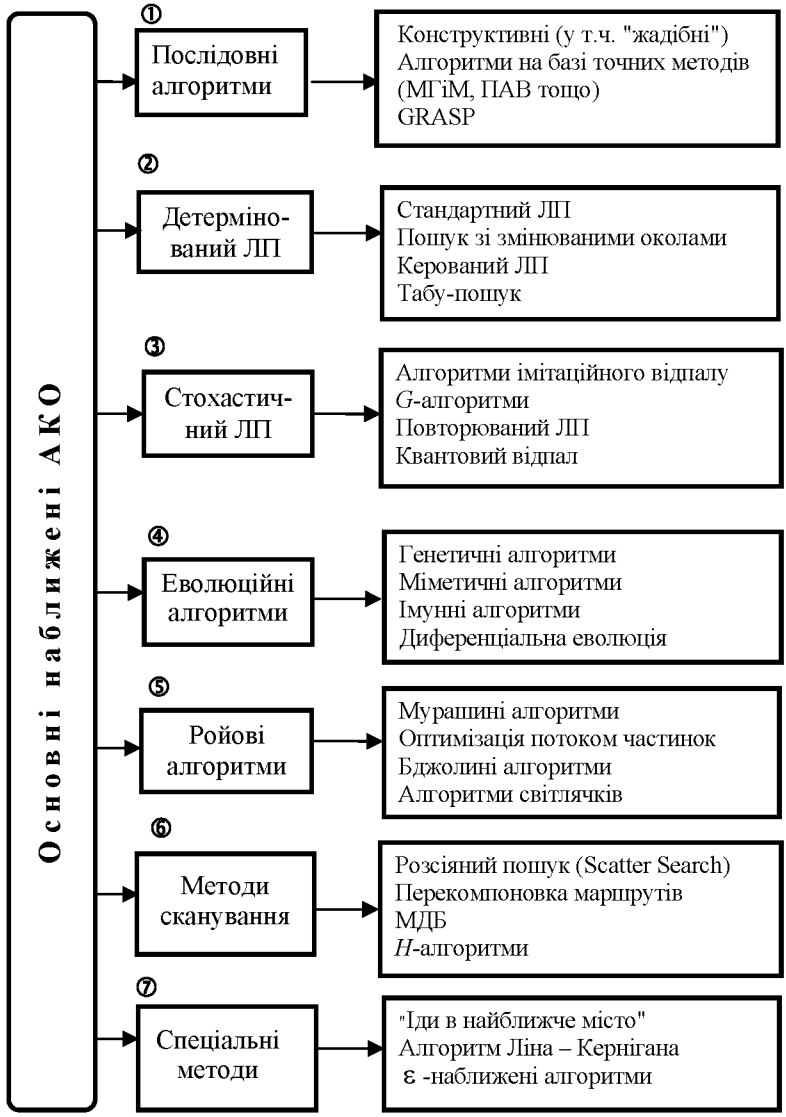


Рис. 2.2. Класифікація основних наближених методів комбінаторної оптимізації

2.3. Конструктивні алгоритми

Конструктивні алгоритми характеризуються простотою обчислювальної схеми та низькою трудомісткістю, а значить, і високою швидкістю роботи, платою за що є незначна точність отримуваних розв'язків. Це визначає основні напрями їх використання: як процедур знаходження початкових наближень для алгоритмів ітераційного типу чи для розв'язання задач великої розмірності.

Основна ідея конструктивних алгоритмів – побудова припустимого варіанта розв'язку задачі шляхом нарощування наявного в поточний момент його *часткового фрагмента*.

Формальніше, при застосуванні конструктивних алгоритмів до розв'язання задач вигляду (1.1):

➢ Замість простору варіантів X розглядається ширший простір $Y \supset X$, для якого X у більшості випадків є межею. Наприклад, у ЗКО на перестановках часто Y – це множина всіх розміщень k елементів множини A, $k \le n$.

➢ Для кожного елемента $y \in Y$ вводиться поняття множини сусідніх елементів – певний аналог поняття околу в просторі Y.

➢ Визначається функція φ: $Y \to R^1$, яка оцінює якість $\varphi(y)$ кожного варіанта $y \in Y$ з погляду можливого значення цільової функції задачі (1.1) чи (1.2), причому природною є умова:

$$\varphi(x) = f(x), \ \forall x \in X.$$

Означення 2.1. Для довільного елемента $y \in Y \setminus X$ множиною сусідніх елементів є множина

$$N(y) = \{x \in Y: \ \|x\| - \|y\| = 1\}.$$

Загальна схема конструктивних алгоритмів складається із таких кроків (для простоти викладення розглядатимемо задачу безумовної оптимізації):

1) *Початок.* Випадковим чином чи з певних міркувань, які враховують специфіку задачі, вибираємо один елемент множини A і включаємо його у фрагмент розв'язку $y \in Y$, тобто $\|y\| = 1$.

Покладаємо $Z = \{y\}$.

2) *Основна процедура.*

2.1) Будуємо підмножину елементів $z \in A \setminus Z$, що є сусідніми до всіх чи деяких $y \in Z$, і обчислюємо для них значення функції φ.

2.2) Серед усіх побудованих сусідніх вибираємо один чи кілька елементів, якім відповідають кращі значення функції $\varphi(z)$.

2.3) Включаємо вибрані елементи в множину Z.

3) *Перевірка критерію завершення.*

Якщо умова завершення не виконується, то знову виконується основна процедура (п. 2), інакше алгоритм припиняє роботу.

Критерієм завершення найчастіше стає умова, що серед елементів множини Z хоча б один елемент (або всі ці елементи) належить простору X. Зрідка викладена схема розширюється шляхом старту в п. 1 не з одного елемента, а відразу з кількох.

Наведена схема охоплює різноманітні евристичні алгоритми, що відрізняються правилами відбору елементів у п. 2.2, заданням функції φ та визначенням критерію зупинення. Конструктивні алгоритми можуть будуватися на основі таких відомих послідовних схем, як послідовний аналіз і відсіювання варіантів чи динамічне програмування.

Найбільшого поширення серед конструктивних алгоритмів набули *жадібні* алгоритми (*greedy*), у схемі яких у п. 2.2 обов'язково вибирається елемент, що доставляє екстремальне (максимальне) значення функції $\varphi(z)$.

2.4. Метаевристики

Серед наближених оптимізаційних методів розв'язання ЗКО окремий клас утворюють метаевристики. За своєю природою метаевристики об'єднують простіші алгоритми чи техніки в межах обчислювальних схем вищого рівня, які спрямовані на ефективне вивчення простору пошуку. Точніше, *метаевристика* – це метод розв'язання широкого класу обчислювальних задач шляхом такого комбінування існуючих процедур, при якому одна є провідною, а інша (чи кілька) – підлеглою (підлеглими). Як провідними, так і підлеглими процедурами часто стають ві-

домі евристики чи алгоритми, зазвичай рандомізовані. Якщо складовою метаевристики є певний математичний метод, то вживають термін *матевристика*.

Можна виділити два типи метаевристик: ті, у яких як провідна, так і підлегла процедури здійснюють пошук у просторі розв'язків вихідної ЗКО (задаче-однорідні), і ті, у яких ці процедури розв'язують різні підзадачі. Прикладом останніх метаевристик є гібридні декомпозиційні алгоритми для розв'язання ЗК, які можуть комбінувати розв'язання задачі розбиття множини *міст* на кластери з процедурами локального пошуку при агрегуванні та поліпшенні розв'язків вихідної ЗК.

Найчастіше метаевристичні алгоритми застосовують при розв'язанні ЗКО, проте вони також можуть бути використані для задач, що зводяться до розв'язання логічних рівнянь.

Серед еволюційних методів, окрім генетичних алгоритмів, які призначені для розв'язання ЗКО, визначених у бінарних просторах і комбінаторних просторах зі складнішими конфігураціями (наприклад у просторі перестановок), виділяють ще метаевристики на основі штучних імунних систем і методи диференціальної еволюції.

Деякі алгоритми базуються на аналогіях із процесами у фізичних системах; серед них найбільшого поширення на практиці набули алгоритми імітаційного відпалу (Simulated Annealing), квантового відпалу, метод гармонійного пошуку (Harmony Search).

Пошук концепцій для розробки алгоритмів комбінаторної оптимізації зумовив появу в останній час нових класів алгоритмів, які навіяні природою. Помітне місце серед таких алгоритмів займають методи ройового інтелекту. Найпоширеніші серед таких обчислювальних схем – оптимізації мурашиними колоніями (Ant Colony Optimization) та потоком частинок (Particle Swarm Optimization), бджолині алгоритми й методи штучних бджолиних колоній (Bees Algorithms, Artificial Bee Colony), метод імітації поведінки бактерій (Bacterial Foraging Optimization), пошук на основі імітації поведінки зграї риб у пошуках корму (Fish School Search), алгоритми, які імітують поведінку летючих мишей (Bat-Inspired Algorithm), світлячків (Glowwarm Swarm Optimization) і жаб (Shuffled Frog-Leaping Algorithm).

Останнім часом розроблені також інші алгоритми (наприклад алгоритм розумних крапель, Intelligent Water Drops), але нинішній початковий етап їх дослідження й апробації не дозволяє поки що робити обґрунтованих висновків щодо практичної ефективності.

Як уже зазначалося, серед прикладних методів розв'язання ЗКО виокремлюється клас моделе-орієнтованих алгоритмів. Альтернативні задаче-орієновані алгоритми при формуванні нових варіантів розв'язків опираються безпосередньо на дані ЗКО та значення своїх параметрів. У моделе-орієнтованих алгоритмах варіанти розв'язків генеруються з використанням спеціальної параметризованої ймовірнісної моделі, яка оновлюється в процесі обчислень так, щоб пошук концентрувався в областях розв'язків "високої якості". У цих алгоритмах здійснюється оптимізація параметрів і/чи структури моделі задачі, тобто замість вихідної ЗКО розв'язується задача оптимізації в просторі параметрів моделі, яка використовується таким алгоритмом. Прикладами розглядуваних алгоритмів є ОМК чи метод обчислення оцінок розподілу.

Використання моделе-орієнтованих алгоритмів для побудови гібридних метаевристик останнім часом створило новий перспективний тип алгоритмів – *кооперативні метаевристики*, у яких здійснюється обмін не лише варіантами розв'язків, а й власне параметризованими моделями ЗКО, що дозволяє підвищити ступінь диверсифікації пошуку.

Аналіз стратегій агрегування метаевристик дозволяє виділити низку ключових аспектів їх розробки (див. табл. 2.1, де МЕ – метаевристики).

Отже, можна виділити такий тренд розвитку алгоритмів розв'язання ЗКО: окремі алгоритми та їх комбінації → метаевристики → гібридні (зокрема кооперативні) метаевристики → гіперевристики.

Утім цю послідовність не слід розуміти буквально: у конкретному випадку специфіка чи розмірність ЗКО, яка розв'язується, можуть вимагати застосування будь-яких алгоритмів із наведених типів.

Таблиця 2.1

Стратегії розробки метаевристик

Ключові аспекти	**Принципи агрегування**	**Приклади, коментарі**
Що гібридизується	МЕ+МЕ	ГА+ОМК ГА+*H*-алгоритми, МА
	МЕ+спеціальні алгоритми	ГА+алгоритм Ліна – Кернігана
	МЕ+математичні методи	Матевристики
Рівень агрегування	Комбіновані алгоритми	Мінімальні зміни в обчислювальних схемах базових алгоритмів. Незалежний пошук розв'язків та їх конвеєрне передавання
	Гібридні МЕ	Включення (вкладання) обчислювальних схем в ієрархічну структуру
	Кооперативні МЕ (гомогенні чи гетерогенні)	Обмін не лише розв'язками, а й моделями ЗКО. Складові МЕ здійснюють пошук у всьому просторі X чи його підпросторах. Приклади – острівні ОМК, кооперативні алгоритми ОМК+*H*-алгоритм
	Гіпереврис-тики	Орієнтовані на синтез МЕ
Базові засади обчислювальних схем	Математичні засади	*H*-алгоритми, розсіяний пошук, переключення шляхів, повторюваний ЛП
	Аналогії з фізико-технічних процесів	АІВ, алгоритм феєрверків, алгоритм інтелектуальних крапель
	Аналогії з природи	ГА, імунні алгоритми, диференціальна та культурна еволюція
	Аналогії з біології	ОМК, бджолині алгоритми, алгоритми поведінки бактерій, павуків, риб, птахів

3. ДЕТЕРМІНОВАНИЙ ЛОКАЛЬНИЙ ПОШУК

3.1. Загальна схема алгоритмів

Алгоритми детермінованого локального пошуку – це сім'я ітераційних методів, заснована на частковому перебиранні варіантів на кожній ітерації серед точок околу поточної точки, тобто серед сусідніх до неї.

В алгоритмах цього типу замість повного перебору застосовується спрямований локальний перебір у підмножинах варіантів, які називаються околами. Цим пояснюється їх назва – *алгоритми локального пошуку* (Local search). У сфері КО алгоритми ЛП мають давню історію через свою наочність і високу ефективність. Наприклад, перший алгоритм локального пошуку для задачі комівояжера був запропонований ще в 1956 р., а локальний пошук для задачі розміщення обладнання був розроблений у 1962 р. Загальна схема ЛП у задачах мінімізації така: починаючи з деякого припустимого розв'язку задачі, новий розв'язок із кращим значенням цільової функції шукають у його околі. Якщо такий розв'язок знайдено, то він приймається і пошук поліпшення розв'язку далі здійснюється вже в його околі і т. д. Алгоритм закінчується, коли досягнуто локального оптимуму, тобто коли в околі поточного розв'язку немає ніякого іншого варіанта з меншим значенням цільової функції.

Загальна схема алгоритмів детермінованого локального пошуку може бути подана таким чином:

1. Генерація початкового припустимого розв'язку x, який обираємо як поточний варіант.

2. Чергова ітерація.

Формуємо окіл $O(x)$ поточного варіанта й точно чи наближено знаходимо елемент $y \in O(x)$, який є субоптимальним розв'язком у цьому околі.

Якщо $y \neq x$, то знайдений елемент оголошується черговим поточним варіантом (здійснюється переприсвоєння $x \leftarrow y$) і починається чергова ітерація, інакше – повернення на п. 3.

3. Завершення роботи алгоритму: x – локальний розв'язок, якщо на останній ітерації здійснюється вичерпний пошук в околі.

Отже, у результаті роботи алгоритму локального пошуку при заданому початковому наближенні x^0 утворюється послідовність точок $x^1,...,x^m$, яка має такі характеристики:

a) розмірність m послідовності є невідомою заздалегідь;

b) $x^{h+1} \in O\left(x^h\right), \forall h \in [0, m-1]$;

c) $f(x^{h+1}) < f(x^h), \forall h \in [0, m-1]$;

d) x^m є локальним розв'язком із зони притягання точки x^0: $f\left(x^m\right) < f(y), \forall y \in O\left(x^m\right)$.

Під зоною притягання розуміємо таку частину простору X, у якій цільова функціє є унімодальною.

Таким чином, локальний пошук подібний простому алгоритму спуску з тією відмінністю, що (a) околи поточного розв'язку досліджуються систематично замість безладного блукання, і (b) пошук в околі повторюється до тих пір, поки не буде знайдений локально оптимальний розв'язок.

3.2. Ключові аспекти реалізації

Принциповими моментами реалізації конкретних алгоритмів локального пошуку є:

1) визначення околів $O(x)$;

2) генерація чергової точки $y \in O(x)$;

3) критерій завершення перегляду точок у поточному околі та переходу до наступного;

4) спосіб обчислення величини зміни цільової функції при переході до нового поточного варіанта;

5) критерій завершення;

6) формування початкового наближення.

Ефективність алгоритмів локального пошуку істотно залежить від вибору відповідного типу околу $O(x)$. Чим більше окіл, тим імовірніше отримати кращий результат, але розширення околів швидко стає непрактичним. Утім для багатьох задач КО алгоритми з околами розміром більше, ніж $O(n^2)$, де n позначає розмірність задачі, стають неефективними й тому досить рідко використовуються на практиці.

Найчастіше за все використовуються метричні околи чи околи, утворені алгоритмічно (як, наприклад, у алгоритмі 2-замін Ліна).

Можливі такі стратегії переходу в околі:

- до першого поліпшення за значенням цільової функції;
- до найкращого (перебирається весь окіл і відбирається найкраща точка).

Отже, пошук в околі може здійснюватися або шляхом повного перебору сусідніх точок (і тоді знайдене поліпшення – це найкращий в околі варіант), або ж перехід здійснюється до першого знайденого варіанта, який поліпшує цільову функцію. Накопичений багаторічний досвід застосування алгоритмів локального пошуку свідчить, що повний перегляд околу на кожній ітерації потребує значних затрат часу, хоча й не гарантує знаходження в підсумку точнішого розв'язку, тому ефективнішим є пошук в околі до першого поліпшення. У деяких модифікаціях методів локального пошуку використовуються околи зі змінюваним радіусом.

Іншим аспектом, який суттєво впливає на ефективність алгоритмів локального пошуку, є організація перебору точок в околах. Можливі два способи організації перебору точок в околі при переході від ітерації до ітерації:

а) *лінійний генератор* – при зміні поточного варіанта в новому околі починається породження елементів околу з початку;

б) *кільцевий генератор* – точки в околі вважаються певним чином упорядкованими й після переходу до нової ітерації про-

довжується перебір точок, починаючи з наступної відповідно до зазначеного упорядкування.

В обох випадках явно чи програмно визначається упорядкування операторів зсуву, які породжують точки в околі, і послідовно формуються такі точки. Однак у першому випадку після переходу до кращого варіанта перебір у новому околі починається з початкового оператора зсуву, а в другому – продовжується з поточного оператора зсуву. Відповідно ці способи дістали назву лінійного та кільцевого генератора.

Перевагою алгоритмів локального пошуку над багатьма іншими евристиками є те, що простір варіантів може бути досліджений дуже ефективно: замість того, щоб обчислювати значення цільової функції для $y \in O(x)$, достатньо обчислити тільки різницю $\Delta = f(x) - f(y)$, використовуючи властивості прикладної задачі, щоб уникнути явного обчислення $f(x)$ та $f(y)$ і перевірити, чи є величина Δ додатною.

Приріст Δ є певним аналогом першої похідної, оскільки, загальніше,

$$\Delta = \frac{f(x) - f(y)}{d(x, y)},$$

де $d(x, y)$ – відстань між x та y (зазвичай для точок околу $d(x, y) = 1$).

Отже, це аналог спуску по антиградієнту, але він ураховує, що поняття напрямку в комбінаторній оптимізації в загальному випадку відсутнє.

Майже для кожної комбінаторної задачі оптимізації є околи, для яких Δ може бути обчислена набагато швидше, ніж $f(y)$. Наприклад, у задачі комівояжера обчислення Δ вимагає часу $O(1)$ (чотири доданки у випадку 2-замін Ліна чи шість – у випадку 3-замін, тоді як обчислення $f(y)$ вимагає $O(n)$ операцій). Іншими словами, алгоритм ЛП здатний опрацювати n розв'язків у просторі пошуку за той самий час, за який, наприклад, еволюційний алгоритм оцінює тільки окремий розв'язок. ЛП використовує цю перевагу – як і інші алгоритми пошуку в околах, включаючи описані нижче.

Критерієм завершення часто вважають такі умови:

1) Відсутність поліпшуючої точки в околі поточного розв'язку, тобто перегляд усіх точок околу не завершився поліпшенням.

2) Проведення наперед заданої кількості ітерацій.

3) Вичерпання заданого ліміту часу.

За початковий варіант розв'язку найчастіше беруть довільний припустимий розв'язок. Для його знаходження часто використовують випадкове генерування варіантів розв'язку з подальшою перевіркою їх на припустимість – інколи таких варіантів генерується кілька, а з них вибирається найкращий. Часто також використовуються розв'язки, знайдені простими евристичними алгоритмами; при цьому намагаються максимально врахувати специфіку задачі.

Зазначимо, що обмеження задачі логічно враховувати при знаходженні чергового варіанта розв'язку, перевіряючи його на припустимість.

У комбінаторних просторах окіл часто визначається з використанням метрик, основні з яких були подані вище (див. п. 1.2). У деяких застосуваннях використовуються спеціальні метрики, які враховують особливості задач. Наприклад, для ЗК поняття сусідства може вводитися на основі 2-замін Ліна чи вставлення фрагментів.

До методів локального пошуку передусім належать:

- пошук (спуск) в околах (локальна оптимізація);
- пошук зі змінними околами;
- керований локальний пошук;
- табу-пошук.

Типовим представником детермінованих локально оптимальних алгоритмів є метод вектора спаду (МВС), запропонований І. В. Сергієнком у 1964 р. для пошуку екстремумів у дискретних метричних просторах.

У цьому методі використовуються метричні околи довільної точки $x \in X$ із заданим радіусом $\rho > 0$, які, як і у розд. 1, будемо позначати $L_\rho(x)$. Псевдокод алгоритму з переходом до першого поліпшення подано схемою 3.1.

У наведеній схемі h – кількість здійснених ітерацій, x^h – поточний розв'язок на ітерації h.

Критерієм завершення є відсутність у околі поточного розв'язку варіантів, які поліпшують значення цільової функції.

procedure *MBC*(x)
Задання $\rho > 0$;
$x^0 :=$ деякий припустимий варіант розв'язку;
$h := 0$;
while *окіл* $L_\rho(x^h)$ *поточного варіанта не переглянутий повністю* **do**
$y :=$ Генерація Чергової Точки Околу $L_\rho(x^h)$;
$\Delta = f(x^h) - f(y)$;
if $\Delta > 0$ **then**
$h := h + 1$; $x^h := y$;
end if;
end while;
return $x = x^h$; {x^h – локально оптимальний розв'язок задачі}
end

Схема 3.1. Псевдокод алгоритму МВС до першого поліпшення

Оцінка кількості ітерацій МВС

Теорема. Нехай цільова функція задачі обмежена знизу та додатна – $f(x) \geq C > 0$, а величина

$$\delta = \min_{f(x) \neq f(y)} \{|f(x) - f(y)|\} > 0.$$

Тоді кількість ітерацій алгоритму МВС не перевищить числа

$$(f(x^0) - C) / \delta,$$

де x^0 – початкове наближення.

Доведення.

У процесі обчислень за m ітерацій отримуємо послідовність $f(x^0) > f(x^1) > \ldots > f(x^m)$, оскільки зі схеми випливає, що ми маємо чітко виражений алгоритм спуску. Звідси

$$f(x^0) - C \geq f(x^0) - f(x^m) \geq m\delta\,,\ f(x^0) - C \geq m\delta\,,$$

$$m \leq \frac{f(x^0) - C}{\delta}\,.$$

Теорему доведено.

Умови теореми не є суттєво обмежувальними й застосовні до широкого кола задач.

3.3. Алгоритми Ліна – Кернігана для ЗК

Як приклад спеціальних алгоритмів розглянемо відомі алгоритми *k-замін Ліна* (*k*-exchange), що базуються на процедурі вилучення *k* ребер і додавання *k* нових.

Для побудови сусідніх варіантів (околу) до поточного маршруту було запропоновано використовувати такі процедури: 1) транспозиції міст; 2) вставка міста; 3) переміщення фрагмента шляху (із 1–3 ребер); 4) заміна *k* ребер.

Схема алгоритму 2-opt Ліна, що базується на процедурі 2-заміни, подана схемою 3.2. Обчислювальна складність алгоритму – $O(n^2)$. За результатами деяких досліджень, алгоритм 2-opt забезпечує якість маршруту, на 4–7 % гіршу від оптимальної.

Аналогічно записується алгоритм 3-замін, у якому вилучаються три ребра з наявного розв'язку та вставляються три нові так, щоб знову був гамільтонів цикл. Така процедура підвищує ефективність пошуку, але суттєво збільшує час розрахунків.

Обчислювальна складність алгоритму – $O(n^3)$. За результатами досліджень, алгоритм 3-opt може знаходити маршрути в деяких евклідових ЗК на площині, довжина яких лише на 2,5–4 % більша від оптимальної.

У своїх роботах Лін показав, що 3-заміни значно кращі за 2-заміни, а ефект від використання 4-замін є незначним і нівелюється суттєвим зростанням обчислювальних затрат.

```
procedure 2-OPT(x);
створення початкового маршруту x за допомогою деякого
алгоритму;
while не виконуються умови завершення do
for i = 1 to n do
    for j = 1 to n do
вибрати несуміжні ребра e_i, e_j;
      if існують ребра e_i', e_j' такі, що забезпечують меншу
довжину маршруту
      then замінити в x ребра e_i, e_j на ребра e_i', e_j';
      endfor
endfor
endwhile
return x
end
```

Схема 3.2. Псевдокод алгоритму 2-opt

На основі запропонованих процедур *k*-замін був розроблений алгоритм Ліна – Кернігана, який широко використовується як автономно, так і як підлеглі процедури в різних метаевристиках для розв'язання низки задач на графах. Головна ідея полягає у формуванні двох підмножин ребер: тих, що входять у поточний маршрут, і тих, що не входять, а також у використанні спеціальної процедури заміни ребер з першої підмножини на ребра з другої. Це називається ротацією, у результаті застосування якої прагнуть досягти зменшення довжини маршруту, якщо це можливо.

Розроблено багато різних модифікацій алгоритму Ліна – Кернігана, серед яких – ітеративний Лін – Керніган (Iterated Lin-Kernighan), ланцюговий Лін – Керніган (Chained Lin-Kernighan), багаторівневий Лін – Керніган (Multilevel Lin-Kernighan), Лін – Керніган із кластерною компенсацією (Lin-Kernighan with cluster compensation) тощо. Усі модифікації розроблено для застосування переважно в задачах великих розмірностей чи у специфічних задачах пошуку маршрутів на площині, перш за все ЗК чи задачах оптимального розбиття графів.

3.4. Приклади обчислення значень Δ-оцінок

Квадратична задача про призначення. Розглянемо метричний окіл $L_1(x)$, тобто покладемо величину радіуса $\rho = 1$. У більшості випадків матриці C та D у КЗП є симетричними. Тоді обчислення значень Δ можна виконати із суттєво нижчою трудомісткістю, уникнувши прямого обчислення значення $f(y)$.

