Розділ 5.

ЕВОЛЮЦІЙНІ ТЕХНОЛОГІЇ ТА ГЕНЕТИЧНІ АЛГОРИТМИ

5.1. Концептуальні засади еволюційної теорії.

Еволюційна теорія, синонімом якої в зарубіжній літературі є термін «Evolutionary computation», довела свою ефективність як при вирішенні складно формалізованих задач штучного інтелекту (розпізнавання образів, кластеризація, асоціативний пошук), так і при вирішенні трудомістких задач оптимізації, апроксимації, інтелектуальної обробки даних. До переваг еволюційної теорії відносяться адаптивність, здібність до навчання, паралелізм, можливість побудови гібридних інтелектуальних систем на основі комбінування з парадигмами штучних нейромереж і нечіткої логіки. Багатообіцяючою виглядає передумова створення єдиної концепції еволюційних обчислень, що включають генетичні алгоритми, генетичне програмування, еволюційні стратегії і еволюційне програмування. На думку багатьох дослідників, ці парадигми є аналогами процесів, що відбуваються в живій природі, і на практиці довели свою непримітивність. Один з піонерів еволюційної теорії Л. Фогель взагалі бачить теорію еволюції і самоорганізації як базову концепцію для всіх інтелектуальних процесів і систем, що значно розширює сферу застосування традиційної парадигми інтелектуального аналізу даних. Навіть якщо це не так, і в природі відбувається революция, ніхто не може сказати, що алгоритми еволюційних обчислень невірні [7, 12, 18].

Основна теза підходу, названого еволюційним, - замінити процес моделювання складного об'єкту моделюванням його еволюції. Він спрямований на застосування механізмів природної еволюції при аналізі складних систем інтелектуальної обробки інформації. У своїй теорії походження видів Ч. Дарвін відкрив і обґрунтував основний закон розвитку органічного світу, охарактеризувавши його взаємодією трьох наступних чинників:

- спадкоємній мінливості;
- боротьби за існування;
- природного відбору.

Дарвінівська теорія отримала підтвердження і розвиток в генетиці та інших науках. Одним з найвідоміших еволюціоністів, нашим співвітчизником І. І. Шмальгаузеном були висунуті наступні необхідні і достатні умови, що визначають неминучість еволюції:

1. Спадкоємна мінливість, тобто мутації як передумова еволюції, її матеріал.
2. Боротьба за існування як контролюючий і направляючий чинник.
3. Природний відбір як перетворюючий чинник.

Моделювання еволюції можна розділити на дві категорії:

1. Системи, які використовують тільки еволюційні принципи. Вони успішно використовувалися для задач типу функціональної оптимізації і можуть легко бути описані на математичній мові. До них відносяться еволюційні алгоритми, такі як еволюційне програмування, генетичні алгоритми та еволюційні стратегії.

2. Системи, які є біологічно реалістичнішими, але які не виявилися корисними в прикладному сенсі. Вони більш схожі на біологічні системи і менш направлені на вирішення технічних задач та інтелектуального аналізу даних. Вони володіють складною і цікавою поведінкою, і, мабуть, незабаром отримають практичне застосування. До цих систем відносять так зване «штучне життя».

Еволюційні обчислення - термін, зазвичай використовують для загального опису алгоритмів пошуку, оптимізації або навчання, заснованих на формалізованих принципах природного еволюційного процесу. Еволюційні методи призначені для пошуку переважних рішень і засновані на статистичному підході до дослідження ситуацій та ітераційному наближенні до шуканого стану систем. На відміну від точних методів дослідження операцій еволюційні методи дозволяють знаходити рішення, близькі до оптимальних, за прийнятний час, а у відмінності від відомих евристичних методів оптимізації

характеризуються істотно меншою залежністю від особливостей додатку (тобто більш універсальні) і у багатьох випадках забезпечують кращий ступінь наближення до оптимального рішення. Основна перевага еволюційної теорії полягає в можливості вирішення багатомодальних (що мають декілька локальних екстремумів) задач із великою розмірністю за рахунок поєднання елементів випадковості і детермінованості точно так, як це відбувається в природному середовищі.

Детермінованість цих методів полягає в моделюванні природних процесів відбору, розмноження і спадкоємства, що відбуваються по строго певних правилах, основним з яких є закон еволюції - «виживає сильніший». Іншим важливим чинником ефективності еволюційних обчислень є моделювання процесів розмноження і спадкоємство. Дані варіанти рішень можуть за певним правилом породжувати нові рішення, які успадковуватимуть кращі риси своїх «предків». Як випадковий елемент в теорії еволюційних обчислень використовується моделювання процесу мутації. З її допомогою характеристики того або іншого рішення можуть бути випадково змінені, що приведе до нового напряму в процесі еволюції рішень і може прискорити процес вироблення кращого рішення.

Методи еволюційної теорії також часто використовуються для опису процесів еволюції програм або функцій (генетичне програмування), кінцевих автоматів (еволюційне програмування) і систем, заснованих на продукційних правилах (класифікаційні системи). Вони застосовуються для навчання штучних нейронних мереж або разом з нейронними мережами для локального пошуку екстремуму цільової функції, а також часто з нечіткою логікою. Для того, щоб описувати генетичні алгоритми, нейроні мережі і нечітку логіку в їх різних поєднаннях, був навіть введений спеціальний термін «м'які обчислення».

Історія еволюційних обчислень почалася з розробки ряду різних незалежних моделей еволюційного процесу. Серед цих моделей слід виділити декілька основних парадигм: генетичні алгоритми, генетичне програмування, еволюційні стратегії, еволюційне програмування. Поняття «еволюційне

моделювання» вперше сформувалося в роботах Л. Фогеля, А. Оуені, М. Уолша. У 1966 році вийшла їх спільна книга «Штучний інтелект і еволюційне моделювання». У далекі 60-і роки Інго Рехенберг (I. Rechenberg), зацікавлений методом «органічної еволюції», висунув ідею вирішення оптимізаційних проблем в аеродинаміці, застосовуючи мутації до вектора параметрів. Ця процедура стала відома під назвою «еволюційної стратегії» (Evolution Strategies). У 1981 році Швефель (H. Schwefel) при дослідженні гідродинамічних задач увів рекомбінації в еволюційну систему та виконав порівняльний аналіз з класичними методами оптимізації. Приблизно в той же час в США незалежно виконувалися дослідження Лоуренсом Фогелем (Lawrence Fogel) еволюції штучного інтелектуального автомата з скінченим числом станів, використовуючи метод, названий «еволюційним програмуванням» (Evolutionary Programming). Джон Холланд (John Holland) аналізував клас репродуктивних систем методом, який нам відомий як генетичний алгоритм (Genetic Algorithms). Така класифікація еволюційних алгоритмів була б неповною без робіт Lynn Cramer (1985), Jac Hicklin (1986), Gory Fujiki (1987), результати яких узагальнив і розширив Джон