Можна показати, що в цьому разі

$$\Delta = \sum_{k \neq i,j} \left(c_{ik} - c_{jk}\right)\left(d_{x_i x_j} - d_{x_j x_k}\right).$$

Дійсно, конкретна точка простору розв'язків (перестановка) $y \in L_1(x)$ утворюється із $x \in P_n$ шляхом однієї транспозиції компонент i, j, $1 \le i \ne j \le n$.

Нехай $x = (x_1, \ldots, x_n)$, $y = (y_1, \ldots, y_n)$, тоді можемо записати:

$$y_k = x_k,\ k \neq i, j;$$

$$y_i = x_j,\ y_j = x_i.$$

Виразимо величину Δ через компоненти перестановки x. Маємо:

$$\Delta \equiv f(x) - f(y) = \sum_{k=1}^{n}\sum_{l=1}^{n} c_{kl}\left(d_{x_k x_l} - d_{y_k y_l}\right) =$$

$$= \sum_{\substack{k \neq i \\ l \neq j}} (\ldots) + \sum_{l \neq j} c_{il}\left(d_{x_i x_l} - d_{y_i y_l}\right) + \sum_{k \neq i} c_{kj}\left(d_{x_k x_j} - d_{y_k y_j}\right) =$$

$$= \sum_{k \neq j} c_{ik}\left(d_{x_i x_k} - d_{x_j x_k}\right) + \sum_{k \neq i} c_{kj}\left(d_{x_k x_j} - d_{x_k x_i}\right) =$$

$$= \sum_{\substack{k \neq i \\ k \neq j}} \left(c_{ik} - c_{kj}\right)\left(d_{x_i x_k} - d_{x_k x_j}\right).$$

Задача комівояжера. Нехай маємо певний маршрут. Для побудови околу з мінімальним радіусом в алгоритмі 2-*замін Ліна* було запропоновано замінювати два довільних неінцидентних ребра на два інших (див. рис. 3.1), чим фактично й утворю-

ється окіл. Неважко переконатися, що кількість елементів у ньому буде $n(n-3)/2$.

Таким чином, у цьому алгоритмі окіл формують маршрути, які утворені заміною чотирьох ребер (рис. 3.1), тому неформально зміну цільової функції задачі можна описати так:

$$\Delta = \textit{довжина}(\textit{старе ребро } 1) + \textit{довжина}(\textit{старе ребро } 2) - \\ - \textit{довжина}(\textit{нове ребро } 1) - \textit{довжина}(\textit{нове ребро } 2).$$

Отже, у цьому алгоритмі, незалежно від розмірності n задачі, для визначення зміни маршруту достатньо лише чотирьох доданків, які визначаються ребрами, задіяними в заміні.

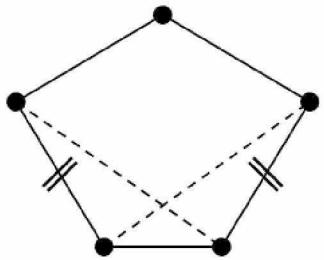


Рис. 3.1. Приклад 2-заміни Ліна

Варто зазначити, що в багатьох типах ЗКО з обмеженнями, коли визначена область $D \subseteq X$ – підмножина припустимих варіантів, тобто маємо постановку задачі у вигляді (1.1), можна ефективно застосовувати алгоритми локального пошуку. У цьому випадку слід на кожній ітерації замість перегляду всього околу $O(x)$ розглядати множину розв'язків $O(x) \cap D$, тобто при перегляді в околі поточного розв'язку необхідно додатково перевіряти досліджуваний варіант розв'язку на припустимість, що в більшості задач не породжує суттєвих ускладнень.

Головним недоліком алгоритмів ЛП є те, що отримані розв'язки (за побудовою обчислювального процесу) є лише локальними оптимумами. Як тільки досягнуто локально оптимальний розв'язок, стандартний алгоритм завершується. Часто це трапляється на початку обчислювального процесу, коли значення цільової функції ще досить віддалене від глобального оптимуму – цю ситуацію називають передчасною збіжністю.

Тому, щоб долати передчасну збіжність і дати змогу знаходити точніші розв'язки, дослідниками запропоновано низку спеціальних підходів до модифікації процедури стандартного детермінованого ЛП, основні з яких відображені на рис. 3.2.

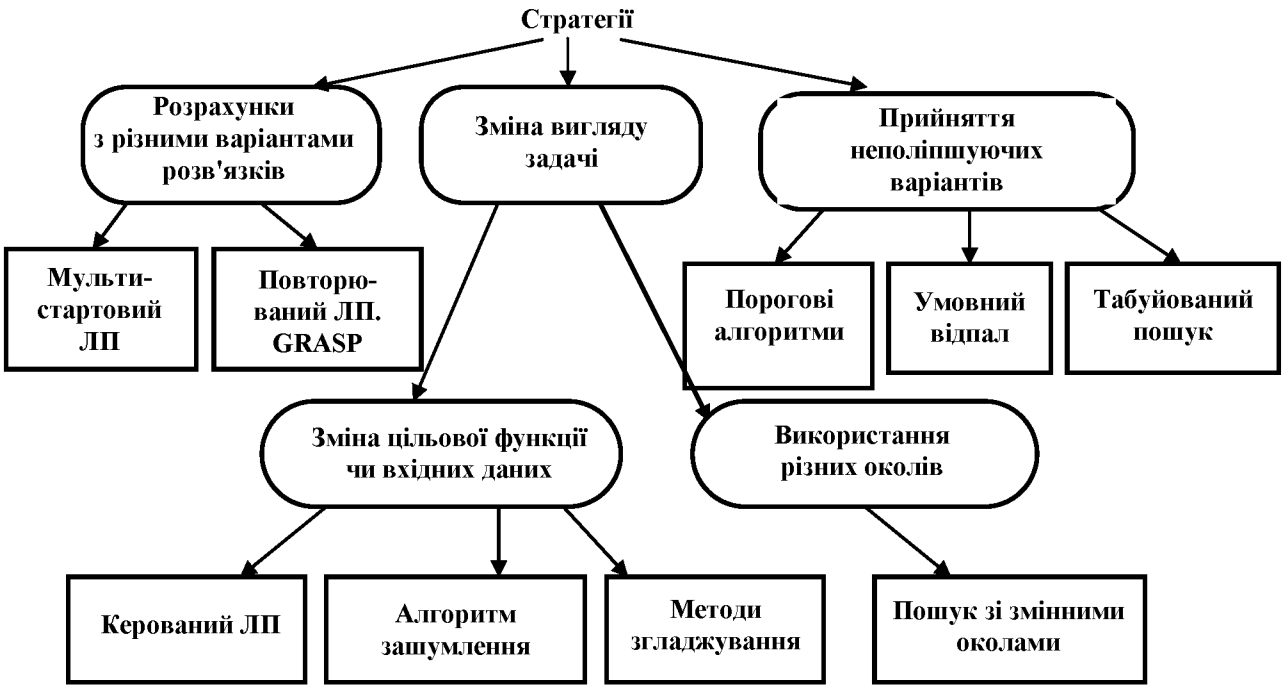


Рис. 3.2. Стратегії поліпшення алгоритмів детермінованого ЛП

3.5. Пошук зі змінними околами

Пошук із пульсуючими чи змінними околами (у зарубіжній літературі – Variable Neighborhood Search, VNS) – метаевристика, яка явно застосовує стратегію, засновану на динамічній зміні структури околів. Алгоритм є дуже загальним і має багато степенів вільності для того, щоб розробляти конкретні модифікації та специфічні реалізації.

На кроці ініціалізації має бути визначена структура околів. Ці околи можуть бути довільними, але доречно вибирати послідовність $\left\|O^1(x)\right\| < \left\|O^2(x)\right\| < \ldots < \left\|O^{\rho_{\max}}(x)\right\|$ околів з потужністю, що збільшується. Теоретично вони можуть включатися один в інший $O^1 \subset O^2 \subset \ldots \subset O^{\rho_{\max}}$, проте при цьому слід уникати повторного перегляду розв'язків. Потім генерується початковий варіант розв'язку, ініціалізується індекс околу й алгоритм ЛП *LS* виконує ітерації доти, поки умова завершення не виконана (схема 3.3).

```
procedure VNS(x)
  Вибрати структуру множини околів O^ρ(x), ρ = 1, …, ρmax;
  x: = ЗгенеруватиПочатковийРозв'язок;
  while умови закінчення не задоволені do
    ρ: = 1
    while ρ ≤ ρmax do
      x′: = Вибрати випадково з O^ρ(x); {стадія струсу}
      x″ := LS(x′);
      if f(x″) < f(x) then
        x := x″;
        ρ := 1
      else
        ρ := ρ+1
      endif
    endwhile
  endwhile
end
```

Схема 3.3. Псевдокод алгоритму пошуку зі змінними околами

Мета стадії струсу полягає у збуренні розв'язку таким чином, щоб забезпечити хорошу початкову точку для локального пошуку. Ця точка в ідеалі має належати чаші тяжіння локального оптимуму, відмінного від поточного, але не бути "дуже далеко" від x, інакше алгоритм вироджується в простий рестарт. Крім того, вибір точки x' в околі поточного варіанта розв'язку ймовірно дасть новий варіант, який успадкує деякі його хороші риси.

Процес зміни околу в разі відсутності поліпшень відповідає диверсифікації пошуку. Зокрема, вибір околу з кількістю елементів, що збільшується, зумовлює прогресивну диверсифікацію. Ефективність цієї динамічної стратегії околів можна пояснити тим фактом, що "погане" місце на ландшафті пошуку, яке визначається одним околом, може бути "хорошим" на ландшафті пошуку відносно іншого околу. Більш того, варіант розв'язку, який є локально оптимальним щодо одного околу, може не бути локально оптимальним щодо іншого, більшого околу.

У спуску зі змінними околами (Variable Neighborhood Descent, VND) стадія струсу відсутня, відбуваються тільки поліпшення. Уперше такий алгоритм був запропонований під назвою *алгоритм пульсуючих околів* у 1979 р.

Як видно з наведеного вище, вибір структури околів – ключова частина пошуку та спуску зі змінними околами. Вибрані околи мають ураховувати й відображувати різні властивості та характеристики простору пошуку, тобто забезпечувати виділення різних його підобластей.

3.6. Керований локальний пошук

Пошук зі змінними околами (як і табуйований пошук, про який ітиметься далі) має справу з явно динамічними околами з метою продуктивно й ефективно досліджувати простір пошуку. Інший підхід керування пошуком полягає в тому, щоб динамічно змінювати цільову функцію. Серед загальних методів, які використовують цей підхід, є керований локальний пошук (Guided Local Search, GLS), який також належить до класу методів детермінованого локального пошуку.

Основний принцип керованого локального пошуку полягає в намаганні в процедурі пошуку поступово виходити з локального оптимуму, змінюючи ландшафт пошуку. При цьому простір розв'язків і структура околу залишаються незмінними, тоді як цільова функція *f* динамічно змінюється з метою оцінювання поточного локального оптимуму як "менш бажаного" (рис. 3.3).

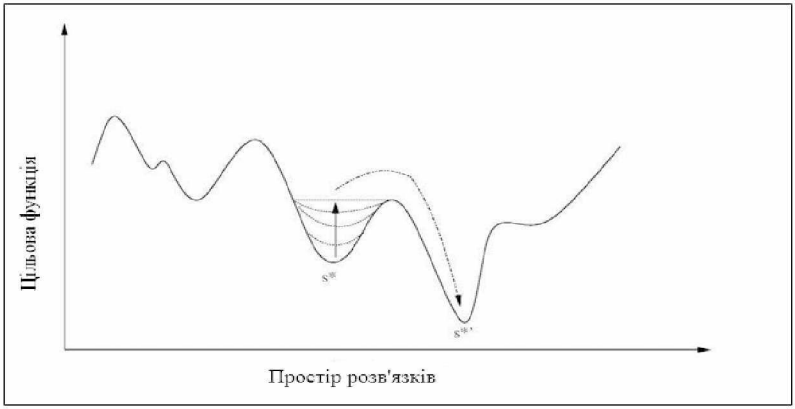


Рис. 3.3. Ілюстрація керованого локального пошуку

Механізм, що застосовується в GLS, заснований на використанні *сутностей* (їх ще називають *особливостями* чи *компонентами*) варіанта розв'язку, які є будь-яким типом властивостей або характеристиками чи складовими розв'язку, що можуть бути задіяні для формування й розрізнення розв'язків. Наприклад, сутностями варіанта розв'язку в задачі комівояжера (тобто маршруту) можуть бути дуги між парами міст, які входять до даного маршруту.

Уводимо індикаторну функцію $I_i(s)$, яка вказує, чи присутня сутність *i* у варіанті *x*:

$$I_i(x) = \begin{cases} 1, \text{ якщо сутність } i \text{ присутня у варіанті розв'язку } x, \\ 0, \text{ в іншому разі.} \end{cases}$$

Цільова функція *f* змінюється на нову *f'* додаванням величини, яка залежить від *m* сутностей:

$$f'(x) = f(x) + \lambda \sum_{i=1}^{m} p_i \cdot I_i(x), \tag{3.1}$$

де p_i називають *штрафними параметрами*, а λ – *параметром регуляризації* (*масштабування*). Штрафні параметри зважують важливість сутностей: чим вище p_i, тим вища важливість сутності i, отже, вище вартість наявності цієї сутності в даному розв'язку. Параметр регуляризації збалансовує значущість сутностей відносно значення основної цільової функції задачі. Індикаторна функція виокремлює лише ті сутності, які мають відношення до поточного розв'язку.

Алгоритм починає з деякого початкового варіанта розв'язку й застосовує той чи інший метод локального пошуку *LS* доти, поки не буде досягнутий локальний мінімум (схема 3.4). Потім вектор $p=(p_1,\ldots,p_m)$ штрафів модифікується шляхом збільшення деяких з елементів і локальний пошук починає знову, використовуючи модифіковану згідно з (3.1) цільову функцію.

procedure *GLS*(x);
  x: = деякий припустимий варіант розв'язку;
  p: = 0;
  while *не виконуються умови завершення* **do**
    x: = *LS* (x, f');
    for i : $I_i(x) \neq 0$ **do**

$$u_i = \frac{c_i}{1+p_i};$$

$$u_j = \max_{s:\, I_s(x)\neq 0} \{u_s\};$$

    p_j : = p_j + 1; {Змінюємо (f', p); p – вектор штрафів}
  end while
  return x
end

Схема 3.4. Структура алгоритму керованого локального пошуку

Штрафуються сутності, які мають максимальну *корисність* (*utility*):

$$Util(x,i) = I_i(x) \cdot \frac{c_i}{1+p_i},$$

де c_i – *витрати*, тобто величини, які відображають евристичну оцінку відносної важливості сутностей порівняно з іншими: чим вища вартість, тим вище корисність сутностей. Проте вартість масштабується штрафним параметром для перешкоджання алгоритму бути повністю зміщеним до вартості, щоб робити його чутливим до історії пошуку.

Процедура оновлення штрафів може бути доповнена додаванням мультиплікативного правила до простого правила приросту (яке застосовується на кожній ітерації).

Тут *LS* – деякий алгоритм локального пошуку.

Мультиплікативне правило має вигляд $p_i \leftarrow p_i \cdot \alpha$, де $\alpha \in (0,1)$. Це правило застосовується з нижчою частотою, ніж приріст штрафів (наприклад, кожні кілька сотень ітерацій), з метою руйнування ваг оштрафованих сутностей, щоб уникати формування надто різкого ландшафту цільової функції. Важливо звернути увагу, що правила оновлення штрафів часто дуже чутливі до конкретної задачі, яка розв'язується.

3.7. Табуйований пошук

Алгоритми *табуйованого пошуку*, чи *табу-пошуку* (Tabu Search), розроблялись із самого початку для роботи з банками пам'яті. Термін *tabu search* був запропонований Ф. Гловером у 1986 р. Головна ідея табу-пошуку полягає в локальній зміні наявного варіанта розв'язку задачі разом із запам'ятовуванням послідовності таких змін та їх використанням, щоб запобігти повторній побудові неефективних розв'язків і зациклення процедури пошуку. Зроблені модифікації запам'ятовуються в банках пам'яті під назвою *списків заборон,* чи *табу-списків* (*tabu list*).

Алгоритм, що лежить у основі процедури табу-пошуку, нагадує процес пошуку людиною задовільного розв'язку оптимізаційної задачі. Пошук починається з початкового варіанта розв'язку, зазвичай не оптимального; потім розв'язок поступово поліпшується локальними модифікаціями. Також слід зазначити, що пізніші модифікації розв'язку можуть не обов'яз-

ково його поліпшувати, але вони спрямовані на поліпшення розв'язку в майбутньому.

Цей підхід частково нагадує алгоритми імітаційного відпалу, що будуть розглянуті далі, у яких також можливе тимчасове погіршення значення цільової функції при формуванні поточного наближення розв'язку. Розв'язок, який був відвіданий нещодавно, включається в *список заборон* (*табу*) і тому далі не розглядатиметься як кандидат на відвідування (схема 3.5). Однак обслуговування списку заборон і пошук у межах цього списку часто є надто трудомістким, щоб бути придатним для багатьох практичних застосувань.

procedure *Tabu_Search* (x);
begin
$x :=$ деякий припустимий варіант розв'язку;
$T := \varnothing$;
$x_{rec} := x$;
repeat
пошук прийнятного варіанта $y \in O(x) \setminus T$;
$x := y$;
$T := T \cup x$;
if $f(x) < f(x_{rec})$ **then** $x_{rec} := x$;
until *виконається критерій завершення*;
return $x = x_{rec}$;
end

Схема 3.5. Табуйований пошук

Виділяють три основні різновиди методу:

- *строге табуювання* – запам'ятовуються всі досліджені розв'язки з метою заборони потрапляння в усі вже відвідані точки простору розв'язків;
- *фіксоване табуювання* – заборона стосується лише k останніх точок, тобто k – довжина списку заборон (список табу);
- *реактивне табуювання* – довжина списку заборон k не є константною, а динамічно змінюється в процесі роботи алгоритму.

Альтернативою наведеному підходу, який вимагає суттєвих затрат пам'яті, є збереження в списку заборон не самого чергового варіанта розв'язку, а *переміщень* (*зсувів*), які приводять до цього розв'язку. Точніше, для кожного можливого переміщення у структурі даних прапорець указує, чи забороняється відповідне переміщення чи ні. Оскільки ця процедура в деяких випадках є дуже обмежувальною, то були включені *умови рівня прагнення*.

Рівень прагнення переміщення показує, чи можна це переміщення використовувати в пошуку, незважаючи на заборонений статус. Наприклад, якщо переміщення приводить до кращого розв'язку, ніж попередній, то його можна дозволити незалежно від того, заборонене воно чи ні. Тоді оператор вибору точки в околі поточного варіанта на схемі 3.5 має трансформуватися таким чином:

пошук можливого переміщення

$$\omega \in \Omega(x) : \left\{ y = \omega(x) \in O(x) \,\&\, \left(\omega \notin T \vee h(\omega) \geq a(\omega)\right)\right\},$$

де $\Omega(x)$ позначає множину можливих переміщень для точки x, T_ω – список заборонених переміщень, $h(\omega)$ – обчислений рівень прагнення переміщення ω, а $a(\omega)$ – поріг прагнення для цього переміщення ω.

Альтернативно можуть використовуватися більш ніж один список заборон (табу) і більш ніж одна функція прагнення. Є різні варіанти розвитку цієї основної схеми: наприклад, можуть використовуватися списки табу змінної довжини, щоб поліпшити ефективність алгоритму.

Крім того, інтенсифікація пошуку в описаній вище процедурі може комбінуватися з етапом *диверсифікації*. Якщо табуйований пошук виявився нездатним знайти новий кращий розв'язок за задану кількість ітерацій, то він може бути зосереджений на іншій області простору розв'язків. Робота етапу диверсифікації має полягати у визначенні перспективних альтернативних областей, ґрунтуючись на пам'яті про пошук.

Алгоритми табуйованого пошуку широко розвивались і застосовувались для розв'язання різноманітних задач. Упродовж досить значного періоду часу вони разом з АІВ вважалися най-

популярнішими методами розв'язання задач КО. Це пов'язувалося із тим, що обчислювальна потужність комп'ютерних систем, на яких здійснювались розрахунки, була відносно невисокою, а зазначені два класи методів здатні знаходити розв'язки з підвищеною точністю за помірних затрат машинного часу та машинної пам'яті. Однак останніми роками, коли обчислювальна потужність систем значно зросла, найпопулярнішими, виходячи з численних публікацій і конференцій з цієї тематики, стали складніші та потужніші алгоритми, наприклад генетичні, що спричинило певне зменшення інтересу дослідників до чистих методів табуйованого пошуку.

Останнім часом табуйований пошук найчастіше застосовується як допоміжна процедура для локального вдосконалення розв'язку в гібридних алгоритмах. Розглядаються можливі підходи до використання алгоритму табуйованого пошуку як процедури удосконалення розв'язку, вбудованої в схему різних метаевристик.

Таким чином, методи табуйованого пошуку з плином часу почали дещо відходити на другий план, даючи дорогу іншим, складнішим і потужнішим методам. Проте простота реалізації та досить висока точність отримуваних розв'язків дозволяє стверджувати, що методи табуйованого пошуку будуть використовуватись ще довго.

3.8. Порогові алгоритми

Порогові алгоритми, або *порогове прийняття* (Threshold accepting), так само як імітаційний відпал і алгоритми прискореного ймовірнісного моделювання, є розвитком алгоритму локального пошуку, розглянутого вище. Названі методи – це методи пошуку в околах, але вони відрізняються критеріями прийняття варіанта розв'язку в околі. Тоді як у локальному пошуку приймаються тільки кращі щодо значення цільової функції розв'язки, зазначені алгоритми дозволяють приймати для подальшого дослідження розв'язки гірші, ніж поточний розв'язок.

Пороговий алгоритм був запропонований як альтернатива імітаційного відпалу, який розглядатимемо далі. Тут критерій Метрополіса, що використовується в імітаційних алгоритмах, замінено набагато простішим (в обчислювальному аспекті) критерієм прийняття. Досліджуваний розв'язок приймається, якщо різниця цільових значень для поточного й сусіднього варіантів розв'язку нижча за деякий поріг θ. На схемі 3.6 зображено обчислювальний процес в алгоритмах порогового прийняття.

```
procedure Threshold_Accepting (z);
 x := деякий припустимий варіант розв'язку;
 h := 0;  θ_h := Θ(0);  z := x;
 repeat
   y := ГенераціяЧерговоїТочкиОколу O(x);
   Δ := f(x) – f(y);
   if Δ > θ_h then x := y;
   if f(x) < f(z) then z := x;
   h := h + 1;
   θ_h := Θ(h);
 until виконається критерій завершення;
 return z;
end
```

Схема 3.6. Метод порогового прийняття

У класичному пороговому алгоритмі використовується детермінований критерій відбору в околі поточного варіанта – на відміну від того, що використовується в алгоритмах, які базуються на ймовірнісних механізмах.

Як і в алгоритмах ЛП, критерієм завершення часто є умова: на поточній ітерації окіл $O(x)$ поточного варіанта переглянутий повністю, а прийняття нового не відбулося. При реалізації алгоритму мають бути визначені початковий припустимий варіант розв'язку та правило модифікації для θ у формі функції $\Theta(h)$, яка повинна

бути монотонно спадною до нуля (зазначимо, що дослідниками був запропонований і алгоритм без вимоги монотонності).

Алгоритми порогового прийняття можна розглядати як узагальнення схеми локального пошуку, оскільки при виборі $\Theta(h)\equiv 0$ можна отримати саме стандартний алгоритм ЛП. Водночас уведення в порогову умову ймовірнісних механізмів дозволяє зробити алгоритм ефективнішим – про це свідчить досвід застосування алгоритмів стохастичного ЛП, таких як алгоритми імітаційного відпалу та *G*-алгоритми.

4. СТОХАСТИЧНИЙ ЛОКАЛЬНИЙ ПОШУК

4.1. Загальна схема стохастичного локального пошуку

Алгоритми стохастичного локального пошуку є одними з найпростіших метаевристичних методів оптимізації. Їх обчислювальна схема базується на процедурі локального пошуку, у якій на кожній ітерації також досліджується окіл поточного розв'язку, але головною особливістю є те, що при переході до наступної ітерації з певною ймовірністю може прийматися розв'язок, який погіршує значення цільової функції, причому не обов'язково з поточного околу. Термін *стохастичний* відображає ту обставину, що обчислювальна схема базується на використанні рандомізованих процедур.

Загальна обчислювальна схема алгоритмів стохастичного локального пошуку показана на рис. 4.1. Робота конкретного алгоритму починається із заданого початкового наближення, яке може бути вибрано випадково чи знайдено іншим алгоритмом. У більшості алгоритмів цей пошук триває до того часу, доки всі сусідні варіанти розв'язків не стануть гіршими за його поточне значення. Зупинка роботи алгоритму в такому разі означає, що досягнутий локальний оптимум.

Блоки 1–2 та 4–8 – це компоненти провідної (керуючої) процедури алгоритму, 3 – процедура локального пошуку, яка відіграє роль підлеглої. Етап 7 збурення, який зазвичай спрямований на диверсифікацію пошуку в просторі розв'язків, може бути відсутнім у деяких алгоритмах СЛП.

У траєкторних метаевристиках стохастичного локального пошуку рандомізовані процедури можуть використовуватися при:

- знаходженні початкового наближення;
- генеруванні досліджуваних точок околу поточного розв'язку;
- розв'язанні питання, чи приймати чергову досліджувану точку як наступний поточний варіант розв'язку;
- реалізації процедури збурення;
- формуванні умов завершення обчислювального процесу.

Рис. 4.1. Загальна схема стохастичного локального пошуку

Відповідно в кожному із зазначених випадків можливого застосування рандомізованих процедур використовуються свої параметри, налаштування яких є окремою складною проблемою, розв'язання якої покладається на розробника алгоритму.

Формування початкового наближення хоча й не є власне частиною алгоритмів СЛП, проте в більшості публікацій розглядається як внутрішня складова алгоритмів цього класу.

Як процедура локального пошуку можуть використовуватися алгоритми детермінованого локального пошуку в "чистому вигляді" (див. розд. 3), а також модифіковані (зокрема рандомізовані) схеми локального пошуку.

Блок 5 у деяких алгоритмів стохастичного локального пошуку пов'язаний з акцентуванням пошуку в напрямі його або інтенсифікації, або диверсифікації.

Процедури збурення, якщо вони присутні, призначені для реалізації можливості переходу до точок простору розв'язків, які не належать до околу поточного розв'язку, тобто для диверсифікації пошуку.

Оскільки алгоритми стохастичного локального пошуку не мають релаксаційного (тобто монотонно поліпшуючого) характеру, то необхідно відслідковувати й фіксувати історію пошуку в пам'яті алгоритму, яка, у мінімальному випадку, використовується для запам'ятовування рекорду – найкращого розв'язку з-поміж тих варіантів, які були знайдені в процесі роботи алгоритму.

Саме тому, що останній зі знайдених розв'язків не обов'язково буде найкращим серед досліджених, результуючий розв'язок формується з використанням пам'яті алгоритму.

Варто зазначити, що будь-який алгоритм стохастичного локального пошуку реалізує марковський процес. Зокрема, поведінка алгоритму при виборі шляху переміщення від поточного розв'язку до наступного не залежить від передісторії пошуку.

Для ефективної реалізації алгоритмів повторюваного поліпшення пряме обчислення значень цільової функції зазвичай замінюється застосуванням оцінки її зміни (що також називається дельта-оцінкою) після кожного пошукового кроку. Це робиться шляхом обчислення різниці між значеннями функцій оцінки поточного й сусіднього варіантів розв'язку. Як уже зазначалося, оскільки

в багатьох задачах значення цільової функції визначається як композиція незалежних внесків окремих компонентів розв'язку, то це може бути зроблено на основі розгляду внесків тільки тих компонентів, якими відрізняються сусідні варіанти розв'язків.

Одним із прямих шляхів модифікації повторюваного поліпшення для розумної роботи з локальними мінімумами є способи запуску алгоритму з різних початкових наближень. Хоча у випадку невеликої кількості локальних мінімумів ця стратегія рестарту може спрацювати добре, але в багатьох практичних задачах вона неефективна. В альтернативному варіанті можна послабити критерій поліпшення у випадку, коли знайдено локальний мінімум, зробити вибраний довільно неполіпшуючий крок або вибрати одного з сусідів, який дає найменший приріст значення цільової функції.

На сучасному етапі розвитку наближених алгоритмів для задач комбінаторної оптимізації для ефективної боротьби з передчасним потраплянням у локальні оптимуми найчастіше використовуються такі алгоритми стохастичного локального пошуку:

- повторюваний локальний пошук;
- алгоритми імітаційного відпалу;
- алгоритми прискореного ймовірнісного моделювання.

4.2. Повторюваний локальний пошук

Головним недоліком алгоритмів локального пошуку є передчасна зупинка обчислень у локальних оптимумах, що віддалені від глобального оптимуму. Найпростішим способом розв'язання цієї проблеми є повторення локального пошуку з інших початкових варіантів розв'язку – такий підхід називається *мультистартовим ЛП*. Іншим способом поліпшення алгоритму є використання випадкових змін (збурень) знайденого локально оптимального розв'язку, після чого отримана точка використовується як початковий варіант розв'язку для локального пошуку.

Основна ідея *повторюваного локального пошуку* (Iterated Local Search) полягає у проведенні спрямованого пошуку не в

повному просторі розв'язків, а в меншому підпросторі, що складається з локально оптимальних розв'язків.

У цих алгоритмах генерується (випадково) початковий варіант розв'язку й застосовується якийсь із алгоритмів локального пошуку *LS* для перетворення наближеного розв'язку на локальний оптимум – такий алгоритм називається *вбудованим*. Потім новий початковий розв'язок генерується шляхом збурення/мутації поточного розв'язку (найкращий знайдений локальний оптимум). Локальний оптимум далі знаходиться з цього розв'язку застосуванням вбудованого алгоритму (зазвичай ним є локальний пошук). Якщо новий локальний оптимум має краще значення цільової функції, то його приймають як новий поточний розв'язок, з якого генерують новий стартовий розв'язок. Ці кроки повторюються, поки не буде виконано визначений розробником критерій завершення (схема 4.1).

```
procedure Iterated_Local_Search (x);
 begin
 x : = деякий припустимий початковий варіант;
 x : = LS (x);
 repeat
 y : = Модифікація(x, історія);
 y : = LS(y);
 x : = КритерійПрийняття(x, y, історія);
 until виконається умова завершення;
 return x;
 end
```

Схема 4.1. Псевдокод повторюваного локального пошуку

Алгоритм повторюваного ЛП був спочатку запропонований Баумом (1986) під назвою *повторюваний спуск*. Використана проста ідея має тривалу історію й перевідкривалася багатьма авторами під різними назвами:

- ланцюги Маркова з великим кроком;
- повторюваний алгоритм Ліна – Кернігана;
- ланцюгова оптимізація;
- комбінація згаданих назв.

Найчастіше вбудований алгоритм, що використовується в повторюваному ЛП, – це простий алгоритм спуску, який оперує локальними оптимумами: поточний розв'язок змінюється випадково та приймається тільки кращий локальний оптимум. Щоб дозволяти алгоритму приймати локальний оптимум з гіршою відповідністю, алгоритм спуску в повторюваному ЛП може бути замінено ефективнішим алгоритмом – імітаційним відпалом, чи *G*-алгоритмом (про це йтиметься далі).

Наприклад, в алгоритмі, названому алгоритмом *марковських ланцюгів з великим кроком*, як ЛП використовується *імітаційний відпал*: генерований локальний оптимум приймається згідно з критерієм Метрополіса залежно від поточної температури.

Зазначимо, що всі розглянуті алгоритми оперують тільки одним варіантом розв'язку.

4.3. Алгоритми імітаційного відпалу

З метою уникнення передчасної зупинки в алгоритмах ЛП на кожній ітерації можна допускати не лише перехід в околі до поліпшуючих варіантів розв'язку, але й дати можливість здійснювати інколи переходи до точок, які мають гірше значення цільової функції, аніж поточний варіант (центр околу). Саме така ідея лягла в основу алгоритмів *імітаційного*, або *моделюваного*, *відпалу* (AIB) (Simulated annealing), які базуються на аналогії з процесом оптимізації та фізичним процесом відпалу. В алгоритмах даного класу пошук глобального розв'язку імітується процесом релаксації термодинамічної системи з багатьма степенями вільності, де значенням узагальненої енергії (гамільтоніана) є цільова функція задачі. Зі статистичної механіки відомо, що такі системи при деякій кінцевій температурі прямують до стану рівноваги. Флуктуації, що виникають при високих температурах, мають імовірнісний характер і можуть зумовлювати тимчасове збільшення енергії системи, однак при поступовому зниженні температури термодинамічна система досягає стану рівноваги, що відповідає мінімальному значенню узагаль-

неної енергії системи. У фізичному аспекті *відпал* – це тепловий процес для отримання стану з низькою енергією тіла в тепловій ванні. Загалом процес можна описати так. Спочатку температура теплової ванни збільшується до максимального значення, при якому тверде тіло тане. Таким чином, усі частинки тіла отримують значну свободу. Згодом температура поступово зменшується спеціальним чином до такого стану, у якому тіло твердне, а його частинки організуються у високоструктуровану ґратку з мінімальною енергією.

З термодинаміки відомо, що ймовірність переходу системи з одного стану в інший обчислюється за формулою Больцмана – Гіббса

$$p = e^{-\frac{\Delta E}{kT}}, \qquad (4.1)$$

де ΔE – зміна узагальненої енергії, T – температура, k – стала Больцмана.

Якщо система перебуває в певному стані, то вона з однаковою ймовірністю може перейти як у вищий, так і в нижчий стани. Однак не виключаються її флуктуації (коливання), але врешті-решт зі зменшенням температури T система "заморожується".

Припустима множина ЗКО розглядається як множина станів: $x = (x_1, ..., x_n) \in X$, а аналогом енергії є значення цільової функції (табл. 4.1). *Стан рівноваги* характеризується певною гармонією, тобто кожна частинка знайшла своє місце і певним чином взаємодіє з іншими.

Таблиця 4.1

Аналоги в термінології

Термодинаміка	Комбінаторна оптимізація
Фізична система	ЗКО
Узагальнена енергія	Значення цільової функції
Температура	Керуючий параметр T алгоритму
Стан/конфігурація	Варіант розв'язку
Метастабільний стан	Локальний розв'язок
Основний стан	Глобальний розв'язок

Імовірність переходу від точки x до сусідньої $y \in O(x)$ на основі (4.1) визначається формулою

$$p = e^{-\Delta/T},$$

де $\Delta = f(y) - f(x)$.

Псевдокод АІВ зображено на схемі 4.2. Початковий етап пошуку (при високій температурі) близький до випадкового блукання, але при зменшенні значень параметра T усе більше наближається до стратегії детермінованого ЛП.

```
procedure SA;
  x: = деякий припустимий варіант розв'язку;
  h: = 0; {h – лічильник ітерацій за температурою};
  T_h: = <початкова температура>;
  x_rec = x;
  while не виконується умова завершення do
    while не досягнута рівновага do
      y: = ГенераціяЧерговоїТочкиОколу O(x);
      Δ = f(y) – f(x);
      p := min{1, e^(-Δ/T_h)}
      ξ: = random[0,1];
      if p ≥ ξ then
          x: = y;
          if f(x) < f(x_rec) then x_rec = x;
      endif
    endwhile;
    h: = h+1; T_h: = <нове значення>;
  endwhile;
  return x_rec;
end
```

Схема 4.2. Псевдокод АІВ

Наведемо ключові моменти реалізації АІВ. При розгляді конкретного алгоритму треба визначити:

1) *Простір розв'язків*.

2) Поняття *околу*.

3) Перебір точок в околі, тобто *спосіб генерації* точок.

4) *Імовірність переходу* від поточного варіанта до нової точки.

5) Принцип *рівноваги*: задаються параметри $\varepsilon > 0$ та v – натуральне, а v переходів у алгоритмі називається п*рогоном*. Тоді, якщо здійснюється k прогонів і маємо значення $f_1,...,f_k$ (характеристики цільових функцій на цих прогонах), то на $(k+1)$-му прогоні вважається досягнутою рівновага, якщо

$$\exists i = \overline{1,k} : \left| f_i - f_{k+1} \right| / f_i \le \varepsilon .$$

6) *Температурний розклад* (правило зміни значення параметра T).

Розклади бувають такими:

- рівномірне (температура змінюється рівномірно);
- помірне спочатку, швидке в кінці;
- прискорене всюди.

При визначенні температурного розкладу значення параметра T найчастіше розраховують за формулою

$$T_{t+1} = \alpha\, T_t ,$$

де $0 < \alpha < 1$; рекомендовано вибирати $\alpha \in [0.5;0.99]$.

7) *Правило зупинки*. Критерій завершення часто пов'язаний з довжиною температурного розкладу, коли задається величина H, а умови зупинки набувають вигляду: кількість проведених ітерацій за температурою h перевищила H.

Інший критерій завершення, що застосовується: при заданій кількості останніх змін температури T рекорд не був покращений.

Приклади дослідження збіжності алгоритму. В алгоритмі імітаційного відпалу перебір точок в околі відбувається випадково для того, щоб можна було перебирати великі околи, але при цьому можна втратити деяку точку з поліпшеним значенням цільової функції.

Нехай маємо параметри керування $T_0, T_1, ..., T_h, ...$ – температурний розклад; відомі правила переходу від T_h до T_{h+1}. Наведемо деякі результати дослідження температурних розкладів.

Теорема 4.1 (С. Джеман, А. Джеман). Якщо температурний розклад задовольняє умови

$$\lim_{h\to\infty} T_h \log h > R >> 0,$$

то завжди

$$\lim_{h\to\infty} p\{x_h \in Arg\min f(x)\} = 1,$$

де x_h – розв'язки, знайдені алгоритмом імітаційного відпалу.

Звідси випливає, що за нескінченну кількість кроків розв'язання задачі завжди можна знайти екстремум.

Теорема 4.1 була підсилена Гайєком (Hajek):

Теорема 4.2. Послідовність $\{T_h\}$ така, що $\lim_{h\to\infty} T_h \log h = R$, є мінімізуючим графіком охолодження тоді й тільки тоді, коли $R > C$, де C – деяка стала.

Гайєк навів алгоритм обчислення сталої C.

Схема АІВ послужила відправною точкою для створення подібних алгоритмів СЛП, серед яких алгоритм глобального рівноважного пошуку та G-алгоритми. Останні розглянемо детальніше.

4.4. Алгоритми прискореного ймовірнісного моделювання: G-алгоритми

Для пояснення ключових принципів роботи цих алгоритмів розглянемо ЗКО без обмежень:

$$x_* = \arg\min_{x\in X} f(x).$$

Досвід застосування АІВ показав, що отримувані результати мають досить високу точність, ефективно реалізуються на БОК. Проте обчислювальні експерименти виявили *суттєві затрати машинного часу* на пошук розв'язків, а також *значні коливання точності* отриманих результатів залежно від вибору значень параметрів алгоритмів, що варіюються.

З метою *зменшення часових затрат і підвищення стабільності результатів* була запропонована сім'я алгоритмів прискореного ймовірнісного моделювання, у яких використовується інша ймовірнісна модель. Вони отримали назву G-алгоритмів. Співвідношення між класами алгоритмів показано на рис. 4.2.

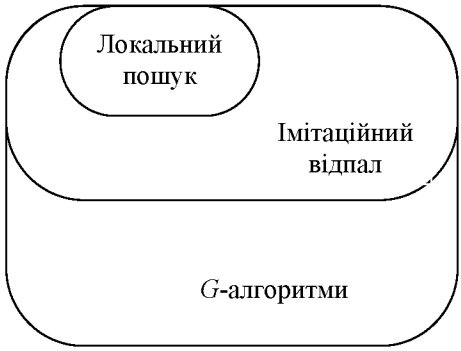


Рис. 4.2. Співвідношення між класами алгоритмів ЛП

Нагадаємо, що в алгоритмах імітаційного відпалу, як це випливає з формули (4.1), при обчисленні ймовірності переходу за формулою $p = e^{-\Delta/T}$ імовірність прийняття чергового варіанта залежить від ступеня зміни цільової функції, абсолютна величина якого масштабується за допомогою параметра T (в АІВ він називається температурою).

Зміну цього параметра (температурний розклад) задають так, щоб у процесі застосування алгоритму ймовірності переходу до гірших варіантів розв'язку зменшувалися б від ітерації до ітерації, прямуючи до нуля. Умови, які визначають необхідність переходу до нового значення параметра T, в АІВ називаються умовами рівноваги.

Наявність такого ймовірнісного механізму створює передумови для виходу на початку пошуку алгоритмів АІВ з локальних екстремумів, а в кінці обчислень сприяє зосередженню пошуку навколо поліпшених локальних розв'язків.

В алгоритмах прискореного ймовірнісного моделювання, які отримали назву *G-алгоритмів*, також здійснюється побудова точок в околі поточного варіанта і поліпшуючі варіанти завжди приймаються як чергове наближення, а варіанти, що відповідають погіршенню (зростанню) цільової функції, теж можуть бути вибрані з деякою ймовірністю. Проте, на відміну від АІВ, значення цієї ймовірності розраховуються однаково впродовж усього обчислювального процесу, але змінюється порогове значення, яке й визначає умови відсіювання погіршуючих варіантів.

Для побудови обчислювального процесу формується строго монотонно зростаюча послідовність дійсних чисел $0 \le \mu_0 < \mu_1 < ... \le 1$, які відіграють роль, у певному розумінні подібну до ролі параметра температури T в АІВ. Якщо x^h – це поточний варіант на кроці h алгоритму, то досліджуваний варіант $y \in O(x^h)$, для якого $f(y) > f(x^h)$, може бути прийнятий як x^{h+1}, якщо розрахована ймовірність $p(x^h, y)$ перевищить поточне значення μ, збільшене на певну випадкову величину.

Нехай γ, $\gamma > 0$, – дійсне число. Задамо для всіх $x, y \in X$ функцію

$$\phi(x, y) = 1 - \frac{f(y) - f(x)}{f(x) \cdot \gamma},$$

а на її підставі – імовірність переходу від точки x до y:

$$p(x, y) = \begin{cases} \min\{1, \phi(x, y)\}, & \phi(x, y) \ge 0, \\ 0 \text{ в іншому разі.} \end{cases} \qquad (4.2)$$

Неважко переконатися, що при поточному варіанті розв'язку x^h імовірність переходу $p(x,y)$ до досліджуваної точки $y \in O(x^h)$, яка визначається формулою (4.2), є кусково-лінійною функцією, що складається з трьох фрагментів, як це показано на рис. 4.3.

Як випливає зі способу обчислення, при $f(y) < f(x^h)$ значення цієї ймовірності дорівнює 1; далі при значеннях $f(x^h) \le f(y) \le (1+\gamma) f(x^h)$ воно спадає від 1 до 0, а при $f(y) > (1+\gamma) f(x^h)$ набуває нульового значення.

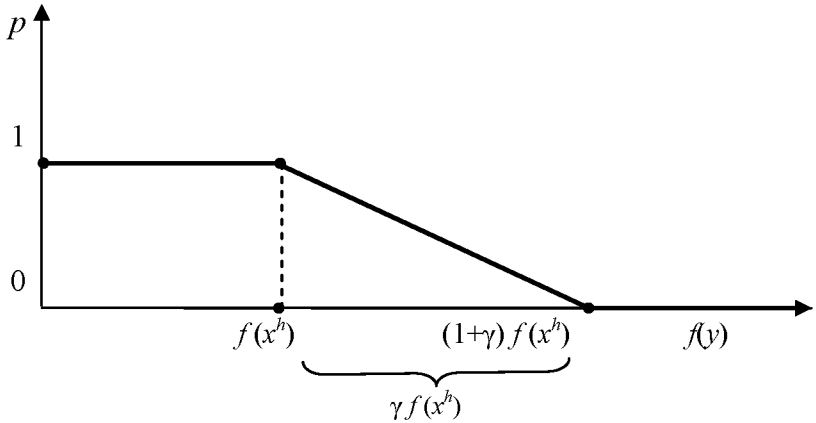


Рис. 4.3. Імовірність переходу до нової точки

Таким чином, значення параметра γ визначає величину середнього інтервалу довжиною $\gamma \cdot f(x^h)$, при потраплянні значень $f(y)$ у який точка y ще може бути вибрана як наступний варіант x^{h+1}.

Зауважимо, що довжина зазначеного інтервалу автоматично зменшується при спаданні значень цільової функції $f(x^h)$, що відбувається при наближенні пошуку до точніших варіантів розв'язку задачі мінімізації, і збільшується в протилежному випадку. Крім того, використання відносних величин дає змогу уникнути залежності від абсолютних значень цільової функції. Тим самим здійснюється адаптація обчислювального процесу до динаміки зміни значень (ландшафту) цільової функції задачі.

Псевдокод G-алгоритмів для розв'язання задач вигляду (4.1) можна подати схемою 4.3. Тут *random*[0,1] – датчик випадкових чисел з відрізка [0,1].

procedure G_Search(x)
  $x^0 :=$ деякий припустимий варіант розв'язку;
  $h := 0$; $t := 0$; $\mu_0 := 0$; $x_{rec} := x^0$;
  while *окіл поточного варіанта* $O(x^h)$ *не переглянутий повністю* **do**
    while *не виконана умова рівноваги* **do**
      $y :=$ ГенераціяЧерговоїТочкиОколу $O(x^h)$;
      Обчислення $\varphi(x^h, y)$;
      $p := p(x^h, y)$; $\xi := \mu_t + random[0,1] \cdot (1 - \mu_t)$;
      if $p \geq \xi$ **then**
        $h := h + 1$; $x^h := y$;
        if $f(x_{rec}) > f(x^h)$ **then** $x_{rec} := x^h$;
      endif;
    endwhile;
  ФормуванняЧерговогоЗначення μ_{t+1};
  $t := t + 1$;
  endwhile;
  return $x = x_{rec}$;
end

Схема 4.3. Псевдокод *G*-алгоритмів

Ключові аспекти реалізації. Побудова конкретного G-алгоритму здійснюється шляхом уточнення таких аспектів обчислювальної схеми:

- механізму побудови послідовності $\{\mu_t\}$;
- способів генерації точок з околу поточного варіанта;
- умов рівноваги при даному значенні величини μ;
- правила зупинки.

На сьогодні найчастіше застосовуються два способи породження послідовності $\{\mu_t\}$: шляхом визначення й використання деякої строго монотонної функції, яка зазвичай позначається через G (звідси скорочена назва – G-алгоритми), а також на основі використання такої функції у поєднанні із застосуванням відомого принципу золотого перерізу.

Нехай G: $[0,1]\rightarrow[0,1]$ – задана строго монотонна функція; тоді в G-алгоритмах можна покласти

$$\mu_{t+1} = G(\mu_t).$$

За функції G на практиці часто вибирають функції вигляду

$$G_k(x) = (x^{1/k} + b)^k,\ x \geq 0,$$

де $k \in \{1, 2, 3\}$, b, $0 < b < 1$, – деяка мала величина.

За аналогією з АІВ можна покласти

$$G_k(x) = \alpha(1 - x)^k,\ x \geq 0.$$

Значення μ_t можуть формуватися шляхом поділу відрізка $[0,1]$ з деяким кроком і послідовного вибору побудованих точок. Зауважимо, що в разі скінченності зазначеної послідовності після досягнення максимального значення (одиниці) алгоритм трансформується в локальний пошук.

Виявилося ефективним використати правило золотого перерізу, яке, хоча й зовсім іншим чином, застосовується також при побудові алгоритмів пошуку екстремумів неперервних функцій одного аргументу. У нашому випадку визначимо величину u_t як меншу (ліву) із двох точок, що здійснюють золотий переріз відрізка $[\mu_t, 1]$ (за умови, що $\mu_0 = 0$, а $\mu_{t+1} = G(u_t)$). Отже, правило золотого перерізу задає швидкість прямування лівої межі цього відрізка до 1, а відповідні величини μ_t є аргументами вибраної функції $G(x)$. Такі алгоритми отримали назву *GS-алгоритми*.

При розробці умов рівноваги, ураховуючи аналогію між параметром "температура" в АІВ і μ, доцільно використовувати досвід, накопичений при створенні алгоритмів АІВ. Зокрема, рівновага може визначатися через прогони: задаються певний натуральний параметр v і дійсне число $\varepsilon > 0$, а виконання v переходів називають *прогоном*.

Якщо при даній температурі (значенні параметра μ) виконано k прогонів і отримані характеристичні значення $f_1,\ldots,f_k$, то вважають, що досягнуто рівновагу, якщо в $(k+1)$-му прогоні для деякого $i \in \{1,\ldots, k\}$ виконується нерівність

$$|f_i - f_{k+1}| / f_i \le \varepsilon. \qquad (4.3)$$

Характеристична величина f_i для прогону може бути, наприклад, середнім або найкращим значенням цільової функції серед розв'язків, які утворюють цей прогін.

Правилом зупинки може бути:

1) закінчення перебору всіх точок в околі без реалізованого переходу в нову точку;

2) обмеження за тривалістю роботи алгоритму;

3) досягнення необхідної точності – при відомій нижній межі цільової функції;

4) порівняння різниці максимального й мінімального значень цільової функції з максимальним значенням зміни цієї функції при даному значенні μ: якщо це співвідношення прямує до одиниці, то обчислення завершуються.

Слід зазначити, що в обчислювальній схемі G-алгоритму досить просто врахувати багато типів обмежувальних умов, які часто зустрічаються на практиці: відповідні умови перевірки на припустимість можуть бути враховані при породженні точок з околу поточного варіанта. Це дає змогу розв'язувати не одну, а відразу цілий підклас задач, причому обмеження й деякі дані задачі, що розв'язується, можуть змінюватися в процесі її розв'язання.

Нарешті, як у повторюваному локальному пошуку, після завершення роботи загальної схеми G-алгоритму можна використати процедуру збурення знайденого локального розв'язку, а отриманий у результаті збурення варіант використовувати як початковий розв'язок для вбудованого G-алгоритму, породжуючи тим метод повторюваних G-алгоритмів.

Отже, *ідеологія* методів прискореного ймовірнісного моделювання полягає в тому, що на кожному кроці, маючи $x^0, x^1, \ldots$ як центри, будуємо точки їх околу із заданим радіусом і дозволяємо перехід не лише до поліпшуючих (за значенням цільової функції), але й до погіршуючих точок з певною ймовірністю. З часом ця ймовірність переходу до гірших точок зменшується (якщо в алгоритмах імітаційного відпалу T прямує до нуля, а в G-алгоритмі штучно підводимо поріг до 1). Таким чином, на завершальних ітераціях цей метод у наведеній вище інтерпретації фактично перетворюється на стандартний локальний пошук.

До переваг G-алгоритмів варто віднести:

- зменшення часу на пошук розв'язку (щодо імітаційного відпалу), звідси – можливість розв'язувати задачі підвищеної розмірності;
- зменшення залежності від початкового наближення;
- застосовність до розв'язання задач із різними обмежувальними умовами.

Дослідження умов збіжності. Позначимо E_* множину всіх глобальних розв'язків $Arg\,min\, f(x)$, тобто

$$E_* = \{x \in X : f(x) \le f(y), \forall y \in X\}.$$

Справедливими є такі твердження.

Лема 1. Нехай $\{x^s\}$ – послідовність розв'язків, побудована G-алгоритмом.

Тоді існують числа $\hat{p}^s$ такі, що $\sum_{t=0}^{\infty} \hat{p}^t < \infty$,

а

$$p^s = p(x^s, y) \le \hat{p}^s \quad p^s = p(x^s, y) \le \hat{p}^s, \ \forall y \in O(x^s).$$

Лема 2. Нехай $\{\omega^s\}$ – обмежена числова послідовність, для якої виконується умова

$$\omega^{s+1} \le \omega^s + \eta^s,$$

де $\eta^s \ge 0, \sum_{t=0}^{\infty} \eta^t < \infty.$

Тоді існує $\lim\limits_{s\to\infty} \omega^s$.

Використовуючи ці дві леми, можна довести таку теорему.

Теорема 4.3. Нехай $\{x^s\}$ – послідовність, яка побудована G-алгоритмом. Тоді

$$\exists\, x^* \in E_*: x^s \to x^* \text{ за ймовірністю.}$$

Нагадаємо, що збіжність за ймовірністю визначається таким чином:

$$\lim_{s\to\infty} p\{x^s = x^*\} = 1.$$

Як показує досвід застосування G-алгоритмів, значення їх ключових параметрів у багатьох ЗКО доцільно вибирати в таких діапазонах (або ж починати з них пошук кращих значень):

$$\nu \sim 20\text{–}50;\ \gamma \sim 1.4\text{–}1.7;\ k = 2,$$

тобто використовувати функцію $G(z) = z^2$ при значеннях $b = 0.01$.

Умову рівноваги виявилось ефективним формулювати через прогони (див. вище), вибираючи у формулі (4.3) значення параметра ε з відрізка [0.01;0.06].

Програмна реалізація алгоритмів цього класу застосовувалася для розв'язання ЗКО з різних класів, зокрема ЗК різної розмірності, і квадратичних задач про призначення, що генерувались за допомогою датчика випадкових чисел для значень n від 15 до 100, а також реальних задач, що виникали при проектуванні радіоелектронних приладів та іншої цифрової апаратури.

Обчислювальний експеримент із розв'язання квадратичних задач про призначення показав, що для параметрів алгоритму ефективні значення перебували в діапазонах: $1.3 < \gamma < 1.5$, $0.01 < \varepsilon < 0.06$. Досвід застосування розроблених алгоритмів дозволив також зробити висновок, що запропоновані G-алгоритми досить легко можуть бути адаптовані до розв'язання широкого кола ЗКО, а також ефективно використовуватися для розробки гібридних метаевристик як підлеглих процедур.

Отже, алгоритми детермінованого й стохастичного ЛП, незважаючи на відносну простоту своєї парадигми і тривалу історію, продовжують бути ефективним інструментом при створенні інформаційних технологій розв'язання багатьох практичних ЗКО як шляхом автономного застосування (особливо при розв'язанні задач підвищеної розмірності та складності), так і при використанні як "будівельних" блоків при створенні сучасних метаевристик і гіперевристик, зокрема з розпаралелювання обчислювального процесу. У багатьох випадках вони органічно доповнюють популяційні метаевристики, про які йтиметься далі.

5. ГЕНЕТИЧНІ АЛГОРИТМИ

5.1. Загальні відомості

Генетичні алгоритми (ГА, genetic algorithm) виникли на основі спостереження за природними процесами еволюції та селекції популяцій живих істот – особин певного виду – і моделювання їх принципів.

При описі ГА зазвичай використовують терміни, запозичені з генетики.

Популяція – це скінченна множина особин.

Особини, що входять до популяції, у ГА подаються хромосомами із закодованими в них множинами параметрів задачі, головним чином розв'язків, які інакше називаються *точками в просторі пошуку* (search points) задачі.

Хромосоми (інші назви – ланцюжки або кодові послідовності) – це впорядковані послідовності генів.

Ген (який також називається властивістю, знаком чи детектором) – це атомарний елемент генотипу, зокрема хромосоми.

Генотип, або структура – це набір хромосом даної особини. Отже, особинами популяції можуть бути генотипи або одиничні хромосоми.

Фенотип – це сукупність зовнішніх і внутрішніх ознак, які відповідають даному генотипу, тобто декодована структура, або множина параметрів задачі (розв'язок, точка простору пошуку).

Алель – це альтернативне значення конкретного гена, також визначається як значення властивості або варіант властивості.

Локус (*позиція*) – указує на місце розміщення даного гена в хромосомі (ланцюжку амінокислот). Множина позицій генів – це локи.

Функція пристосованості (fitness function), яка ще називається *функцією корисності* чи *придатності*, виражає міру пристосованості особин у популяції. Ця функція дозволяє оці-

нити ступінь пристосованості конкретних особин у популяції й вибрати з них найбільш придатні відповідно до еволюційного принципу виживання найсильніших. У задачах оптимізації функція пристосованості зазвичай формується на основі цільової функції й оптимізується.

Процес появи в гені нових рис або підсилення вже наявних сильних рис, відображених у алелях, під впливом зовнішнього середовища називається *мутацією.* Основну схему еволюційного відбору, покладену в основу розробки ГА, наведено на рис. 5.1.

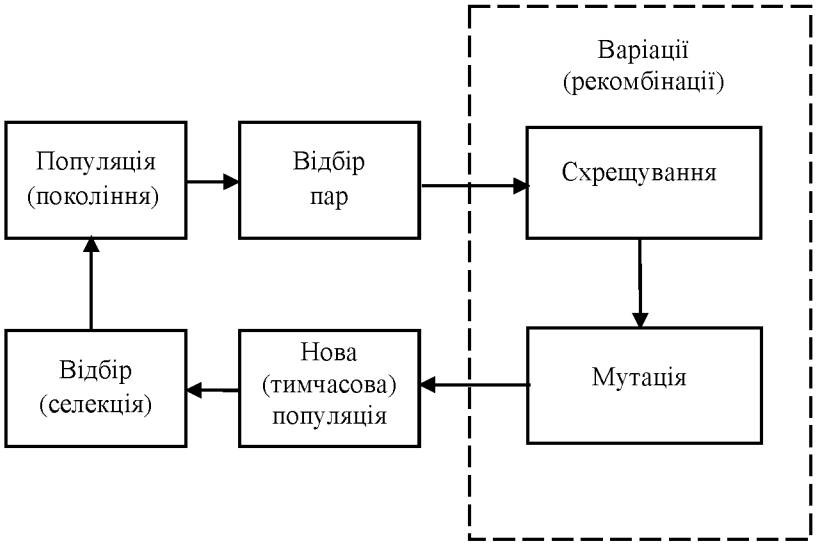


Рис. 5.1. Схема еволюційного відбору в ГА

Отже, *генетичний алгоритм* – це еволюційний алгоритм пошуку, що використовується для розв'язання задач оптимізації, моделювання чи підтримки прийняття рішень шляхом послідовного відбору, комбінування й варіації шуканих параметрів з використанням механізмів, що нагадують біологічну еволюцію.

Загальна ідея ГА – перенесення принципу еволюційного відбору на процес оптимізації. Мета таких алгоритмів – підвищення загального рівня пристосованості популяції.

Ця схема була покладена в основу еволюційних обчислень. Виділяють такі основні класи еволюційних алгоритмів (рис. 5.2):

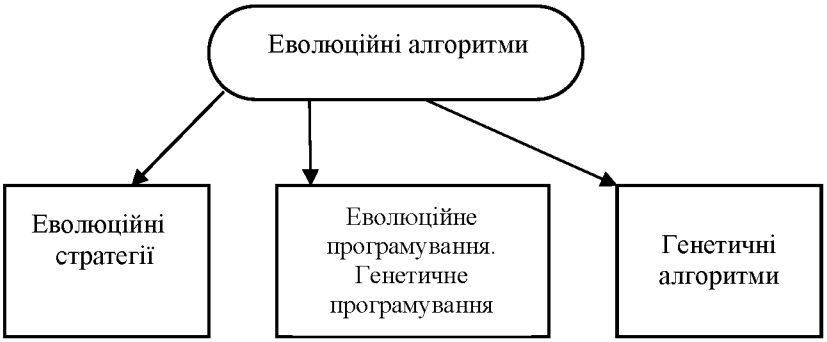


Рис. 5.2. Класи еволюційних алгоритмів

Еволюційні стратегії – це алгоритми розв'язання задач оптимізації з неперервними (континуальними) змінними.

Еволюційне програмування (генетичне програмування) об'єднує алгоритми, які спрямовані на оптимізацію структури програмних комплексів і систем.

Одне із основних застосувань ГА – розв'язання ЗКО, розпізнавання та прогнозування, хоча вже автор цих алгоритмів проф. Дж. Голанд розглядав їхнє застосування для формування вирішувальних правил.

Оскільки вся інформація про об'єкт міститься у хромосомних наборах, то для розв'язання оптимізаційних задач необхідно подати кожну ознаку об'єкта в закодованому вигляді. Найпоширенішим типом кодування є побітове кодування, при цьому кожній ознаці (алелі) у фенотипі відповідає один ген у генотипі. У цьому випадку ген розглядається як послідовність із нулів та одиниць, а хромосома може бути подана у вигляді булевого вектора, наприклад $x = (0110\ 1110\ 1110)$.

Часто для кодування хромосом використовують систему кодів Ґрея, у якій два послідовні коди відрізняються значенням лише одного біта. Порівняльну таблицю видів двійкового кодування характеристик особин подано нижче.

Види кодування характеристик особин

Число	Звичайні коди	Код Ґрея
0	0000	0000
1	0001	0001
2	0010	0011
3	0011	0010
4	0100	0110
5	0101	0111
6	0110	0101
7	0111	0100
8	1000	1100
…	…	…
15	1111	1000

На прикладі кодувань чисел 7 і 8 бачимо, що природно сусідні два числа у звичайному двійковому коді можуть мати значні відмінності. Для кодів Ґрея важливо те, що два сусідні числа мінімально відмінні (тобто відстань Гемінга буде одиничною).

У задачах цілочислової оптимізації кожна компонента вектора розв'язків (ціле число) кодується відповідною до розрядності кількістю булевих чисел.

Якщо маємо вектор з неперервними компонентами, то його компоненти наближуємо булевими числами. Для цього припустимо, що діапазоном зміни кожної компоненти вектора розв'язку є відрізок [0,1]. Ділимо його на 256 проміжків і кожному такому проміжку приписуємо відповідний номер та значення, що дорівнює його середині.

5.2. Обчислювальна схема традиційного ГА

Загальний підхід, що моделює процес еволюції в живій природі та використовується в ЕО, подано на рис. 5.3.

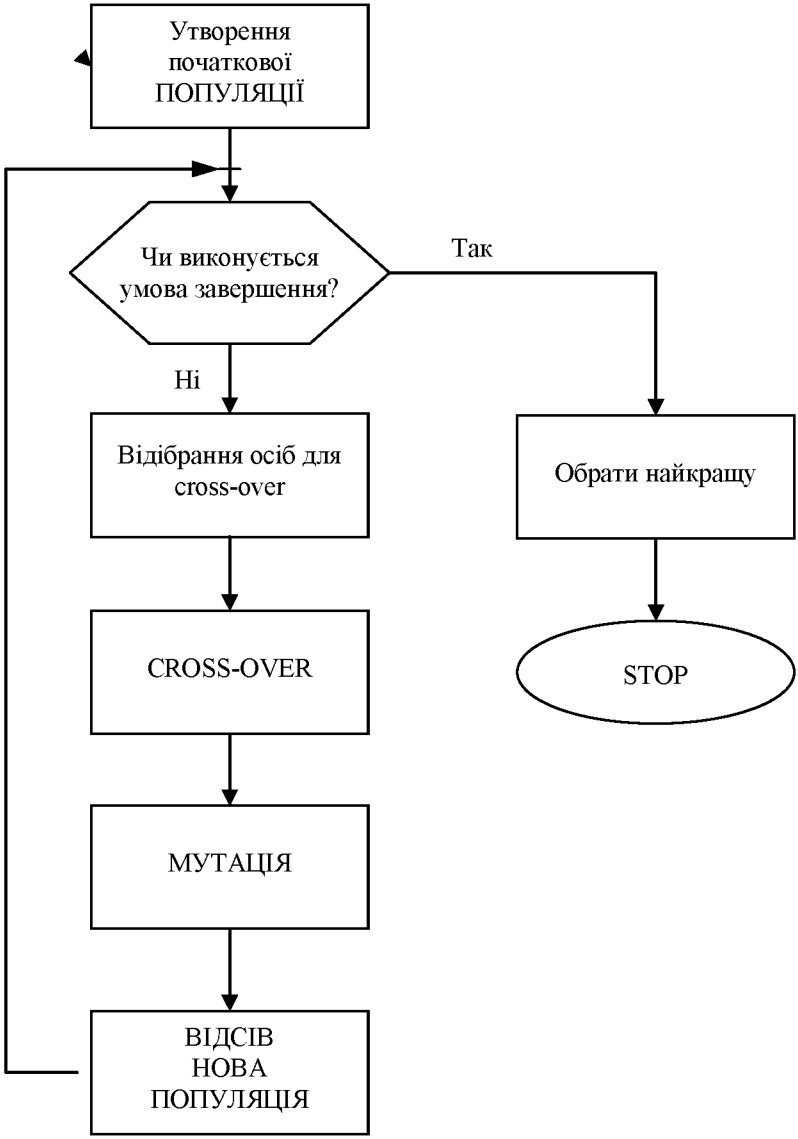


Рис. 5.3. Блок-схема еволюційних алгоритмів

У ГА ця схема формалізується таким чином (схема 5.1):

```
procedure EA(x);
begin
g: = 0;                    {g – лічильник поколінь}
ФормуванняПочатковоїПопуляції P^0; {m розв'язків}
Оцінювання особин із P^0; {обчислення фітнес-функції}
repeat
P̂ : = ВідбірДляВаріації (P^g);
P : = Схрещення (P̂ );                    {кросовер}
P : = Мутація (P, умова);
Оцінювання особин із P;
P^(g+1): = ВідбірПопуляції (P^g, P);      {селекція}
g: = g+1;
until виконається умова завершення;
x: = arg min {f(y): y∈P};
return x;
end
```

Схема 5.1. Псевдокод ГА

Розробка ГА для розв'язання певної ЗКО вимагає конкретизації таких аспектів:

- принципів формування початкової популяції;
- механізмів відбору;
- вибору типу кросовера;
- вибору оператора мутації;
- умови завершення.

Формування початкової популяції. При *виборі початкової популяції* часто застосовують процедури породження випадковим чином m варіантів розв'язку ЗКО, де m – параметр алгоритму, який задається.

При такому формуванні використовуються чотири основні принципи:

1) *дробовик* – випадковий вибір з усього простору розв'язків X;

2) *фокусування* – випадковий вибір із заданої підобласті X;

3) *ковдра* – генерування повної популяції, що включає всі можливі розв'язки із заданої підобласті X;

4) *комбінування* – сумісна реалізація всіх трьох попередніх принципів.

Критерії завершення роботи ГА. Найчастіше використовуються такі умови завершення обчислень у ГА:

1. Перевищення заданої кількості H ітерацій (змін поколінь/популяцій), H – параметр алгоритму.

2. Відсутність (або невелика зміна) поліпшення наявного рекорду чи середньої пристосованості популяції на кількох останніх ітераціях.

3. Вичерпання заданого часу на обчислення.

Далі розглянемо загальні підходи до конкретизації основних етапів ГА: *відбору*, *мутації* та *рекомбінації* (*схрещування*), які вимагають детальнішого висвітлення.

У ГА (як і в інших еволюційних алгоритмах) може бути виділено дві *форми відбору* (див. схему 5.1). У першій, *відборі для варіації*, особини вибираються для рекомбінації та/або мутації. У другій, *відбір популяції*, особини відбираються для нового покоління – це інколи називається *заміною*, оскільки деякі або всі батьки замінюються деякими або всіма нащадками.

5.3. Стратегії відбору для варіації

При *виборі пропорційно пристосованості* ймовірність вибору деякого варіанта x_i із поточної популяції P у більшості ГА задається або формулою

$$p(x_i) = 1 - \frac{f(x_i)}{\sum\limits_{x_j \in P} f(x_j)}, \tag{5.2}$$

або

$$p(x_i) = 1 - \frac{f(x_i)}{\bar{f}}, \tag{5.3}$$

де $\bar{f}$ – максимальне значення цільової функції на популяції.

Зрозуміло, що тут, як і раніше, припускаємо строгу додатність цільової функції.

Вибір пропорційно придатності може реалізовуватися імітацією *колеса рулетки*. На колесі рулетки кожній особині виділяється така частина колеса (дуга), довжина якої пропорційна значенню його придатності.

Недоліком пропорційного методу відбору є те, що зі зменшенням варіацій значень придатності в популяції вибір наближається до випадкового. Щоб зменшити залежність від розкиду значень придатності, було запропоновано здійснювати *вибір на основі рангу*. У лінійній моделі ранжування ймовірність вибору варіанта x_i задається формулою

$$p(x_i) = p_{\max} - (p_{\max} - p_{\min})\frac{i-1}{m-1},$$

де m означає розмір популяції, а $p_{\min}$ та $p_{\max}$ $(0 < p_{\min} < p_{\max} < 1)$ – відповідно мінімальну й максимальну ймовірність вибору (останні є параметрами методу). У деяких випадках для розрахунку $p(x_i)$ використовують нелінійні функції.

Третій спосіб вибору для варіації – це *турнірний відбір*. На кожному кроці k особин популяції спочатку відбираються випадково (незалежно від придатності), а потім вибирається краща з них – k є параметром методу.

Нарешті, інколи згадується й найпростіший *вибір – з рівною ймовірністю* серед усіх особин чи лише серед тих, пристосованість яких більша за середню по популяції.

Стратегії відбору для виживання. У еволюційних обчисленнях часто вживають позначення, які запозичені з еволюційних стратегій: μ – кількість батьків, λ – кількість нащадків.

1) Найпростіша форма – це *заміна покоління*, у якій усі батьки змінюються їх потомством ($\mu = \lambda$). Цей метод використовувався в традиційних ГА в комбінації з пропорційним придатності відбором для варіацій, щоб змінити тиск відбору.

2) У *відборі стійких станів* (*steady state selection*) кількість нащадків, утворюваних при варіації, менше кількості батьків ($\mu > \lambda$). Тому вимагається, щоб у стратегію були закладені механізми визначення, які батьки повинні бути замінені.

Існує кілька варіантів, наприклад, *заміна найгіршого варіанта* (*особини*) і *заміна найстарішого*.

3) (μ, λ)-*відбір*: μ батьків замінюються кращими із λ нащадків, $\lambda \geq \mu$. Вплив відбору може збільшуватися шляхом збільшення кількості нащадків λ.

4) ($\mu + \lambda$) *відбір*: з тимчасової популяції, що містить μ батьків і λ нащадків, вибирається μ кращих особин.

Існують інші способи відбору, які використовуються в комбінації з наведеними вище стратегіями.

Найважливіші з них:

- принцип *елітизму*, який полягає в тому, що одна чи кілька кращих особин завжди без відбору переходять у наступне покоління;
- *перевірка дублікатів* – утворені нащадки проходять перевірку на ідентичність батькам, чим виключається потрапляння однакових розв'язків у нове покоління. Цей принцип особливо важливий у еволюційних алгоритмах із невеликими розмірами популяції.

За будь-якої стратегії рекомендується при зміні покоління застосовувати вилучення 10 % найгірших варіантів (правило елімінації). Інколи суттєво збільшують частку особин, які потрапляють у наступне покоління згідно з принципом елітизму – аж до застосування правила 80 : 20, тобто 80 % нової популяції заповнюється найкращими особинами з тимчасової популяції.

5.4. Оператори рекомбінацій для задач з булевими змінними

Еволюційні оператори варіації – рекомбінація й мутація – залежать від способів кодування варіантів розв'язку задачі оптимізації. Оператори схрещування утворюють нащадків шляхом комбінування рис, які притаманні батькам, у той час як ідейна роль операторів мутації – вносити нові риси, які можуть бути відсутніми в усій популяції. Вибір типу кросовера та способи досягнення компромісу між близькістю й віддаленістю від батьків залежать від специфіки ЗКО і суттєво впливають на ефективність ГА.

Ці вимоги формалізуються часто так: якщо $x, y \in X$ – батьки, а $z \in X$ – нащадок, то кросовер має задовольняти нерівності

$$d(x,z) \le d(x,y),\ d(y,z) \le d(x,y).$$

Спочатку розглянемо ці оператори для бінарного кодування, яке було запропоноване ще Дж. Голландом і яке часто використовують у застосуваннях ГА. Нехай $x,\ y \in \{0,1\}^n$ – бітові рядки довжиною n, що зображують варіанти розв'язку.

Оператори схрещування

Одноточковий кросовер. Дія цього оператора полягає в розрізанні рядка бітів надвоє у випадково вибраній точці перерізу t (координаті). Після цього початок першої (другої) хромосоми з'єднується з хвостом (початком) другої. Таким чином, одноточковий кросовер породжує два розв'язки – бінарні вектори x' та y' з координатами

$$x_i' = \begin{cases} x_i, \text{ якщо } i \le t, \\ y_i, \text{ якщо } i > t \end{cases}$$

та

$$y_i' = \begin{cases} y_i, \text{ якщо } i \le t, \\ x_i, \text{ якщо } i > t. \end{cases}$$

Двоточковий кросовер. На відміну від одноточкового кросовера, наявні дві хромосоми варіанта розв'язку розрізають у двох випадково вибраних точках t_1 та t_2 ($t_1 < t_2$), що зумовлює утворення трьох фрагментів. Тому два розв'язки x' та y' генеруються згідно з умовами:

$$x_i' = \begin{cases} y_i, \text{ якщо } t_1 \le i \le t_2, \\ x_i \text{ в іншому разі} \end{cases}$$

та

$$y_i' = \begin{cases} x_i, \text{ якщо } t_1 \le i \le t_2, \\ y_i \text{ в іншому разі.} \end{cases}$$

Існує узагальнення цього оператора, у якому рядки бітів розрізають у k випадково вибраних точках; він називається *k-точковим кросовером*.

Узагальнений кросовер (*Uniform Crossover*) використовує маску кросовера, щоб дозволити альтернативні форми схрещування. Маска кросовера M – це просто рядок бітів тієї самої довжини, що й вектор розв'язку. Значення кожного біта в масці M_i визначає для кожного відповідного гена-нащадка, від якого батька він отримає своє значення:

$$x_i' = \begin{cases} y_i, \text{ якщо } M_i = 1, \\ x_i \text{ в іншому разі} \end{cases}$$

та

$$y_i' = \begin{cases} x_i, \text{ якщо } M_i = 1, \\ y_i \text{ в іншому разі.} \end{cases}$$

Одно- та двоточкові кросовери є особливими випадками узагальненого кросовера. Порівняно з узагальненим кросовером, 1-бітовий є рівномірно розподіленим по масці зі значенням перебування на кожному місці з імовірністю 0.5.

Рівномірний кросовер: задається ймовірність p і значення чергової компоненти з імовірністю p береться з першого батька, а з імовірністю $(1–p)$ – з другого.

Оператори мутації

Інвертування біта. Оператор *інвертування* (або дзеркального відбиття біта) дзеркально відображає невелику кількість генів у геномі – наприклад один біт:

$$x = 01101\underline{\mathbf{1}}001 \rightarrow x' = 01101\underline{\mathbf{0}}001$$

Є два способи реалізації такого оператора мутації. У першому задаються розряди (координати), у яких кожний біт у геномі інвертується. Інший шлях – задається кількість бітів для віддзеркалення в геномі, а їх розташування в хромосомі вибирають довільно.

Альтернативний оператору мутації – оператор *інверсії*: вибираються два номери компонент хромосоми і середній фрагмент (від першого до другого номера) повністю "розвертається", тобто записується у зворотному порядку. Наприклад, вибрано компоненти 3 та 8, тоді застосування оператора інверсії до $x = 01|\mathbf{101100}|1$ викличе утворення хромосоми $x' = 01|\mathbf{001101}|1$.

Оператори мутації відіграють вторинну роль у ГА. Їх часто використовують як фонові оператори, щоб додати джерело різноманітності з метою запобігання передчасній збіжності. Мутація зазвичай застосовується до нащадків, які генеровані кросовером, перед оцінюванням пристосованості.

5.5. Особливості реалізації еволюційних операторів у просторі перестановок

Розглянемо найвідоміші оператори варіації у випадку, коли простір розв'язків ЗКО X – це простір перестановок

$$P_n = \left\{ x = (x_1, ..., x_n) : x_i \in \{1, ...n\}, x_i \neq x_j, \forall i, j \right\}.$$

Нехай маємо дві довільні перестановки: $x = (x_1, \ldots, x_n) \in P_n$ та $y = (y_1, \ldots, y_n) \in P_n$, які вибрані як батьки. Зазвичай при виконанні операторів рекомбінації один з таких батьків оголошується базовим, а потім ролі міняються, щоб отримати ще одного нащадка.

Оператори схрещування

1. Частково відповідний кросовер PMX (Partially-Mapped Crossover).

1) Задаємо чи генеруємо випадково (наприклад, з імовірністю $\frac{1}{n}$), два числа: k_1 та k_2, $1 \le k_1 < k_2 \le n$.

Виділяємо фрагмент від k_1 до k_2 у батьках x, y, який називається *сегментом відповідності* (mapping section).

2) Вибираємо базовим варіант x, на основі якого формуємо нащадка z^x:

а) переносимо в нього виділений сегмент відповідності з x, розміщуючи його на тому самому місці, де він містився в перестановці x;

b) поза сегментом відповідності вибираємо чергову компоненту y з мінімальним номером; якщо її немає в побудованому на даний момент фрагменті z^x, то записуємо її на те саме місце, що й в y.

Якщо ця компонента вже є в побудованому фрагменті z^x, то:

– шукаємо її в перестановці x і беремо відповідну компоненту з y. Якщо її теж уже вибрано, то продовжуємо аналогічний пошук, будуючи ланцюг, доки не знайдемо компоненту, що не збігається, яку й переносимо в побудований фрагмент нащадка z^x.

Побудова завершується, коли фрагмент z^x стає перестановкою.

3) Вибираємо базовою перестановку y і аналогічно формуємо нащадка z^y.

Приклад.

Нехай вибрано двох батьків і дві точки розрізу $k_1 = 4,\ k_2 = 7$.

$$x = (10,9,6 \overset{k_1}{|}\ 5,3,7,8 \overset{k_2}{|}\ 1,4,2);$$

$$y = (10,5,3 |\ 7,4,1,8 |\ 2,6,9)$$

Тоді утворюються такі нащадки:

$$x' = (10,\ 9,\ 6 \mid 7,\ 4,\ 1,\ 8 \mid 5,\ 3,\ 2);$$

$$y' = (10,\ 1,\ 4 \mid 5,\ 3,\ 7,\ 8 \mid 2,\ 6,\ 9).$$

2. Частково відповідний порядковий кросовер PMOX (Partially-Mapped Order Crossover).

1) Задаємо:

– параметр k, $k < n$, – кількість компонент базового варіанта, які переносяться до нащадка без змін;

– p^x, p^y – імовірності, що можуть відображати переваги батька x чи y ($p^x, p^y \in (0,\ 1)$).

2) Вибираємо x як базовий варіант і формуємо нащадка $z = (z_1, \ldots, z_n)$ так:

a) Покладаємо $z_1 = x_i,\ i = 1, \ldots, k$.

b) формуємо впорядковані послідовності компонентів $\mathcal{X} = (x_{k+1},\ \ldots,\ x_n)$ та $\mathcal{Y} = (y_{j1}, \ldots, y_{jn-k})$, $y_i \in \mathcal{X}$, де в послідовності $\mathcal{Y}$ зберігається таке саме відносне впорядкування, як і в перестановці y;

с) для $i = k+1,...,n$ значення z_i з імовірністю p^x покладаємо рівним першому елементу упорядкування X, а з імовірністю $(1-p^x)$ – з упорядкування Y.

Вибраний елемент вилучається з обох упорядкувань X та Y. Ця процедура повторюється до побудови повної перестановки $z = (z_1,...,z_n)$.

3) Вибираємо як базового батька y і процес повторюємо.

Інколи задають діапазон значень k_1, k_2, $1 \le k_1 < k_2 \le n$, і в п. а) переноситься від базового варіанта фрагмент від k_1 до k_2, а не від 1 до k, як записано вище. Зрозуміло, що це загальніший випадок і в цьому разі множина X набуде дещо іншого вигляду, а саме черги з вилученими елементами, що ввійшли у виділений фрагмент.

3. Порядковий кросовер OX (Order Crossover). Є фактично видозміною кросовера PMOX.

1) Задаємо параметри k_1, k_2, $1 \le k_1 < k_2 \le n$.

2) Вибираємо базовий варіант – наприклад x.

3) З базового варіанта до нащадка переноситься фрагмент від k_1 до k_2 на ті самі позиції.

4) Із другого батька y вилучаються всі компоненти, які є в зазначеному фрагменті, а решта компонент утворює впорядковану послідовність Y.

5) Елементи сформованої послідовності Y заносяться в порядку черги на ті місця нащадка, які не заповнені, починаючи з компоненти k_2+1, доки не доходимо до кінця перестановки – компоненти n. Після цього заповнення продовжується з першої компоненти до k_1-1, чим і завершується процес формування нащадка.

6) Обираємо тепер батька y як базового, і процес повторюється.

4. Максимально зберігаючий кросовер MPX (Maximal Preservative Crossover). Ще одна схема, що подібна до кросоверу PMOX і є, по суті, модифікацією OX.

1) Задаємо чи випадково генеруємо параметри k_1, k_2, $1 \le k_1 < k_2 \le n$.

2) Вибираємо базовий варіант – наприклад x.

3) З базового варіанта до нащадка переноситься фрагмент від k_1 до k_2 на перші позиції, тобто з номерами від 1 до $k_2 - k_1 + 1$.

4) Із другого батька y вилучаються всі компоненти, які є в зазначеному фрагменті, а решта компонент утворює впорядковану послідовність Y.

5) Елементи сформованої послідовності Y заносяться в порядку черги на ті місця нащадка, які не заповнені, починаючи з компоненти $k_2 + 1$, доки процес формування нащадка не завершиться повністю.

5. Циклічний кросовер CX (Cycle Crossover). Формує нащадка з максимальним збереженням місць компонентів батьків. Для цього розглядаються цикли в підстановках $\frac{x}{y}$ та $\frac{y}{x}$.

1) Вибираємо базовий варіант. Нехай це спочатку x.

2) Формуємо вектор номерів циклів $c = (c_1, \ldots, c_n)$. Покладаємо $c_i = 0$, $i = 1,\ldots,n$.

3) Покладаємо $k = 0$, де k – номер поточного циклу, що використовується для схрещування.

4) $k := k + 1$.

5) З базового батька x серед тих компонент, які ще не вибиралися, вибираємо j-ту компоненту x_j (довільно чи з мінімальним номером), чим започатковуємо початок циклу.

6) Факт вибору компоненти x_j відображається у векторі циклів: $c_j = k$.

7) Знаходимо в x компоненту i таку, що $x_i = y_j$, і перепозначаємо: $j := i$.

8) Якщо x_j ще немає в k-му циклі, то повторюємо дії, починаючи з п. 6, інакше переходимо до п. 9.

9) Цикл k сформовано. Довільно з x та y вибираємо одного батька й копіюємо з нього всі елементи з циклу k до нащадка z:

$$z_i = b_i \text{ для всіх } i \text{ таких, що } c_i = k,\ i = 1,\ldots,n,$$

де b_i – компоненти вибраного батька.

Якщо ще є невибрані компоненти: $\exists\ i \in \{1, \ldots, n\}$: $c_i \neq 0$, то повторюємо пп. 4–9, інакше – СТОП.

У деяких роботах показано, що абсолютна позиція в середньому половини елементів обох батьків зберігається. Це дало деяким авторам підстави зробити висновок з теоретичних та емпіричних результатів, що оператор СХ дає кращі результати для ЗК, ніж оператор РМХ. Утім цей висновок потребує подальших досліджень.

Оператори мутацій

Процедури мутації можуть здійснюватися за однією з альтернативних схем на основі:

1. Транспозицій. Задаємо значення параметра L, L – деяке – натуральне число, і здійснюємо L транспозицій $\omega^{ij}x$, де i, j – випадкові індекси транспозиції.

2. Інверсій. Виділяється фрагмент перестановки, наприклад шляхом випадкового вибору номерів першої та останньої компоненти, і виділені компоненти записуються у зворотному порядку.

3. Вставлення фрагмента. Випадковим чином виділяється фрагмент, який вставляється в іншому місці перестановки.

4. Циклічного зсуву. Довільно генерується значення $i, j, k \in \{1,...,n\}$ і здійснюється циклічний зсув компонент: $i \to j \to k \to i$.

5. Використання одного кроку алгоритму локального пошуку як оператора мутації.

Можливе використання й комбінованих схем, наприклад вставлення фрагмента разом з його інверсією.

Завершуючи огляд ГА, зазначимо деяку суттєву особливість. Ці алгоритми повною мірою використовують простір розв'язків і мало залежать від інших компонент постановки задачі, яка розв'язується. Важливою умовою їх застосування є те, що розв'язок складається із фрагментів (блоків), які можна компонувати, переставляти чи модифікувати. Цільова функція грає другорядну роль, виникаючи лише у правилах відборів як компонент функції пристосованості. Позитивним моментом цього є можливість застосування ГА до розв'язання широкого кола різних проблем, зокрема визначених у нечислових просторах розв'язків, напри-

клад при підтримці прийняття рішень, розпізнаванні образів чи прогнозуванні. Зворотною стороною є та обставина, що врахування специфіки задач є одним із дієвих способів підвищення ефективності алгоритмів розв'язання.

До переваг ГА слід віднести нелокальний характер пошуку в просторі розв'язків на основі поєднання в нащадках властивостей одразу кількох розв'язків. Утім, з часом усе ж проявляється концентрація розв'язків у певній обмеженій області простору пошуку. Для подолання цього ефекту застосовуються різні способи диверсифікації, зокрема відбір у чергову популяцію з урахуванням метричних відстаней між розв'язками.

Оскільки вибір альтернативних варіантів реалізації ключових аспектів і значень параметрів ГА є складною проблемою (на яку не завжди звертають належну увагу при їх розробці та застосуванні), деякі дослідники пропонують приймати рішення, спираючись на вибір з кількох варіантів (типів кросовера, видів селекції, значень імовірності мутацій тощо), на основі аналізу перебігу обчислювального процесу, що дозволяє розробляти адаптивні версії ГА.

Однак більш кардинальною є розробка на основі ГА гібридних метаевристик, до яких належать, наприклад, міметичні алгоритми.

6. МІМЕТИЧНІ АЛГОРИТМИ

6.1. Принципи створення

Алгоритми, що мають тіснішу аналогію з культурною еволюцією, ніж з біологічною, називають *гібридними генетичними*, або *міметичними* (*memetic*) *алгоритмами* (МА). Нова термінологія пояснює те, що стратегія пошуку цих алгоритмів істотно відрізняється від стратегії пошуку інших еволюційних алгоритмів.

Можна виділити кілька цілей їх розробки, які важливі для ефективного еволюційного пошуку, що здійснюється зазначеними алгоритмами.

1) Алгоритми оптимізації мають бути ефективними, тобто здатними до конструювання прийнятних розв'язків за короткий час.

2) Алгоритми оптимізації орієнтуються на певну мету. Розглядаючи особин, що представляють розв'язки, як агентів, їх можна розцінити як свідомих суб'єктів, які в процесі пошуку співпрацюють і змагаються один з одним.

3) Унаслідок того, що оперуючий багатьма варіантами сліпий генетичний пошук вимагає значних ресурсів, тільки маленька частина популяції може бути розвинутою за короткий час. Отже, неадекватне здійснення вибірки з області пошуку веде до втрати різноманітності в генетичній інформації, що призводить до передчасної збіжності. Щоб подолати цей недолік, потрібен механізм диверсифікації (різноманітності) для введення інноваційних ідей (нових компонентів розв'язку). Певною мірою це може бути досягнуто шляхом застосування алгоритму пошуку в околі. Проте часто необхідні такі альтер-

нативні схеми рекомбінації для введення диверсифікації та інновацій, для яких ніякої біологічної аналогії не існує – наприклад оператори, які моделюють поведінку "неслухняних" агентів ("бунтівників проти миротворця").

Поняття міма ввів Р. Доукінз (Richard Dawkins) у книзі "Егоїстичний ген" (1976) як таку одиницю культурної еволюції, яка здатна до відтворення, поширення й модифікації. Сам термін утворений шляхом скорочення грецького слова mimema (*μίμημα*), що означає "наслідувати, бути подібним". Нинішня трактовка визначає мім як одиницю культурної інформації, здатної до реплікації. Особливістю культурної еволюції є те, що міми є одиницями передання культурної спадщини, але модифікуються й розповсюджуються не випадковим чином, а цілеспрямовано.

В основу МА якраз і покладено принцип цілеспрямованого поліпшення множини розв'язків, які утворюють популяцію.

6.2. Обчислювальна схема МА

МА може бути описаний кортежем (p, k_r, k_m), у якому p позначає кількість особин у популяції, k_r – ступінь застосування рекомбінації, а k_m – ступінь застосування мутації. Таким чином, кількість нащадків, які генеровані рекомбінацією, дорівнює $p \times k_r$, а шляхом мутації – $p \times k_m$. З використанням позначень із еволюційних алгоритмів досліджуваний алгоритм може бути описаний як (μ+λ)-стратегія при $\mu = p$ та $\lambda = p\ (k_r + k_m)$.

На відміну від ГА, рекомбінація й мутація в більшості МА застосовуються незалежно одна від одної. Для породження розв'язку як під час ініціалізації популяції, так і у варіаційних операторах можуть використовуватися жадібні конструктивні алгоритми.

Псевдокод МА показано на схемі 6.1. Усі особини в популяції представляють локальні оптимуми, що забезпечуєть-

ся застосуванням процедури локального пошуку *LS* як після генерації, так і після використання еволюційних варіаційних операторів.

```
procedure Memetic_Algorithm (x);
P: = ∅;
for j: = 1 to p do
x: = ФормуванняПрипустимогоВаріанта;
x: = LS (x);
P: = P ∪ x;
endfor;
repeat
for i: = 1 to k_r do
довільно вибрати особини x, y∈P;
z: = Рекомбінація (x, y);
z: = LS (z);
P: = P ∪ z;
endfor;
for i: = 1 to k_m do
довільно вибрати особину x∈P;
x: = Мутація (x);
x: = LS (x);
P: = P ∪ z;
endfor;
P: = ВідбірПопуляції (P);
if популяція P зійшлася then P: = Мутація& LS(P);
until
until виконається умова завершення;
x: = найкращий зі знайдених розв'язків;
return x;
end
```

Схема 6.1. Псевдокод міметичних алгоритмів

Крім того, включена схема диверсифікації, яка запозичена від CHC-методу (Cross-population selection, Heterogeneous recombination and Cataclysmic mutation – перехресний відбір, гетерогенне схрещування, катастрофічна мутація, відповідно), запропонованого Л. Ешелменом: якщо популяція збігається, то всі в ній піддаються мутації, окрім кращої особини, а згодом застосовується локальний пошук, щоб гарантувати, що всім особинам відповідають локальні оптимуми. Таким чином, диверсифікація – це оператор мутації вищого рівня, що діє відразу на всю популяцію – на відміну від еволюційних варіаційних операторів, які діють на окремі особини. Ця форма мутації може бути розцінена як метамутація. Необхідність у цьому механізмі пояснюється тим, що через відносно тривалі розрахунки в локальному пошуку можуть активно використовуватися тільки невеликі сукупності розв'язків. У свою чергу, це викликає швидку збіжність до субоптимальних областей у просторі пошуку, яку можна подолати зазначеною методикою перезапуску.

Вибір для відтворення популяції виконується довільно, тоді як вибір для розвитку – як у $(\mu+\lambda)$-еволюційній стратегії: оптимізоване в локальному масштабі потомство, що генероване рекомбінацією й мутацією, разом з поточною популяцією формує тимчасову популяцію, з якої далі вибираються кращі особини. Проте кожна особина вибирається тільки одного разу, тобто ідентичні копії вилучаються.

Деякі з опублікованих міметичних підходів використовують замість стандартного локального пошуку певні його модифікації, наприклад табуйований пошук, імітаційний відпал чи *G*-алгоритми.

МА моделюють процеси поширення, відбору мутацій і рекомбінації мімів – елементарних структурних одиниць знання. Реалізуючи певні аналогії з культурною еволюцією, МА, на відміну від ГА, використовують:

1. Роздільне, тобто паралельне, застосування операцій схрещування й мутації до всіх розв'язків, що утворюють поточну популяцію.

2. Поліпшення кожної створеної в результаті схрещування чи мутації особини на основі використання тих чи інших процедур локального пошуку.

3. Спеціальні процедури диверсифікації, які діють на всю популяцію одночасно.

Отже, МА формують диверсифіковані в просторі розв'язки популяції, у які особини включаються лише після опрацювання процедурами локального пошуку, тобто вони ведуть пошук у просторах локальних оптимумів.

7. ОПТИМІЗАЦІЯ МУРАШИНИМИ КОЛОНІЯМИ

7.1. Аналогії з природи

Ідеї деяких сучасних алгоритмів оптимізації навіяні природою. Серед них виділяються такі, які відносять до *ройового інтелекту* (swarm intelligence) – дисципліни, що має справу з природними та штучними системами, які складаються з багатьох індивідів (елементів, агентів, частинок), координація/узгодження дій між якими здійснюється шляхом децентралізованого управління та самоорганізації. Зокрема, ця дисципліна зосереджується на колективній поведінці, що є результатом локальних взаємодій окремих особин між собою та з навколишнім середовищем. Прикладами таких систем є колонії мурах і термітів, косяки риб, зграї птахів, стада наземних тварин.

Серед відомих алгоритмів ройового інтелекту в комбінаторній оптимізації найбільший інтерес становлять алгоритми оптимізації мурашиною колонією (ОМК; ant colony optimization – ACO), оптимізації роєм частинок (ОРЧ), бджолині алгоритми. Найбільшого поширення при розв'язанні ЗКО набули запропоновані Марком Доріго (1992) мурашині алгоритми, які далі розглянемо детальніше.

Мурашині алгоритми – це багатоагентні системи, де поведінка кожного агента, який називатимемо штучною мурахою, або надалі просто *мурахою*, заснована на поведінці справжніх мурашок. Вони застосовуються для розв'язання багатьох типів оптимізаційних задач, починаючи з класичної задачі комівояжера. Деякі дослідники під ОМК спочатку розуміли певний підклас мурашиних алгоритмів, однак останнім часом зазначений

термін у більшості публікацій вживається стосовно всіх алгоритмів цього типу – так будемо робити й ми.

Алгоритми ОМК були навіяні поведінкою колонії аргентинських мурашок (Iridomyrmex humilis). В експерименті їм було надано доступ до джерела їжі на майданчику, сполученому з мурашником мостом, що складається з двох гілок різної довжини (рис. 7.1). Гілки розташовані таким чином, що мурахи, які пересуваються в будь-якому напрямку (від мурашника до джерела їжі або навпаки), повинні вибирати одну з двох гілок. Експериментальні спостереження показали, що після перехідної фази, яка могла тривати кілька хвилин, більшість мурашок стали використовувати коротшу гілку. Було також помічено, що ймовірність вибору колонією коротшої гілки збільшується з різницею в довжині гілок.

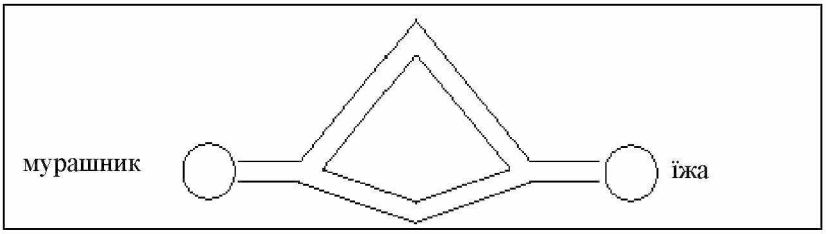


Рис. 7.1. Схема експерименту з мостами

Пояснити такий ефект можуть два принципи: автокаталізу (додатного зворотного зв'язку) і ефекту різниці в довжині шляхів (differential path length – DLP), використання яких стає можливим завдяки непрямій формі рознесеної в часі комунікації, опосередкованої локальною модифікацією оточення й відомої як *стигмергія* (stigmergy). Біологи виявили, що така відкладена взаємодія реалізується за допомогою спеціальної хімічної речовини – *феромону* (pheromone), яка виділяється спеціальними залозами мурах при їхньому переміщенні. Коли мурахи прибувають до точки розгалуження коротшої й довшої гілок, вони роблять імовірнісний вибір, заснований на відчутті ними кількості феромону на обох гілках. Така поведінка має автокаталітичний ефект, оскільки сам факт вибору шляху збільшуватиме ймо-

вірність того, що він буде знову вибраний іншими мурахами в майбутньому. На початку експерименту на обох гілках немає феромону, тому мурахи, що прямують від мурашника до джерела їжі, вибиратимуть будь-яку з двох гілок з рівною ймовірністю. Завдяки різній довжині гілок мурахи, що вибрали коротшу гілку, першими досягнуть джерела їжі. Коли вони, прямуючи назад до мурашника, досягнуть точки розгалуження, то знайдуть слід з більшою кількістю феромону на коротшому шляху, який відклали під час попереднього переходу до джерела їжі, і виберуть цей шлях з більшою ймовірністю, ніж довший. Зауважимо, що у природі кількість феромону, яка відкладається мурахами, може залежати від кількості та якості вжитої їжі.

З часом феромону на коротшому шляху стає все більше порівняно з довшим, завдяки чому мурахи все частіше й частіше вибирають короткий шлях доти, поки всі вони не починають його використовувати.

Моделювання подібної поведінки на основі колонії штучних мурашок дозволяє знаходити достатньо точні розв'язки багатьох оптимізаційних задач за прийнятний час обчислень.

7.2. Загальні принципи розробки мурашиних алгоритмів

Алгоритми ОМК застосовуються до задач оптимізації, які можуть бути охарактеризовані таким чином: їх розв'язок складається з компонентів, з яких можна покроково будувати фрагменти розв'язків, а на завершальному етапі – і весь розв'язок. Наприклад, у ЗК, що стала однією з перших задач, до якої був застосований алгоритм ОМК, такими фрагментами є дуги, що з'єднують міста і з яких формується маршрут – гамільтонів контур.

В алгоритмах ОМК формується *спеціальна модель задачі*, тому вони належать до класу моделе-орієнтованих методів. Така модель задачі подається у вигляді повного зваженого графа $G=(V,E)$, де $v_i \in V$, $i=1,...,n$, – вершини, що відповідають ком-

понентам розв'язку, а $e_{ij} \in E, e_{ij} = (v_i, v_j), v_i, v_j \in V,$ – ребра, які відповідають можливим з'єднанням (переходам) між відповідними вершинами. Для кожного ребра може бути визначена функція вартості з'єднання, можливо, залежна від деякого параметра часу t.

Умови ЗКО, що розв'язується, можуть визначати *набір обмежень* $\Pi = \Pi(V, E, t)$ для елементів V та E, які визначають припустимість зв'язків між компонентами та з'єднаннями, а в підсумку – і побудованого з них розв'язку.

Розв'язки задачі оптимізації можуть бути подані як припустимі шляхи на графі G. Алгоритми ОМК можуть використовуватися для знаходження припустимих шляхів мінімальної вартості, що задовольняють обмеження Π. *Вартість* $f(x)$, що відповідає розв'язку x, є функцією всіх вартостей з'єднань, які належать цьому розв'язку.

Мурахою в таких алгоритмах вважають параметричний рандомізований жадібний алгоритм, який покроково будує з множини компонент (вершин чи ребер графа задачі) припустимий розв'язок ЗКО. У своїй роботі він використовує евристичну інформацію та феромонний слід, які характеризують ребра (рідше – вершини) графа задачі.

Евристична інформація (зазвичай позначається η_{ij}) – це числове значення, що не залежить від знайдених на попередніх кроках розв'язків і відображає ступінь бажаності включення в побудований фрагмент розв'язку того чи іншого нового ребра графа моделі $e_{ij} \in E$. Евристичні значення η_{ij} базуються на апріорній інформації, що відображає умови конкретної задачі та надається джерелом, відмінним від мурах; вони можуть залежати від часу t.

Рівень феромону (*феромонний слід*) – τ_{ij}, що відповідає ребру $e_{ij} \in E$, – це додатне число, яке показує, наскільки часто мурахами використовувалося це ребро на попередніх кроках чи при формуванні повного розв'язку. Феромонні сліди є для мурах *довготривалою пам'яттю* щодо всього процесу пошуку. Залеж-

но від вибраного способу подання задачі, феромонні сліди можуть відповідати всім дугам задачі або тільки деяким з них.

У мурашиних алгоритмах популяція (колонія) агентів (або мурашок) спільно розв'язує сформульовану задачу оптимізації, використовуючи вищезазначене подання на графі моделі задачі.

Отже, основні компоненти обчислювальної схеми мурашиних алгоритмів такі:

- модель задачі, що подається спеціальним графом;
- феромонні значення (сліди);
- евристична інформація;
- пам'ять (локальна та глобальна).

Хоча кожна одна мураха достатньо складна, щоб бути в змозі знайти (імовірно, далеко не найкращий) розв'язок даної задачі, усе ж точніші розв'язки можуть з'явитися тільки в результаті колективної взаємодії між мурахами.

Мурахи мають такі загальні характеристики:

1. Кожна мураха використовує особисту інформацію та інформацію, локальну для вершини, у якій вона міститься.

2. Мурахи спілкуються з іншими мурахами тільки *непрямим способом* за допомогою інформації, яку вони зчитують/записують у змінні, що зберігають значення феромонних слідів.

3. Мурахи *не адаптуються*. Навпаки, вони адаптивно змінюють вигляд і сприйняття задачі іншими мурахами.

Отже, в алгоритмі ОМК кожна *штучна мураха* – це послідовний жадібний алгоритм, що при побудові шляху в графі задачі, який описується упорядкованою послідовністю вершин, використовує ймовірнісне правило для вибору чергового компонента розв'язку.

Нехай m – кількість мурах в одному поколінні. У мурашиних алгоритмах кожна мураха k, $(k = 1,...,m)$, має *пам'ять* M^k, яку вона використовує для зберігання інформації про пройдений шлях.

Пам'ять мурахи може використовуватися для:

- знаходження припустимих розв'язків;
- оцінки знайденого розв'язку;
- дослідження шляху назад.

Мурахи починають із початкового фрагмента (вершини графа) і рухаються в припустимі сусідні вершини, поступово формуючи розв'язок. Процедура пошуку завершується, якщо для мурахи k виконано принаймні одну з її завершальних умов.

Стани задачі визначаються в термінах скінченних послідовностей $y = (v_{s_1}, v_{s_2}, ...)$, $v_{s_r} \in V$, елементів V (або, що рівносильно, E), які на всіх проміжних кроках мурахи є фрагментами розв'язку задачі оптимізації. Якщо Y – множина всіх можливих послідовностей, то множина $\tilde{Y}$ усіх (під)послідовностей, які задовольняють обмеження $\Pi = \Pi(V, E, t)$, є підмножиною Y: $\tilde{Y} \subseteq Y$, а її елементи визначають припустимі стани задачі.

Нехай на певному кроці мураха k побудувала фрагмент розв'язку y, останньою компонентою якого є вершина $i \in V$, тобто вона перебуває в цій вершині: $y = (..., i)$. Тоді мураха може переміститися до будь-якої вершини j із множини припустимих сусідніх вершин N_i^k, визначуваних як $N_i^k = \{j : j \in N_i \wedge (y, j) \in \tilde{Y}\}$, де N_i – множина всіх сусідніх до i вершин графа задачі. Перехід здійснюється на основі ймовірнісного правила рішення.

Імовірнісне правило рішення для мурах – це функція:

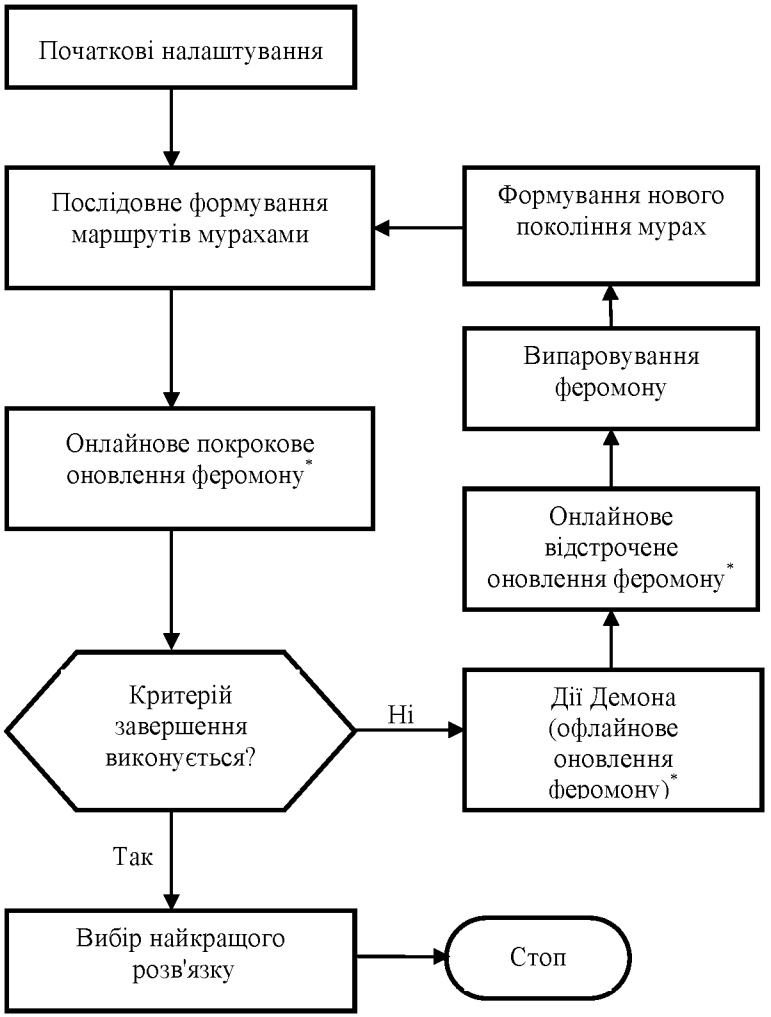
1) значень, що зберігаються в локальній для вершини структурі даних – *матриці мурашиних маршрутів* $A_i = (a_{ij})$, значення елемента якої отримують функціональною композицією локально доступних для вершини значень феромонних слідів і евристичних значень;

2) особистої пам'яті мурахи, що зберігає її передісторію;

3) обмежень задачі.

При переміщенні з вершини i в сусідню вершину j мураха може змінити феромонний слід τ_{ij} на дузі (i, j). Це називається *онлайновим покроковим оновленням* феромону. Завершивши формування розв'язку, мураха може пройти той самий шлях назад і поновити феромонні сліди на пройдених дугах. Це називається *онлайновим відстроченим оновленням* феромону.

Після того, як мураха, знайшовши розв'язок, пройшла шлях назад до початкової вершини, вона завершує діяльність, звільняючи всі зайняті ресурси (рис. 7.2).



Примітка: * – необов'язкові дії.

Рис. 7.2. Схема мурашиних алгоритмів

Крім діяльності мурашок, ці алгоритми можуть включати ще дві процедури.

1. *Випаровування феромону* – це процес, за допомогою якого інтенсивність феромонного сліду на ребрах графа задачі автоматично зменшується з часом. Випаровування феромону необхідне для того, щоб уникнути дуже швидкої збіжності алгоритму до субоптимальної області. Воно здійснює корисну форму *забування*, сприяючи дослідженню нових областей у просторі пошуку.

2. *Дії Демона*, що можуть використовуватися для здійснення централізованих дій, які не можуть бути виконані окремими мурахами. Прикладом такої діяльності є активація й виконання *процедури локальної оптимізації*, у якій як початкове наближення застосовується один чи кілька знайдених мурахами розв'язків, або ж *збирання глобальної інформації*, що може бути потрібна для прийняття рішення: буде або не буде корисним відкладання додаткового феромону для відхилення процесу пошуку від зони знайденого локального розв'язку. Важливо наголосити, що включення процедури дій Демона в алгоритм необов'язкове.

Наприклад, Демон може проглядати шляхи, знайдені кожною мурахою в колонії, і відкладати *додатковий* феромон на дугах, використаних однією чи кількома мурахами, що знайшли найкоротші шляхи.

Оновлення феромону, виконані Демоном, називаються *офлайновими оновленнями феромону* (див. рис. 7.2).

7.3. Обчислювальна схема та її ключові аспекти

Пошук розв'язку задачі комбінаторної оптимізації інтерпретується як знаходження оптимального маршруту в графі G групою штучних мурах (агентів), які не взаємодіють між собою безпосередньо, а лише через зміну середовища, використовуючи параметричну модель розподілу ймовірностей у просторі розв'язку задачі.

Загальну обчислювальну схему алгоритмів ОМК можна подати так (схема 7.1).

```
procedure ACO (x)
 ініціалізація_алоритму;
 while критерій_завершення_не_задоволений do
 формування_популяції_мурах;    {поточне покоління}
 foreach мураха_з_популяції do   {життєвий цикл мурахи}
 ініціалізація_мурахи;
 M = оновлення_пам'яті_мурахи;
 while поточний_стан ≠ повний_розв'язок do
     A = локальна_ матриця_мурашиних_маршрутів;
     сформувати_множину припустимих_вершин;
     p = обчислити_ймовірність_переходів(A, M, П);
     наступний_стан = правило_прийняття_рішення(p, П);
     перейти_в_наступний_стан(наступний_стан);
     if онлайнове_покрокове_оновлення_феромону then
         відкласти_феромон_на_відвіданій_дузі;
         поновити_
матрицю_мурашиних_маршрутів_A;
     endif
     M = оновити_внутрішній_стан;
   endwhile
   if онлайнове_відстрочене_оновлення_феромону then
     foreach відвіданої_дуги_побудованого_розв'язку do
         відкласти_феромон_на_відвіданій_дузі;
         оновити_матрицю_мурашиних_маршрутів_A;
     endforeach
 endif
 завершити діяльність;
 endforeach
 випаровування_феромону;
 оновлення_рекорду (x);
 дії_Демона;                    {необов'язково}
 endwhile
end
```

Схема 7.1. Псевдокод алгоритмів ОМК

На етапі ініціалізації здійснюються початкові налаштування: вибір значень параметрів алгоритму (зокрема кількості мурах m у колонії), задання значень феромонного сліду. Після утворення нового покоління мурах їх розміщують довільно у вершинах V графа G. Зазначені вершини є початковими фрагментами маршрутів.

Як зазначалося вище, на кожному кроці конкретна мураха досліджує найближчих сусідів тієї вершини графа задачі, якою закінчується побудований нею на даний момент фрагмент маршруту. Отже, для кожної мурахи k із популяції здійснюється:

a) формування підмножини припустимих вершин $N_i^k \subseteq N_i$ із множини вершин, сусідніх для тієї вершини $i \in V$, яка є останньою в поточному фрагменті маршруту;

b) обчислення ймовірності p_{ij}^k переходу від вершини i до довільної припустимої вершини $j \in N_i^k$ як функції від значень феромону τ_{ij} на ребрі $(i, j) \in E$ та евристичної інформації η_{ij};

c) імовірнісний вибір чергової припустимої вершини $s \in N_i^k$, включення її в кінець фрагмента маршруту, що будується;

d) модифікація значень τ_{ij}, якщо вибрано онлайнове оновлення.

Коли всі мурахи завершують роботу, може здійснюватися модифікація феромонних значень для певних дуг графа G (офлайнове оновлення), а також випаровування феромону (зниження значень τ_{ij}) для всіх його ребер, якщо такі дії не поєднані з оновленням феромону.

Оскільки алгоритм не має релаксаційного характеру, то необхідне запам'ятовування найкращого маршруту.

Найважливішими аспектами мурашиних алгоритмів є: спосіб оновлення феромону, імовірності переходів до наступної вершини та правила завершення роботи алгоритму.

Оновлення феромону. Оскільки за наявності лише фрагмента розв'язку відсутня оцінка значення цільової функції, то *онлайнове покрокове* оновлення феромону на дузі (i, j) графа задачі зазвичай здійснюється додаванням фіксованої величини:

$$\tau_{ij}(t+1) = \tau_{ij}(t) + \Delta, \qquad (7.1)$$

де Δ – параметр алгоритму, а значення t відповідає ітерації алгоритму, пов'язаній зі зміною поколінь мурах.

У деяких алгоритмах ОМК замість (7.1) використовується складніша формула оновлення феромону з одночасним випаровуванням:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + (1-\rho)\tau_0, \qquad (7.2)$$

де τ_0 – початкова кількість феромону, ρ – ще один параметр алгоритму, що називається коефіцієнтом випаровування, $0 \le \rho \le 1$.

Інколи в (7.2) покладають $\tau_0 = \dfrac{1}{f^0{}_{\min}}$, де $f^0{}_{\min}$ – найкраще значення цільової функції на початковій популяції мурах.

Онлайнове відстрочене оновлення феромону на ребрі зазвичай визначається з урахуванням якості знайденого розв'язку:

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + \Delta\tau_{ij}, \qquad (7.3)$$

де $\Delta\tau_{ij}$ – сума внесків усіх мурашок, $\Delta\tau_{ij} = \dfrac{1}{m}\sum_{k=1}^{m}\Delta\tau_{ij}^k$, а внесок конкретної мурахи $k, k = 1,...,m$,

$$\Delta\tau_{ij}^k = \begin{cases} Q \Big/ f\left(x^k\right), \text{ якщо ребро } (i,j) \text{ належить розв'язку } x^k, \\ 0 \text{ в іншому разі,} \end{cases}$$

де Q – параметр алгоритму, а $f\left(x^k\right)$ – значення цільової функції задачі у знайденому мурахою розв'язку x^k.

Іноді використовується поширений принцип *елітизму*: таке оновлення здійснювати лише для найкращого зі знайдених розв'язків (на поточній ітерації чи за весь період пошуку).

При цьому часто застосовують підсилення принципу елітизму: здійснюється ще й відкладання Демоном додаткового феромону на тих ребрах графа, які входять в один чи кілька най-

кращих на даний момент розв'язків, тобто з урахуванням усієї історії пошуку.

Модифікація феромонного сліду в (7.3) з метою уникнення повторної побудови деяких шляхів може здійснюватися таким чином: якщо відомі кілька розв'язків, то на їх основі формується середнє значення цільової функції $\bar{f}$, а внесок поточного розв'язку визначається формулою

$$\Delta\tau_{ij} = \tau_0(1 - \frac{f - f_{\min}}{\bar{f} - f_{\min}}),$$

де $f_{\min}$ – найкраще з відомих значень цільової функції (рекорд), f – поточне значення.

Випаровування феромону має на меті створити умови для диверсифікації пошуку й уникнення зациклювання колонії на локальних розв'язках:

$$\tau_{ij}(t+1) = (1-\rho)\tau_{ij}(t).$$

Вище зазначалося, що випаровування може бути поєднане з підвищенням значень феромону як у формулі (7.2). Зрозуміло, що в конкретному алгоритмі слід вибирати лише один зі способів випаровування.

Правило переходу до нової вершини. Імовірнісне правило переходу мурахи k від поточної вершини $i \in V$ до нової вершини $j \in N_i^k$ у типових мурашиних алгоритмах базується на визначенні ймовірності p_{ij}^k:

$$p_{ij}^k = \frac{a_{ij}(t)}{\sum_{r \in N_i^k} a_{ir}(t)}, \qquad (7.4)$$

де $a_{ij}(t)$ – елемент матриці мурашиних маршрутів, який залежить від значень феромонного сліду та евристичної інформації в момент часу t.

Найпоширенішими варіантами обчислення ймовірностей при переході є

$$p_{ij}^{\,k} = \frac{\tau_{ij}^{\,\alpha}(t)\eta_{ij}^{\,\beta}(t)}{\sum\limits_{r \in N_i^k} \tau_{ir}^{\,\alpha}(t)\eta_{ir}^{\,\beta}(t)} \tag{7.5}$$

чи

$$p_{ij}^{\,k} = \frac{\tau_{ij}^{\,\alpha}(t) + \eta_{ij}^{\,\beta}(t)}{\sum\limits_{r \in N_i^k} [\tau_{ir}^{\,\alpha}(t) + \eta_{ir}^{\,\beta}(t)]}, \tag{7.6}$$

де α, β – параметри алгоритму, які відображають ступінь значущості феромонного сліду та евристичної інформації ($\alpha, \beta > 0$).

Якщо кожне ребро $(i, j) \in E$ характеризується числовим значенням q_{ij}, що відображає ступінь його непривабливості (наприклад, довжина в задачі комівояжера), то використовують нормовані евристичні значення:

$$\eta_{ij} = 1 - \frac{q_{ij}}{\sum\limits_{s \in N_i} q_{is}}. \tag{7.7}$$

Тоді значення елементів матриці мурашиних маршрутів можна розраховувати як комбінацію феромонних та евристичних значень:

$$a_{ij}(t) = \frac{w\,\tau_{ij}(t) + (1-w)\,\eta_{ij}}{w + \dfrac{1-w}{\|N_i\| - 1}}, \tag{7.8}$$

де $w \in [0,1]$.

У деяких алгоритмах застосовується *псевдовипадкове пропорційне правило*, засноване на використанні додаткового параметра $p_0 \in [0,1]$: з імовірністю p_0 мураха переходить з вершини $i \in V$ до нової вершини $j \in N_i^k$, для якої максимізується добуток кількості феромону на величину евристичної інформації, піднесених до відповідних степенів, тобто

$$j = \arg\max\{\tau_{ir}^{\alpha}(t)\eta_{ir}^{\beta}(t) : r \in N_i^k\},$$

у той час як з імовірністю $1-p_0$ обирається вершина за загальним правилом, тобто за формулою (7.4) з використанням (7.5)–(7.8).

Зрозуміло, що у випадку вибору $p_0=0$ отримуємо принцип вибору, який застосовується у звичайних алгоритмах ОМК; якщо ж $p_0=1$, то працює лише псевдовипадкове пропорційне правило.

Найчастіше використовуються такі **критерії завершення** обчислень у мурашиних алгоритмах:

1. Відсутність (чи невелика зміна) поліпшення наявного рекорду чи середньої пристосованості популяції мурашок на кількох останніх ітераціях.

2. Перевищення заданої кількості $H_{\max}$ ітерацій (змін поколінь/популяцій), де $H_{\max}$ – параметр алгоритму.

3. Вичерпання заданого часу на обчислення.

Важливо зазначити, що алгоритми ОМК володіють природним паралелізмом.

7.4. Про модифікації мурашиних алгоритмів

За останній час розроблено багато модифікацій мурашиних алгоритмів, які використовують ті чи інші способи реалізації їх ключових аспектів і принципів побудови. Деякі прикладні алгоритми ОМК використовують комбінації основних принципів, тому зазначимо лише декотрі відомі конкретні схеми, а саме:

1) алгоритм мурашиних систем (Ant System, AS);

2) алгоритм системи мурашиних колоній (Ant Colony System, ACS);

3) алгоритм *макс-мін мурашині системи* (MAX-MIN Ant System, MMAS).

Першими реалізаціями мурашиних алгоритмів були *алгоритми мурашиних систем* AS. Розглянемо їх основні особливості, які стосуються оновлення феромону та правила прийняття рішення.

Для оновлення феромонних значень використовується онлайнове відстрочене оновлення феромону, яке стосується дуг графа, що ввійшли до всіх розв'язків, побудованих мурахами. При цьому використовується формула (7.3), тобто величина зміни феромону пропорційна якості знайденого розв'язку.

Розширенням алгоритму мурашиних систем стало використання елітної стратегії – відкладанні додаткового феромону Демоном на ребрах графа, які входять до найкращого на даний момент розв'язку.

Правило переходу до нової вершини в більшості алгоритмів мурашиних систем базується на обчисленні ймовірностей за формулами (7.4)–(7.6).

Також був запропонований алгоритм мурашиних систем, заснований на ранжуванні (AS_{rank}). Він, по суті, є розширенням застосування елітної стратегії й полягає в такому. Після завершення діяльності чергового покоління мурахи сортуються за вартістю побудованих ними розв'язків і для найкращого розв'язку, знайденого за весь час роботи алгоритму, феромонні значення збільшуються з вагою w, а саме збільшення феромонів виконується тільки для ребер, що ввійшли у розв'язок (w–1) кращих мурах. При цьому для k-і кращої мурахи буде додаватися феромон з вагою ($w - k$) відповідно до (7.9):

$$\tau_{ru}(t+1) = \rho \cdot \tau_{ru}(t) + w \cdot \Delta\tau_{ru}^{gb}(t) + \sum_{k=1}^{w-1} [(w-1)\Delta\tau_{ru}^{k}(t)], \qquad (7.9)$$

де $\Delta\tau_{ru}^{gb}(t) = 1/f^{gb}(t)$, $f^{gb}(t)$ – значення цільової функція для найкращого з відомих розв'язків, а t – номер ітерації (покоління).

Алгоритм системи мурашиних колоній ACS поліпшує алгоритм мурашиних систем AS на основі використання інформації, отриманої попередніми поколіннями, для модифікації ймовірнісної моделі пошуку. Це досягається за допомогою двох механізмів. По-перше, використовується строго елітна стратегія при поновленні феромонів на ребрах, тобто феромон змінюється лише на ребрах, які входять у найкращий знайдений мурахами розв'язок. По-друге, мурахи вибирають наступну вершину для

включення у фрагмент розв'язку, використовуючи псевдовипадкове пропорційне правило (див. п. 7.3).

При оновленні феромону після завершення діяльності чергового покоління його відкладає лише та мураха, яка знайшла найкращий розв'язок, використовуючи формулу

$$\tau_{ij}(t+1) = \rho\tau_{ij}(t) + (1-\rho)\Delta\tau_{ij}^{\text{best}}(t),$$

де величина $\Delta\tau_{ij}^{\text{best}}(t)$ розраховується як і в (7.3), але лише на основі одного цього найкращого розв'язку, а t, як і раніше – номер ітерації/покоління.

За кращу мураху може вибиратися та, яка знайшла кращий розв'язок на даній ітерації, тобто з поточного покоління (iteration-best), або ж мураха, яка знайшла найкращий розв'язок, ураховуючи всі ітерації від початку роботи алгоритму (best-so-far).

Також у алгоритмі системи мурашиних колоній застосовується онлайнове покрокове оновлення феромону згідно з формулою (7.2), що сприяє зменшенню ймовірності вибору однакових шляхів усіма членами популяції.

При розв'язанні задач комівояжера в алгоритм системи мурашиних колоній було включено процедуру Демона, яка базувалася на алгоритмі локального пошуку з використанням 2- і 3-замін Ліна (див. розд. 3).

В *алгоритмі макс-мін мурашині системи* MMAS задаються нижня й верхня межі для значень феромонів на довільному ребрі. При цьому використовується і спеціальний підхід до визначення феромонних значень при ініціалізації розрахунків. Отже, в алгоритмі MMAS задається інтервал можливих значень феромону у вигляді меж $\tau_{\min}$ та $\tau_{\max}$, тобто досягають виконання нерівностей $\tau_{\min} \le \tau_{ij} \le \tau_{\max}$ для всіх ребер $(i, j) \in E$. Кількість феромону на ребрах при ініціалізації задається рівною нижній межі інтервалу, що забезпечує краще дослідження простору розв'язків. У MMAS, як і в системі мурашиних колоній, тільки найкраща мураха (глобально або локально краща в даній попу-

ляції) виконує додавання феромону після кожної ітерації алгоритму. Результати обчислень показали, що кращі результати досягаються тоді, коли оновлення феромону виконується з використанням глобально кращого розв'язку. Крім того, в алгоритмі MMAS часто застосовується як Демон *локальний пошук* для поліпшення якості розв'язків, що будуються.

Основні параметри та часто використовувані значення:

α, β – відносні важливості феромонного сліду і привабливості ребра ($\alpha = 1–1.5$; $\beta = 0.5–2$);

ρ – стала випаровування (зазвичай покладають $\rho = 0.9$);

m – кількість мурашок;

Q – ваговий коефіцієнт;

τ_0 – початкова кількість феромону;

p_0 – імовірність у псевдовипадковому правилі (часто покладають $p_0 = 0.9$).

Роль параметрів α та β така:

- якщо $\alpha = 0$, то найбажаніші вершини будуть вибрані з більшою ймовірністю – це відповідає класичному стохастичному жадібному алгоритму (з багатьма стартовими точками, по яких мурахи спочатку хаотично розподілені);
- якщо $\beta = 0$, то працює тільки збільшення феромону – такий спосіб зумовлює швидку появу стагнації, тобто ситуацію, при якій усі мурахи роблять однакові обходи, які, узагалі кажучи, є строго субоптимальними, тому має бути встановлено відповідний компроміс між значенням евристики та інтенсивністю сліду.

У деяких модифікаціях ОМК пропонується розглядати α, β не як сталі параметри, а змінювати їх значення у процесі обчислень із урахуванням отримуваних проміжних результатів.

Прагнення до підвищення ефективності алгоритмів ОМК спричинило, зокрема, появу комбінованих обчислювальних схем, серед яких алгоритми, що використовують кілька колоній (т. зв. острівні моделі), і деякі гібридні метаевристики (наприклад на основі кооперування ОМК із *H*-алгоритмами).

7.5. Практичні питання застосування мурашиних алгоритмів

Розглянемо деякі з найхарактерніших аспектів алгоритмів ОМК. Зокрема, зосередимо увагу на способах оцінювання отриманих мурахами розв'язків і застосування цих оцінок для вибору напрямів за допомогою відкладання феромонних слідів мурашок, а також на важливості використання колонії мурашок.

Неявна та явна оцінки розв'язків. В алгоритмах ОМК знайдені мурахами розв'язки створюють зворотний зв'язок для спрямування пошуку майбутніх мурашок, що входять у систему. Це зроблено за допомогою двох механізмів.

Перший механізм, спільний для всіх алгоритмів ОМК, складається з *явної* оцінки розв'язку. У цьому разі використовується деяка міра якості знайденого розв'язку для прийняття рішення, яку кількість феромону мурахам слід відкласти.

Другий механізм – тип *неявної* оцінки розв'язків. У цьому випадку мурахи використовують ефект різниці довжин шляхів (DPL) подібно до поведінки реальних мурашок, тобто якщо мураха вибирає коротший шлях, то вона першою відкладає феромон і впливає на пошук наступних мурашок.

Зазначимо, що в географічно розподілених проблемах, подібних до задач проектування й експлуатації мереж різної природи, неявна оцінка розв'язку, заснована на ефекті DPL, може грати важливу роль. Часто можна знайти хороші розв'язки мережних проблем, використовуючи тільки ефект DPL. Також показано, що поєднання явної та неявної оцінок розв'язку (роблячи кількість феромону пропорційною вартості знайденого розв'язку) збільшує ймовірність отримання точніших розв'язків.

Явна оцінка розв'язку й відкладання феромону. Після того, як мураха побудувала розв'язок, його використовують для обчислення кількості феромону, який мураха повинна відкласти на відвіданих дугах. У мурашиній системі AS, наприклад, кожна мураха відкладає кількість феромону, обернено пропорційну

вартості знайденого розв'язку. Очевидно, що це тільки один з можливих варіантів та існує безліч застосувань алгоритмів ОМК для задачі комівояжера або інших задач комбінаторної оптимізації, які використовують різні функціональні характеристики розв'язку для визначення того, як багато феромону мурахи або Демон повинні відкласти. Проблема, яка виникає в задачах маршрутизації й узагалі в будь-якій задачі, характеристики якої протягом розв'язання змінюються непередбачувано, полягає в тому, що не існує простого способу оцінювання розв'язку й визначення, скільки феромону повинні відкласти мурахи. Вихід із цієї проблеми, який використовується мурашиною мережею (AntNet), полягає в тому, щоб за допомогою мурашок вивчати в режимі онлайн стан мережі, що може застосовуватися для оцінки того, наскільки якісними є знайдені мурахами розв'язки.

Кількість мурашок. Кількість мурашок, що використовуються, – це параметр, який, у більшості випадків, має бути визначений експериментально. На щастя, алгоритми ОМК виявляються досить стійкими до фактичної кількості мурашок, що використовуються. Тому обмежимо розгляд таким запитанням: навіщо використовувати колонію мурашок (тобто встановлювати $m > 1$) замість єдиної мурахи? Фактично, хоча одна окрема мураха здатна до знаходження розв'язку, з міркувань ефективності припускають, що бажаним є використання колонії мурашок. Це особливо стосується географічно розподілених проблем, оскільки ефект різниці в довжині, що використовується мурахами для розв'язання цього класу проблем, може виникати тільки за наявності колонії мурашок. Також варто звернути увагу на те, що в задачах маршрутизації мурахи розв'язують $m < n^2$ задач найкоротшого шляху і колонія мурашок потрібна для кожної з цих задач.

З іншого боку, для задач комбінаторної оптимізації, у яких мурахи рухаються синхронно, використання m мурашок, які будують M розв'язків кожна (тобто алгоритм ОМК запускається для M ітерацій), може бути еквівалентним, принаймні теоретично, використанню однієї мурахи, яка генерує $m \cdot M$ розв'язків.

Проте результати експериментів показують, що продуктивність вище, коли кількості мурашок m надаються значення більше одиниці, які в загальному випадку залежать від класу задач, до розв'язання яких застосовується алгоритм ОМК.

Хоча алгоритми ОМК застосовні для знаходження шляхів мінімальної вартості (найкоротших шляхів) на графі взагалі, важливо звернути увагу на те, що вони є продуктивним підходом особливо в тих задачах знаходження найкоротшого шляху, до розв'язання яких більшість класичних алгоритмів, подібних динамічному програмуванню або методу зміни позначок, не можуть бути ефективно застосовані.

Це стосується, наприклад, таких типів задач знаходження найкоротшого шляху:

- *NP*-складних задач, для яких розмірність простору розв'язків експоненційно залежить від розмірності задачі. У цьому випадку виявляється та особливість, що штучні мурахи використовують набагато менший граф моделі задачі, побудований з елементів задачі, і свою пам'ять для генерації припустимих розв'язків, які в більшості застосувань ОМК потім зводяться до локального оптимуму специфічним для задачі алгоритмом (наприклад тим чи іншим ЛП).
- Задач знаходження найкоротшого шляху, де властивості графа досліджуваної задачі змінюються з часом одночасно з процесом оптимізації, який повинен пристосуватися до динаміки проблеми. У цьому випадку граф задачі хоча й може бути навіть фізично доступним (як у мережних задачах), але його властивості, подібно значенню вартості з'єднання, можуть змінюватися з часом. У цьому разі можна припустити, що використання алгоритмів ОМК стає ефективнішим, коли наявна інформація щодо характеру чи закономірностей зміни вартостей дуг графа задачі.
- Задач, для розв'язання яких обчислювальна архітектура розподілена просторово, як у разі паралельної та/чи мережної обробки. Тут алгоритми ОМК можуть бути дуже ефективними завдяки притаманній їм розподіленій і багатоагентній природі, що добре відповідає цим типам архітектури комп'ютерних засобів.

Результати експериментів показують, що часто ефективність мурашиних алгоритмів збільшується зі зростанням розмірності розв'язуваних задач оптимізації. Прийнятні на практиці результати отримують, зокрема, для нестаціонарних систем зі змінними в часі параметрами, наприклад для розрахунків телекомунікаційних і комп'ютерних мереж. Утім за відповідної інтерпретації параметрів оптимізаційних проблем на основі застосування мурашиних алгоритмів отримані конкурентні результати для таких складних і різноманітних оптимізаційних задач, як задача комівояжера, квадратична задача про призначення, задача маршрутизації транспортних засобів з урахуванням часових вікон, транспортна задача, задача календарного планування, задача оптимізації мережних трафіків, передбачення третинної структури протеїнів і багатьох інших.

Якість отримуваних розв'язків багато в чому залежить від значень змінюваних параметрів алгоритмів у ймовірнісному правилі вибору шляху на основі поточної кількості феромону та параметрів правил відкладання й випаровування феромону. Можна очікувати, що динамічне адаптивне налаштування цих параметрів сприятиме отриманню кращих розв'язків. Важливу роль відіграє й початковий розподіл феромону, задання обсягу мурашиної колонії, а також спосіб вибору початкових варіантів розв'язку на кроці ініціалізації.

Перспективними шляхами поліпшення мурашиних алгоритмів є різні адаптації параметрів з використанням бази нечітких правил чи їх гібридизація, наприклад з генетичними чи іншими популяційними алгоритмами. Як варіант, така гібридизація може полягати в обміні через певні проміжки часу поточними найкращими розв'язками. Глибше поєднання гібридних схем може досягатися за рахунок обміну внутрішніми даними алгоритмів, наприклад елементами матриці феромонних слідів чи всією множиною поточних розв'язків (популяцією), що приводить до побудови кооперативних метаевристик.

До *недоліків* алгоритмів ОМК можна віднести таке.

1. Теоретичний аналіз ускладнений унаслідок того, що в процесі роботи ОМК отримуються послідовності випадкових, але не незалежних розв'язків, причому функція розподілу ймовірностей змінюється на кожній ітерації.

2. Хоча збіжність гарантується за досить загальних умов, проте час збіжності (і відповідно трудомісткість) важко піддається теоретичному оцінюванню.

3. Для отримання поліпшених розв'язків зазвичай необхідним є застосування додаткових методів, таких як локальний пошук чи навіть інші популяційні алгоритми, що досягається в деяких розроблених з їх використанням гіперевристичних методах.

4. Алгоритми суттєво залежать від налаштування параметрів, які підбираються в загальному випадку лише експериментально. Утім це зауваження стосується практично всіх ітераційних алгоритмів комбінаторної оптимізації.

8. МЕТОД ДЕФОРМАЦІЙ У КОМБІНАТОРНІЙ ОПТИМІЗАЦІЇ

8.1. Метод Нелдера – Міда

При розв'язанні задач недиференційовної нелінійної оптимізації успішно застосовується пошук методом деформованого багатогранника Нелдера – Міда (Nelder-Mead Downhill Simplex Method). Ґрунтуючись на основних засадах цього методу, було запропоновано алгоритм комбінаторної оптимізації, призначений для використання як на традиційних комп'ютерах, так і багатопроцесорних обчислювальних комплексах (БОК), який називається дискретним *методом деформованого багатогранника* (МДБ). Згодом на основі МДБ було розроблено гібридну метаевристику, названу *H*-алгоритмом – про ці алгоритми йтиметься далі.

Метод Нелдера – Міда призначений для мінімізації нелінійних недиференційовних функцій $f(x)$, які визначені на n-вимірному евклідовому просторі: $x \in E^n$. Багатьма дослідниками він вважається одним з найкращих методів оптимізації, якщо розмірність задачі оптимізації n має порядок 10 – 12.

На початковому етапі ($k = 0$) у просторі E^n будується регулярний багатогранник (симплекс) – він має $n+1$ вершину. Перебіг ітерацій проілюструємо для випадку, коли $n = 2$ (рис. 8.1–8.3).

1. Початок ітерації k ($k = 0,1, \ldots$).

Сортування. Серед вершин багатогранника за значеннями цільової функції $f(x)$ вибираються найгірша точка $x_h^{(k)}$, друга найгірша точка $x_g^{(k)}$ і найкраща точка $x_l^{(k)}$. Позначимо відповідні значення цільової функції f_h, f_g, f_l, звідки маємо $f_l < f_g < f_h$.

2. Побудова центра ваги. Визначається центр ваги $\overline{x}$ багатогранника, утвореного всіма вершинами $x_i^{(k)}$, за винятком найгіршої $x_h^{(k)}$:

$$\overline{x} = \frac{1}{n}\sum_{i \neq h} x_i^{(k)}.$$

3. Віддзеркалення (reflection). Вершина $x_h^{(k)}$ проектується через центр ваги $\overline{x}^{(k)}$ з коефіцієнтом α>0. Отриману точку позначимо x_r (рис. 8.1).

$$x_r = (1+\alpha)\overline{x} - \alpha x_h.$$

4. Порівняння f_r із f_l, f_g, f_h. Можливі такі випадки:

4.1) $f_r < f_l$, тобто знайдене значення краще за всі відомі. Тоді здійснюється розтягування (expansion) відрізка $[\overline{x}, x_r]$ з коефіцієнтом γ ($\gamma > 1$), чим утворюється точка x_e:

$$x_e = (1-\gamma)\overline{x} + x_r.$$

Якщо $f_e < f_r$, то замінюємо $x_h^{(k)}$ на x_e, інакше – на x_r.

Переходимо на п. 8 (перевірка умов завершення роботи алгоритму).

4.2) $f_l < f_r < f_g$. Знайдено досить непогану точку, якою можна замінити найгіршу: замінюємо $x_h^{(k)}$ на x_r.

Переходимо на п. 8 (перевірка умов завершення).

4.3) $f_g < f_r < f_h$. Здійснюється перепозначення: точка x_r позначається як $x_h^{(k)}$ і навпаки (відповідно перепозначаються також значення f_r, f_h).

Переходимо на п. 5.

4.4) $f_h < f_r$. Переходимо на п. 5.

5. Стискування (contraction). У даний момент роботи алгоритму (можливо, після перепозначення) маємо $f_l < f_g < f_h < f$.

Отримуємо точку x_s шляхом зміщення точки $x_h^{(k)}$ до $\overline{x}$ із коефіцієнтом $\beta > 0$:

$$x_s = \beta x_h + (1-\beta)\overline{x}\,.$$

6. Якщо $f_s < f_h$, то замінюємо точку $x_h^{(k)}$ на x_s.

Переходимо на п. 8 (перевірка умов завершення роботи алгоритму).

7. Редукція (shrinkage). Якщо $f_s > f_h$, то поліпшення не знайдено. Усі вектори $\left[x_l^{(k)}, x_i^{(k)}\right]$, $i = 1,\ldots, n+1$, $i \neq l$, зменшуємо у два рази з відліком у точці $x_l^{(k)}$, тобто "стягуємо" вершини до найкращої.

8. Перевірка критерію завершення.

Авторами методу було запропоновано таку умову завершення (критерій Нелдера – Міда):

$$\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}\left[f(x_i^{(k)}) - \overline{f}\right]^2} \le \varepsilon\,,$$

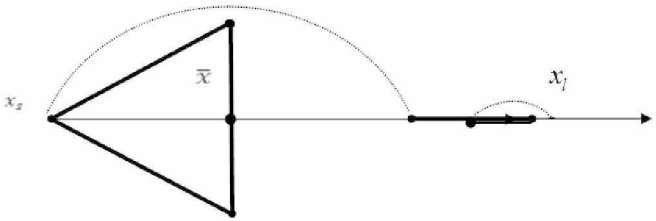
де $\overline{f} = \dfrac{1}{n+1}\sum_{i=1}^{n+1} f\left(x_i^{(k)}\right)$ – середнє значення функції, ε – довільне мале число (дисперсія за значеннями цільової функції).

Рекомендується цей критерій доповнювати оцінкою дисперсії за розмірами багатогранника:

$$\sqrt{\frac{1}{n+1}\sum_{i=1}^{n+1}\left(x_i^{(k)} - \overline{x}^{(k)}\right)^2} \le \varepsilon,$$

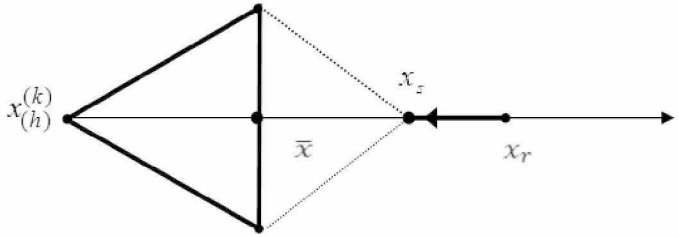
де $\overline{x}^{(k)}$ – центр ваги всього багатогранника.

Якщо умови не виконуються, то покладаємо $k := k+1$ і переходимо на початок ітерації.



$$x_l = (1-\gamma)\overline{x} + x_r$$

Рис. 8.1. Відображення й розтягування



$$x_s = \beta x_n + (1-\beta)\overline{x}$$

Рис. 8.2. Стискування

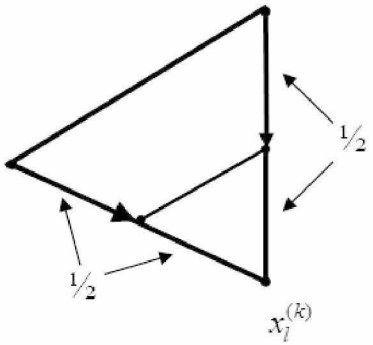


Рис. 8.3. Редукція

8.2. Дискретний метод деформованого багатогранника

Нехай тепер (X, d) – деякий дискретний метричний простір з метрикою $d(x, y)$. Покладемо $h = \inf\limits_{x,y\in X}\{d(x,y): x \neq y\}$. В алгоритмах деформованого багатогранника істотну роль грає поняття променю, яке в нашому випадку введемо на основі поняття *d-відрізка*.

Означення 8.1. *d*-відрізком $/x, y/$, що з'єднує довільні дві точки $x, y\in X$, назвемо впорядковану сукупність точок $x_i\in X$, $i = 1,...,k$, які задовольняють умови:

1) $d(x, x_i)+ d(x_i, y) = d(x, y)$, $i = 1,...,k$;

2) $x_1 = x$, $x_k = y$;

3) $d(x, x_i) < d(x, x_{i+1})$, $i = 1,...,k-1$;

4) не існує такої точки $z\in X$, що

$d(x_i, z) + d(z, x_{i+1}) = d(x_i, x_{i+1})$, $z\neq x_i$, $z\neq x_{i+1}$, $i = 1,...,k-1$.

Означення 8.2. Під *d*-інтервалом $<x, y>$ будемо розуміти сукупність

$$<x,y>=/x, y/ \setminus \{x, y\}.$$

Означення 8.3. Назвемо *d*-півінтервалом, що з'єднує дві довільні точки $x, y\in X$, *d*-відрізок без однієї крайньої точки:

$$<x, y/ \equiv /x, y/ \setminus \{x\}.$$

Далі будемо розглядати в певному сенсі рівномірні простори: якщо $d(x, y) > h$, то вважатимемо, що інтервал $<x,y>\neq\varnothing$.

Загальна схема згаданого дискретного *методу деформованого багатогранника* МДБ може бути подана таким чином.

1. Формуємо початкову популяцію.

2. Довільно вибираємо дві точки з популяції.

3. Проводимо півінтервал від точки з гіршим значенням цільової функції через другу точку (у випадку скінченного простору півінтервал теж буде скінченним).

4. Розв'язуємо підзадачу: знаходимо на цьому півінтервалі (виключаючи окіл другої вибраної з популяції точки) точку з найменшим значенням цільової функції.

5. Якщо знайденій точці відповідає значення цільової функції, краще за значення у вихідних точках популяції, то знайдена

точка замінює відповідну (найгіршу) точку в популяції. В іншому разі обидві вибрані точки зазначаються як неперспективні й далі *у парі* не вибираються.

У деяких розвинених алгоритмах комбінаторної оптимізації для уникнення концентрації пошуку в обмеженій підобласті простору варіантів розв'язку задачі X і підвищення точності отримуваних розв'язків використовують процедури збурення (як у повторюваному ЛП та G-алгоритмах) чи схрещування й мутації (як у ГА та МА). Зауважимо, що подібні процедури породжують підмножини варіантів розв'язку, які не узгоджені з топологією простору X. Прикладом подібного узгодження є алгоритм МДБ. Здійснювана в ньому побудова півінтервалів дає можливість поєднувати пошук в околах із глобальним скануванням простору X, причому процедура сканування, на відміну від загальних операторів збурення в більшості інших методів, визначена конкретно.

Зазначимо, що дискретний МДБ можна вважати певним узагальненням запропонованого Ф. Гловером (1989) алгоритму переключення шляхів (path relinking).

8.3. *H*-АЛГОРИТМИ. ОСНОВНІ ПОЛОЖЕННЯ

На основі поєднання ідей МДБ та алгоритмів ЛП із використанням підходу на основі популяцій було запропоновано схему гібридних метаевристик, які називаються H-алгоритмами. Будемо використовувати термінологію, яка застосовується також при описі еволюційних алгоритмів.

Нехай маємо чергову популяцію $P \subset X$. Тоді основний механізм пошуку на ітерації алгоритму полягає в реалізації таких кроків.

1) Відбір пари точок $x, y \in P$: $f(x) > f(y)$.

2) Побудова півінтервалу $< x, x^{\infty}/$ такого, що $y \in < x, x^{\infty}/$. Пошук z – кращої точки (субоптимального розв'язку) уздовж півінтервалу, виключаючи окіл точки y.

3) Запуск ЛП LS із точкою z, яку беруть за початкове наближення.

4) Доповнення популяції P знайденою точкою локального мінімуму.

Загальна схема алгоритму, що ілюструє зазначений механізм пошуку, зображена на рис. 8.4.

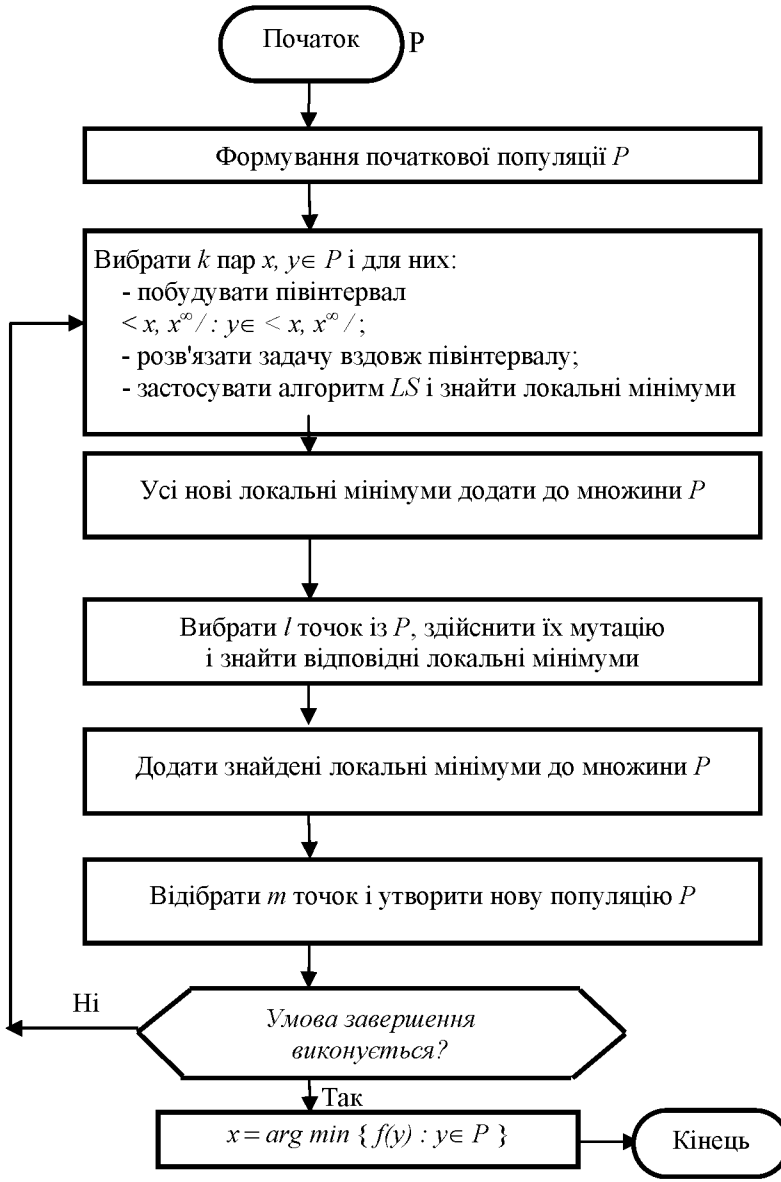


Рис. 8.4. Блок-схема *H*-алгоритмів

Формальніше псевдокод *H*-алгоритмів подано на схемі 8.1.

```
procedure H(x);
  h: = 0; P^0 = ∅;
  for i: = 1 to m do
    x: = ГенераціяПрипустимогоВаріанта;
    x: = LS (x);
    P: = P ∪ x;
  endfor;      {завершено формування початкової популяції P}
  repeat
    P: = ∅;
    for i: = 1 to k do
      ВідбірДляВаріації (x,y∈P^h);
      будуємо півінтервал <x,x^∞/ такий, що y∈<x,x^∞/;
      z: = arg min {f(u) : u ∈<x,x^∞/ \ L_s(y), y∈<x,x^∞/ };
      z: = LS (z);
      P: = P ∪ z;
    endfor;
            {сформовано тимчасову популяцію з (m+k) точок}
    for i: = 1 to l do
      ВідбірДляМутації (x∈P);
      x: = Мутація (x);
      x: = LS (x);
      P: = P ∪ x;
    endfor;
            {сформовано тимчасову популяцію з m+k+l точок}
    P^{h+1}: = ВідбірПопуляції (P^h, P);
    h: = h+1;
  until виконається умова завершення;
  x: = arg min {f(u): u∈P};
  return x;
end
```

Схема 8.1. Псевдокод *H*-алгоритмів

Параметри конкретного алгоритму можуть змінюватися від ітерації до ітерації, тобто визначатися динамічно з урахуванням перебігу процесу пошуку екстремумів.

Подібно повторюваному ЛП чи МА, пропоновані алгоритми оперують локальними екстремумами. Їх множина P грає роль, аналогічну популяції в ГА чи МА. Тому використані три процедури відбору можуть реалізовуватися за аналогією з еволюційними алгоритмами. Принципова відмінність H-алгоритмів – глобальний характер пошуку в просторі розв'язків X шляхом знаходження субоптимального розв'язку задачі вигляду

$$z = \arg \min_{u \in <x, x^{\infty}/ \setminus L_s(y).} f(u), \quad y \in <x, x^{\infty}/. \qquad (8.1)$$

При практичному застосуванні H-алгоритмів через те, що відібрані точки x та y є локальними екстремумами, часто виникала ситуація, коли $z = y$. Для уникнення цього вводиться параметр $s > 0$ – значення радіуса околу точки y, точки якого виключаються із розгляду в підзадачі (8.1). Зрозуміло, що для більшості комбінаторних просторів доцільно вибирати $s>1$, зменшуючи тим імовірність визначення точки y як такого субоптимального розв'язку. Як і в алгоритмах МВС, АІВ чи G-алгоритмах, при розв'язанні підзадачі (8.1) можна уникнути повторного обчислення значень цільової функції: варто обчислювати лише різницю $\Delta = f(x) - f(y)$, ураховуючи той факт, що відрізок складають лише сусідні точки (як уже зазначалося, у багатьох випадках обчислення різниці має суттєво нижчу трудомісткість).

На відміну від операторів схрещування чи інших неструктурованих операторів рекомбінації, пропонований підхід не дозволяє "злипатися" точкам поточної популяції P, тому зникає необхідність у диверсифікації результатів, яка пропонується в МА.

Після етапів варіації та мутації утворюється тимчасова популяція P, кількість точок у якій обмежена величиною $m + k + l$. Завданням процедури *ВідбірПопуляції* є зведення кількості точок у P знову до величини m. Важливо при цьому, як і в ГА чи алгоритмах ОМК, використовувати принцип елітизму, щоб не втратити у процесі пошуку рекордні значення (див. розд. 5).

Використання наведеної схеми може породжувати сім'ю алгоритмів комбінаторної оптимізації. Зауважимо, що при $m=1$, $k=0$, $l>0$ отримуємо обчислювальну схему повторюваних G-алгоритмів, а при $l=0$ – алгоритм без використання збурень (мутацій).

Зазначимо також, що механізм породження субоптимального розв'язку на півінтервалі дозволяє провести віддалену аналогію з алгоритмом розсіяного пошуку: процес розв'язання задачі (8.1) можна уявляти як дію специфічного оператора рекомбінації, який формує нащадка від трьох батьків – точок x, y, $x^{\infty}\in X$.

Інший шлях породження алгоритмів – за рахунок можливих способів конкретизації (з урахуванням специфіки задачі, яка розв'язується) ключових аспектів загальної схеми, наведеної на рис. 8.4.

8.4. Основні аспекти реалізації *H*-алгоритмів

Загальна схема H-алгоритмів дає змогу широко варіювати ключові аспекти. Зупинимося на найуживаніших підходах до їх конкретизації.

1) Правила *відбору чергової пари точок* $x, y\in P$ для варіації:

- вибір таких точок, які відповідають найкращому й найгіршому значенням функції придатності серед точок P;
- випадковий вибір точок з імовірністю, пропорційній придатності чи іншим показникам;
- вибір найгіршої точки x і випадковий вибір точки y; при цьому слід ураховувати також віддаленість точок x та y – як у просторі варіантів розв'язку задачі, так і за значеннями цільової функції.

У всіх випадках доцільно, як показав досвід розв'язання ЗКО, за початкову точку x вибирати ту точку з відібраної пари, у якої гірше значення цільової функції (утворювати спадний промінь).

2) *Відбір для мутації*: у найпоширенішому випадку використовується випадковий відбір l точок. Сама мутація полягає у збуренні кількох компонентів вибраного розв'язку.

3) Стратегія формування нової популяції (*ВідбірПопуляції*), яка має складатися із m точок, з тимчасової популяції обсягом $(m+k+l)$ точок:

- відбір m найпридатніших точок;

- включення нащадків на основі критерію Метрополіса;
- заміна *всіх m батьків кращими з (k + l) нащадків, m≤k + l.*

При визначенні для конкретної точки з P функції придатності окрім значення цільової функції (як це робиться в ГА) доречно додатково враховувати ще й такі показники:

- кількість процедур відбору, у яких точку вже було вибрано для варіації;
- кількість нащадків точки, що відбиралися в оновлені популяції;
- час життя: кількість ітерацій, протягом яких точка не вибувала з множини P.

4) Критерієм *завершення обчислювального процесу* може бути:

- перевищення заданої кількості $H_{\max}$ оновлень множини P: алгоритм завершує роботу, коли $h > H_{\max}$;
- стабілізація середнього значення цільової функції в точках множини P:

$$\overline{f} = \sum_{x \in P} f(x) / m;$$

- прямування до нуля розкиду значень цільової функції в точках множини P: $\sigma = \max\{f(x): x \in P\} - \min\{f(x): x \in P\}$;
- усі пари точок уже взяли участь у формуванні півпроменів, якщо використовується однозначний спосіб побудови $<x,x^{\infty}/$.

5) *Спосіб побудови півінтервалів* $<x, x^{\infty}/$. Найчастіше мають справу зі скінченними комбінаторними метричними просторами X, для яких характерні дві особливості.

По-перше, для довільного $x \in P$ максимально віддаленою є точка, що задовольняє умову: $x^{\infty} = \max\{d(x,u): u \in X\}$, де $d(x,u)$ – метрика на X. Як уже зазначалося, часто це значення дорівнює діаметру простору X.

По-друге, між двома несусідніми точками $x, y \in P$ можна побудувати не один, а кілька d-відрізків. Інколи пропонують увести поняття *відрізок з позначкою* для певного впорядкування множини d-відрізків між заданими двома точками й виокремлення конкретного d-відрізка. Тому при реалізації H-алгоритмів слід розрізняти випадок, коли з множини можливих d-відрізків $|x, y|$, $d(x, y)>2$, завжди за конкретним правилом/способом побудови, вибирається лише один відрізок, чи коли можуть роз-

глядатися/будуватися всі можливі відрізки. Залежно від цього може модифікуватися і правило *ВідбірДляВаріації*: якщо розглядається один можливий півінтервал $<x, x^{\infty}/$, причому $y \in <x, x^{\infty}/$, то вершини x, y після відбору слід позначити як відпрацьовані, оскільки їх повторний вибір стає недоречним. Якщо ж є можливість будувати всі чи кілька можливих півінтервалів $<x, x^{\infty}/$, то повторний відбір деякої пари x, y стає доцільним, якщо при цьому буде забезпечено побудову нового півінтервалу.

При розв'язанні задач великої розмірності процедура оптимізації вздовж півінтервалу може виявитися досить трудомісткою. У таких випадках можна ввести додатковий параметр алгоритму, який би визначав ту частину півінтервалу $<x, x^{\infty}/$, яку слід переглядати в задачі (8.1), тобто максимально можливу віддаленість точок x та x^{∞}. Доречно зауважити, що аналогічний параметр обов'язково слід вводити у випадку, коли простір X є нескінченним, оскільки тоді точка x^{∞} прямує у нескінченність, а півінтервал перетворюється на промінь.

Пропонована обчислювальна схема може бути використана для розробки алгоритмів розв'язання широкого кола ЗКО. Вона дає змогу реалізовувати в прикладних програмних комплексах розпаралелювання обчислень на БОК.

Важливим напрямом розвитку H-алгоритмів є гібридизація їх з іншими методами комбінаторної оптимізації з метою отримання метаевристик вищих порядків, зокрема кооперативних. Ефективність такого поєднання була підтверджена при розв'язанні практичних ЗКО.

9. РОЙОВИЙ ІНТЕЛЕКТ У КОМБІНАТОРНІЙ ОПТИМІЗАЦІЇ

9.1. Метод оптимізації роєм частинок

Ройовий інтелект – це дисципліна, яка має справу з природними та штучними системами, що складаються з багатьох агентів (елементів, індивідів, частинок), координація/узгодження дій між якими здійснюється шляхом децентралізованого контролю та самоорганізації. Зокрема, ця дисципліна досліджує колективну поведінку, яка є результатом локальних взаємодій агентів між собою та з навколишнім середовищем. Прикладами таких систем є колонії мурах і термітів, косяки риб, зграї птахів, стада наземних тварин. Деякі розробки людини також належать до сфери ройового інтелекту, зокрема окремі мультироботні системи чи деякі комп'ютерні програми, що написані для розв'язання задач оптимізації та аналізу даних.

Ідея методу оптимізації роєм частинок (ОРЧ), який був сформульований Дж. Кеннеді та Р. Ебергартом (J. Kennedy, R. Eberhart) у 1995 р., навіяна схемою соціальної взаємодії та спілкування, властивої окремим видам птахів і риб. Варто зазначити, що метод ОРЧ – це інструмент не тільки оптимізації, але й подання соціального пізнання (пізнавальної здатності) людини та штучних агентів, яке базується на принципах соціальної психології. Окремі науковці вважають, що знання оптимізується соціальною взаємодією, а мислення є не тільки особистісним процесом, але й міжособистісним.

ОРЧ як оптимізаційний інструмент є популяційною пошуковою процедурою, де агенти, які називаються частинками, взає-

модіючи між собою, змінюють свої позиції (положення, стани) з часом з метою досягнення поставлених цілей.

У схемі ОРЧ частинки рухаються в багатовимірному просторі пошуку задачі, що розв'язується. Під час руху кожна частинка коригує своє положення (варіант розв'язку) з урахуванням свого досвіду та досвіду сусідньої(іх) частинки(ок), використовуючи при цьому найкраще з положень, знайдених нею, і найкращі положення, знайдені її сусідами. Таким чином, метод ОРЧ комбінує локальний пошук із глобальним пошуковим методом, намагаючись збалансувати стратегії інтенсифікації та диверсифікації пошуку.

ОРЧ спочатку було запропоновано як засіб неперервної оптимізації, але пізніше його було модифіковано для застосування до розв'язання ЗКО. Тому спочатку розглянемо класичну схему ОРЧ, після якої буде викладено схему ОРЧ для розв'язання різних ЗКО.

Базові принципи ОРЧ описуються доволі просто. Множина частинок (варіантів розв'язків), що можуть рухатися, "закидається" у простір пошуку. Кожна частинка володіє такими властивостями:

- позицією та швидкістю;
- знає своє поточне положення та значення цільової функції свого поточного положення;
- пам'ятає своє найкраще попереднє положення;
- знає своїх сусідів, їхнє найкраще попереднє (або поточне) положення та відповідні значення цільової функції.

На кожному кроці поведінка певної частинки є компромісом між:

- продовженням свого руху;
- рухом до свого найкращого (у розумінні цільової функції) положення;
- рухом до найкращої частинки з околу або до найкращого попереднього положення сусідніх частинок.

Формалізується цей компроміс співвідношеннями

$$V_{i,t+1} = c_1 V_{i,t} + c_2 \left(P_{i,t} - X_{i,t} \right) + c_3 \left(G_{i,t} - X_{i,t} \right), \qquad (9.1)$$

$$X_{i,t+1} = X_{i,t} + V_{i,t+1}, \qquad (9.2)$$

де $X_{i,t}$ – положення частинки i на ітерації (момент часу) t, $V_{i,t}$ – швидкість частинки i на ітерації t, $P_{i,t}$ – найкраще попереднє положення частинки i на ітерації t, $G_{i,t}$ – найкраще попереднє положення найкращої частинки, сусідньої до частинки i на ітерації t, c_1, c_2, c_3 – соціально-когнітивні довірчі коефіцієнти. Зазвичай ці коефіцієнти на кожній ітерації обираються довільно з наперед заданих інтервалів і відповідно визначають, наскільки частинка "довіряє" собі, власному досвіду та своїм сусідам.

Ідейно $P_{i,t}$ нагадує автобіографічну пам'ять, що відображає, як кожний індивід пам'ятає свій попередній досвід (навіть якщо це лише один факт із нього), а коригування швидкості відповідно до $P_{i,t}$ називається *простою ностальгією*, оскільки індивід схиляється до повернення на місце, яке його найбільше задовольняло. З іншого боку, $G_{i,t}$ ідейно подібне до загальновідомого знання норм для групи індивідів або ж стандарту, якого ці індивіди прагнуть досягти.

Допустимі способи визначення околу можна виділити у два класи:

➢ Геометричні (фізичні) околи, які враховують відстані. На практиці такі відстані можуть перераховуватися на кожній ітерації, що вкрай ресурсоємно, хоча деякі методи кластеризації потребують подібної інформації.

➢ Соціальні околи, які враховують взаємозв'язки індивідів чи часток. На практиці для кожної частинки її окіл одразу визначається як список частинок й не змінюється протягом роботи алгоритму. Зазначимо, що під час обчислювального процесу соціальний окіл зазвичай еволюціонує до геометричного.

Незважаючи на схожість між ОРЧ та еволюційними методами, обчислювальна схема ройових алгоритмів має суттєві відмінності, оскільки, зокрема, у ній відсутні оператори відбору (такі як вибір індивідів для наступної ітерації). Усі члени популяції зберігаються протягом усього процесу пошуку для того, щоб актуальна інформація була соціально розподілена поміж

індивідами для спрямовування пошуку до найкращої знайденої точки у просторі розв'язків.

Псевдокод алгоритму ОРЧ наведено на схемі 9.1. Тут Max*Iter* – це параметр, який визначає кількість ітерацій, після виконання яких алгоритм завершує роботу.

```
procedure PSO
 iter := 1;
 foreach i do
 X_{i,0} := ІніціалізаціяПозиції();
 V_{i,0} := ІніціалізаціяШвидкості();
 endfor
 while iter ≠ MaxIter do
 foreach i do
 P_{i,iter} := ОсобистийРекорд();
 G_{i,iter} := РекордОточення(i);
 endfor
 foreach i do
 V_{i,iter} := ОновитиШвидкість( );
 X_{i,iter} := ОновитиПозицію( );
 endfor;
 iter := iter + 1;
 endwhile;
end
```

Схема 9.1. Псевдокод алгоритму оптимізації роєм частинок

9.2. Особливості ОРЧ для розв'язання ЗКО

Для застосування ОРЧ до задачі оптимізації необхідно конкретизувати, як уже зазначалося раніше:

- *простір* (варіантів) розв'язків X;
- дійснозначну *цільову функцію* f, визначену на X;

➢ якщо будуть використовуватися "фізичні" околи, то необхідно визначити на *X функцію відстані* (метрику) чи іншу процедуру для визначення сусідства.

Крім того, необхідно встановити такі базові об'єкти та операції:

1) позицію частинки;

2) швидкість частинки;

3) операцію переміщення, яка, будучи застосована до позиції та швидкості, зумовлює нову позицію (єдиним критерієм до визначення цієї операції є зв'язність простору операціями переміщення);

4) операцію різниці двох позицій, що визначається як швидкість, яка знаходиться певним алгоритмом і є такою, що операція переміщення за цією швидкістю переводить частинку з першої позиції у другу;

5) операцію множення швидкості на дійсне число;

6) операцію додавання двох швидкостей.

Розглянемо задачу пошуку мінімуму цільової функції, заданої на просторі перестановок, для чого конкретизуємо поняття, які використовуються в алгоритмах ОРЧ.

1) Позицією частинки є перестановка.

2) Швидкість – це впорядкована послідовність індексів транспозицій (пар різних індексів):

$$V = \left((i_1, j_1),\ldots,(i_k, j_k)\right),\ i_l \neq j_l\ ,\ i_l, j_l \in \{1,\ldots,n\},\ l = 1,\ldots,k\ ,$$

де k – кількість транспозицій у послідовності, тобто $\|V\| = k$.

Нульовій швидкості буде відповідати порожній список: $V = \varnothing$. Протилежною швидкістю оголошується інвертований список транспозицій: $\overleftarrow{V} = \left((i_k, j_k),\ldots,(i_1, j_1)\right)$.

3) Операція переміщення $x' = x + V$ – це послідовне застосування до перестановки (позиції) x транспозицій зі списку (швидкості) V.

4) Операція різниці двох позицій $y - x = V$ – це швидкість V, яка визначається за деяким алгоритмом, так що $y = x + V$. Умова "за деяким алгоритмом" необхідна, оскільки в загальному випа-

дку швидкостей, що переводитимуть одну позицію в іншу, може бути кілька. Зокрема, алгоритм визначення має бути таким, щоб

$$y - x = \overleftarrow{(x - y)} \text{ та } y = x \Rightarrow V = x - y = (\ ).$$

5) Множення cV швидкості на дійсне число c, якщо $V = \left((i_1, j_1), \ldots, (i_k, j_k)\right)$, залежно від c визначимо так:

$$c = 0 : cV = (\ );$$

$$c \in (0;1] : V = \left((i_1, j_1), \ldots, \left(i_{\lfloor ck \rfloor}, j_{\lfloor ck \rfloor}\right)\right);$$

$$c > 1 : l = \lfloor c \rfloor,\ c = l + c',\ c' \in (0;1];$$

$$cV = \underbrace{V \oplus \ldots \oplus V}_{l} \oplus c'V;$$

$$c < 0 : cV = (-c)\overleftarrow{V},$$

де $\overleftarrow{V}$ – це інвертований список транспозицій V, а $\lfloor z \rfloor$ – найбільше ціле число, яке не перевищує z.

6) Операція додавання двох швидкостей $V_1 \oplus V_2$ – це список транспозицій, що спочатку містить список транспозицій V_1, а потім – V_2:

$$V_1 = \left((i_1^1, j_1^1), \ldots, (i_k^1, j_k^1)\right),\ V_2 = \left((i_1^2, j_1^2), \ldots, (i_k^2, j_k^2)\right) \Rightarrow$$

$$\Rightarrow V_1 \oplus V_2 = \left((i_1^1, j_1^1), \ldots, (i_k^1, j_k^1), (i_1^1, j_1^1), \ldots, (i_k^1, j_k^1)\right).$$

Відстані можуть базуватися на тій чи іншій метриці (див. розд. 1). Тоді рівняння (9.1) – (9.2) можна переформулювати у вигляді (9.3) – (9.4):

$$V_{i,t+1} = c_1 V_{i,t} \oplus c_2 \left(P_{i,t} - X_{i,t}\right) \oplus c_3 \left(G_{i,t} - X_{i,t}\right), \tag{9.3}$$

$$X_{i,t+1} = X_{i,t} \oplus V_{i,t+1}. \tag{9.4}$$

Таким чином, алгоритми ОРЧ є розвитком одночасно як еволюційного підходу, так і клітинних автоматів та характеризуються простотою реалізації. Вище наведено базовий

варіант обчислювальної схеми, на основі якої можуть розроблюватися спеціалізовані алгоритми розв'язання конкретних задач оптимізації. Наприклад, відомі модифікації ОРЧ, які враховують значення цільової функції для всіх частинок рою; у деяких з них частинки групуються в кілька роїв, що фактично ініціює утворення гіперевристик на кшталт острівних моделей у алгоритмах ОМК. І хоча алгоритми ОРЧ у багатьох випадках програють спеціалізованим алгоритмам, відомі дані про успішне їх застосування для розв'язання різних типів спеціальних ЗКО.

10. БДЖОЛИНІ АЛГОРИТМИ

10.1. Основні поняття

Серед біологічно навіяних алгоритмів комбінаторної оптимізації важливе місце займають бджолині, які базуються на результатах спостереження за бджолиними роями при пошуку ними нектару. Особливості поведінки бджолиних колоній можуть бути використані для моделювання інтелектуальної та колективної поведінки.

Основними поняттями у бджолиних алгоритмах є (табл. 1.10):

- джерела нектару;
- фуражири (робочі бджоли);
- бджоли-розвідники.

Таблиця 10.1

Відповідність термінів у природних і штучних бджолиних роях

Природна бджолина колонія	Штучна бджолина колонія
Джерело нектару	Розв'язок
Якість нектару	Цільова функція
Спостерігачі	Користувачі характеристик пошуку
Розвідники	Пошук розвідкою

Загалом поведінка бджіл у процесі пошуку нектару може бути описана таким чином: спочатку з вулика у випадкових напрямках вилітають бджоли-розвідники, які намагаються відшукати місця, де є нектар (формують пробні розв'язки). Через деякий час ці бджоли повертаються у вулик і особливим чином (бджолиним танцем) повідомляють іншим, де знайшли нектар. Після цього на знайдені місця вирушають робочі бджоли – фуражири. Чим більше в конкретному місці очікується зібрати нектару, тим більше

фуражирів летить у цьому напрямку, а розвідники відлітають шукати інші місця, після чого процес повторюється. Середня кількість розвідників у рої зазвичай становить 10–15 %.

10.2. Моделі поведінки у бджолиних алгоритмах

Існує два типи алгоритмів: алгоритми штучної бджолиної колонії (Artificial Bee Colony), автор – Д. Карабога, та оптимізації бджолиними колоніями (Bee Colony Optimization), запропоновані Д. Теодоровічем. Наведемо основні положення моделі, покладеної в основу цих алгоритмів.

На початку роботи алгоритмів рою бджіл для визначення місць, де містяться джерела нектару, у випадкових напрямках вирушають бджоли-розвідники. Як уже зазначалося, інформація від цих бджіл отримується шляхом виконання розвідниками танцю, у якому описується відстань, напрямок і кількість їжі.

Основними етапами алгоритмів штучного бджолиного рою є:

- відправлення робочих бджіл до джерел нектару;
- установлення кількості наявного нектару;
- обчислення значення ймовірності для розробки джерела нектару;
- зупинка експлуатації джерел, до яких припинили літати бджоли;
- виліт бджіл-розвідників у випадкових напрямках для пошуку нових джерел живлення;
- запам'ятовування кращих серед знайдених джерел;
- вихляючий танок.

Якщо за встановлену кількість кроків не знайдено кращого розв'язку, то джерело живлення вилучається з розгляду, а зайняті на ньому бджоли перетворюються на розвідників.

Отже, загальний опис бджолиного алгоритму є таким:

1. Генерування місць для пошуку нектару.
2. Оцінювання корисності місць.
3. Вибір місць для пошуку в їх околах.
4. Відправлення фуражирів.
5. Пошук в околах джерел нектару.

6. Відправлення бджіл-розвідників.
7. Випадковий пошук.
8. Оцінювання корисності нових місць.
9. Якщо умови зупинки не виконуються, то перехід на крок 2.
10. Завершення роботи.

Формально бджолині алгоритми показані на схемі 10.1.

procedure *BeeAlgorithm*;
Випадкова ініціалізація колонії бджіл;
Оцінювання придатності популяції бджіл;
repeat
Вибір напрямку для пошуку в околі;
Визначення розміру джерела;
Набір бджіл для вибраних напрямків та оцінювання їх придатності;
Вибір представника для кожного напрямку для формування наступної Популяції бджіл;
Випадковий розподіл бджіл-розвідників та оцінювання їх придатності;
until критерій зупинки;
end

Схема 10.1. Схема бджолиних алгоритмів

Особливістю зазначених алгоритмів є те, що бджоли-фуражири прямують не точно на те саме місце, у якому бджоли-розвідники знайшли перспективні джерела нектару, а в їх околи, уточнюючи координати місць випадковим чином.

10.3. Алгоритм BCO

Розглянемо алгоритм оптимізації бджолиною колонією BCO (Bee Colony Optimization) для розв'язання ЗКО. Відповідно до нього, штучні бджоли живуть у середовищі з дискретним часом і спільно шукають оптимальний розв'язок задачі. Кожна штучна бджола генерує окремий розв'язок.

Кожен крок алгоритму ВСО має такі дві складові:

1. Проходження вперед. Під час виконання цієї складової кожна штучна бджола створює частковий розв'язок, відвідуючи можливі компоненти розв'язку задачі, і повертається до вулика. Важливою особливістю алгоритму є те, що вулик не має точного розташування й не впливає на виконання алгоритму, а є тільки точкою синхронізації та обміну інформацією про поточний стан пошуку. Кількість компонент, які відвідує бджола за один прохід, установлюється дослідником.

2. Зворотний прохід. На цьому етапі всі штучні бджоли обмінюються інформацією про якість знайдених ними розв'язків. Кожна бджола, оперуючи всіма наявними розв'язками, з певною ймовірністю вирішує, чи буде вона працювати над "своїм" напрямом. Бджоли з кращими розв'язками мають більше шансів зберегти їх та залучити до їх розвитку інших бджіл.

Псевдокод алгоритму ВСО зображений на схемі 10.2.

```
procedure BCO;
 Задання кількості бджіл B;
 Задання кількості кроків N_C;
1: Ініціалізація: кожній бджолі присвоюється порожній розв'язок;
2: repeat
3: k: = 1;                {Кількість конструктивних кроків}
4: Оцінка всіх можливих конструктивних кроків;
5: Вибір одного кроку;
6: k: = k+1;
 if k ≤ N_C then goto 4;
7: Повернення бджіл у вулик; {зворотний прохід}
8: Оцінювання значення цільової функції для кожної бджоли;
9: Перегрупування бджіл;
10: Вибір нових рішень для кожного послідовника;
11: if розв'язок не сформовано then Goto 2;
12: Оцінювання розв'язків;
13: until умови зупинки;
end
```

Схема 10.2. Псевдокод алгоритму ВСО

Дослідники пропонують такі підходи до реалізації окремих кроків алгоритму:

➢ Крок 9: імовірність того, що бджола b залишиться лояльною до знайденого нею частинного розв'язку, обчислювати за формулою

$$p_b^{u+1} = e^{-\frac{C_{\max} - C_b}{u}}, \; b = 1,...,B,$$

де u – кількість проходів вперед; $C_{\max}$ – максимальне значення серед нормованих частинних розв'язків; C_b – частинний розв'язок відповідної бджоли.

➢ Крок 10: імовірність того, що частинний розв'язок бджоли b буде успадкований бджолами-послідовниками, визначається формулою

$$p_b = \frac{C_b}{\sum_{k=1}^{R} C_k}, \; b = 1,...,R,$$

де R – кількість бджіл, які залишилися лояльними до своїх розв'язків.

ПІСЛЯМОВА

Незважаючи на більш ніж півстолітню історію, і нині триває бурхливий розвиток теорії та методів комбінаторної оптимізації. Головна причина – розширення сфери її застосування й необхідність у розв'язанні задач підвищеної розмірності.

Коротко підсумуємо викладене. Загальний тренд розвитку алгоритмів комбінаторної оптимізації можна подати як послідовність: прості алгоритми → комбіновані алгоритми → метаевристики → гібридні метаевристики → кооперативні алгоритми. Утім, із цього не випливає пріоритетність того чи іншого класу алгоритмів: для конкретної ЗКО відкритим є питання, розробка якого з алгоритмів буде найбільш вигідною й доцільною, враховуючи всі умови задачі та супутні аспекти.

До основних факторів, які впливають на вирішення проблеми вибору алгоритму розв'язання, слід, у першу чергу, віднести:

- складність математичної моделі задачі, яка розв'язується;
- розмірність і трудомісткість задачі;
- наявність директивних термінів отримання розв'язку;
- можливість використання наявного програмного забезпечення;
- доступні засоби обчислювальної техніки, їх продуктивність;
- наявність кваліфікованих фахівців.

Незважаючи на накопичений досвід, розробка й упровадження АКО вимагає як наявності базових знань, так і творчого підходу: складність поєднання цих аспектів якраз визначає і успішність фахівців, і велике задоволення від процесу розв'язування ЗКО та отриманих результатів. Саме такому синергетичному ефекту має сприяти пропонована книга.

ЛІТЕРАТУРА

1. Волошин А. Ф. Последовательный анализ вариантов в задачах исследования сложных систем : монография / А. Ф. Волошин, В. И. Кудин. – К. : Изд.-полиграф. центр "Киев. ун-т", 2015.

2. Глибовець М. М. Еволюційні алгоритми : підручник / М. М. Глибовець, Н. М. Гуляєва. – К. : НаУКМА, 2013.

3. Гуляницкий Л. Ф. Метаэвристический метод деформированного многогранника в комбинаторной оптимизации / Л. Ф. Гуляницкий, И. В. Сергиенко // Кибернетика и системный анализ. – 2007. – № 6. – С. 70–79.

4. Гуляницкий Л. Ф. Решение задач комбинаторной оптимизации алгоритмами ускоренного вероятностного моделирования / Л. Ф. Гуляницкий // Компьютерная математика. – К. : Ин-т кибернетики им. В. М. Глушкова НАН Украины, 2004. – № 1. – С. 64–72.

5. Гуляницький Л. Ф. Застосування *H*-методу для розв'язання задач комбінаторної оптимізації на перестановках / Л. Ф. Гуляницький, Д. А. Гобов // Системні дослідження та інформаційні технології. – 2007. – № 2. – С. 74–87.

6. Гэри М. Вычислительные машины и труднорешаемые задачи / М. Гэри, Д. Джонсон. – М. : Мир, 1982.

7. Пападимитриу Х. Комбинаторная оптимизация. Алгоритмы и сложность / Х. Пападимитриу, К. Стайглиц. – М. : Мир, 1985.

8. Сергиенко И. В. Классификация прикладных методов комбинаторной оптимизации / И. В. Сергиенко, Л. Ф. Гуляницкий, С. И. Сиренко // Кибернетика и системный анализ. – 2009. – № 5. – С. 71–83.

9. Сергиенко И. В. Математические модели и методы решения задач дискретной оптимизации / И. В. Сергиенко. – К. : Наук. думка, 1988.

10. Сергиенко И. В. Модели и методы решения на ЭВМ комбинаторных задач оптимизации / И. В. Сергиенко, М. Ф. Каспшицкая. – К. : Наук. думка, 1981.

11. Сергиенко И. В. Задачи дискретной оптимизации / И. В. Сергиенко, В. П. Шило. – К. : Наук. думка, 2003.

12. Субботін С. О. Неітеративні, еволюційні та мультиагентні методи синтезу нечіткологічних і нейромережних моделей : монографія / С. О. Субботін, А. О. Олійник, О. О. Олійник ; під заг. ред. С. О. Субботіна. – Запоріжжя : ЗНТУ, 2009.

13. Dorigo M. Ant colony optimization theory: A survey / M. Dorigo, C. Blum // Theoretical Computer Science. – 2005. – 344. – P. 243–278.

14. Handbook of Combinatorial Optimization / eds. P. Pardalos, D.-Z. Du, R. L. Graham. – 2nd ed. – Heidelberg : Springer. – 2013, XXI.

15. Handbook of Memetic Algorithms / eds. F. Neri, C. Cotta, P. Moscato. – Berlin, Heidelberg : Springer-Verlag, 2012.

16. Hoos H. H. Stochastic Local Search: Foundations and Applications / H. H. Hoos, T. Stützle. – San Francisco : Morgan Kaufmann Publ., 2005.

17. A comprehensive survey: artificial bee colony (ABC) algorithm and applications / D. Karaboga, B. Gorkemli, C. Ozturk, N. Karaboga // Artificial Intelligence Review. – 2014. – 42(1). – P. 21–57.

18. Lourenço H. R. Iterated local search / H. R. Lourenço, O. Martin, T. Stützle // Handbook of Metaheuristics: International Series in Operations Research & Management Science, vol. 57 (Eds. F. Glover and G. Kochenberger). – Norwell, MA : Kluwer Academic Publishers, 2002. – P. 321–353.

19. Pintea C.-M. Advances in Bio-inspired Computing for Combinatorial Optimization Problems / C.-M. Pintea. – Heidelberg : Springer, 2014.

20. Talbi E.-G. Metaheuristics: from design to implementation / E.-G. Talbi. – Hoboken, New Jersey : John Wiley & Sons, Inc., 2009.

21. Nikolić M. Empirical study of the bee colony optimization (BCO) algorithm / M. Nikolić, D. Teodorović // Expert Systems with Applications. – 2013. – 40(11). – P. 4609–4620.

ЗМІСТ

Навчальне видання

ГУЛЯНИЦЬКИЙ Леонід Федорович
МУЛЕСА Оксана Юріївна

ПРИКЛАДНІ МЕТОДИ КОБІНАТОРНОЇ ОПТИМІЗАЦІЇ

Навчальний посібник

Редактор *Н. М. Земляна*

Оригінал-макет виготовлено ВПЦ "Київський університет"

Виконавець *Т. С. Яшкова*

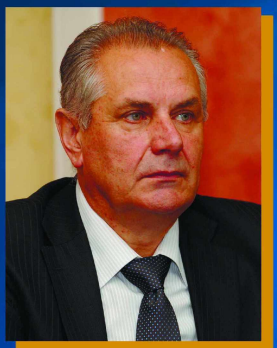
Формат 60x84$^{1/16}$. Ум. друк. арк. 8,4. Наклад 300. Зам. № 216-7877.

Гарнітура Times New Roman. Папір офсетний. Друк офсетний. Вид. № К3.

Підписано до друку 27.07.16

Видавець і виготовлювач
ВПЦ "Київський університет",
б-р Т. Шевченка, 14, м. Київ, 01601
☎ (044) 239 32 22; (044) 239 31 72; тел./факс (044) 239 31 28
e-mail: vpc@univ.kiev.ua
http: vpc.univ.kiev.ua
Свідоцтво суб' єкта видавничої справи ДК № 1103 від 31.10.02

Гуляницький Леонід Федорович



Доктор технічних наук, завідувач відділу Інституту кібернетики ім.В.М.Глушкова НАН України, професор ФІОТ НТУУ "КПІ" та факультету кібернетики Київського державного університету ім.Тараса Шевченка. Лауреат Державної премії України в галузі науки та техніки, премій ім.В.М.Глушкова НАН України та Республіканської премії ім.М.Островського в галузі науки та техніки для молодих учених.

Автор понад 185 наукових статей і трьох монографій. Член редколегій журналів International Journal "Information Theories and Applications", International Journal "Information Models and Analyses", "Математичне моделювання в економіці", а також періодичного збірника "Теорія оптимальних рішень". Науковий керівник багатьох наукових проектів, керівник української команди європейського проекту INTAS (2007-2008 рр.).

Сфера наукових інтересів: моделі та методи комбінаторної оптимізації; підтримка прийняття й оптимізація рішень в технічних та економічних системах; паралельні алгоритми; розробка та впровадження інформаційних технологій; проблеми використання штучного інтелекту в методах оптимізації.

Мулеса Оксана Юріївна

Кандидат технічних наук, доцент кафедри кібернетики і прикладної математики ДВНЗ "Ужгородський національний університет".

Автор понад 50 наукових та науково-методичних праць.

Член оргкомітету Міжнародної школи-семінару "Теорія прийняття рішень".

Веде лекційні курси, серед яких "Методи комбінаторної оптимізації", "Основи інтелектуальних обчислень", "Сучасні інформаційні технології", "Бази даних та інформаційні системи", "Системи і методи прийняття рішень".

Наукові інтереси: моделі та методи комбінаторної оптимізації; розробка та впровадження інформаційних технологій; еволюційне моделювання; моделі та методи прийняття рішень в умовах нечіткої інформації.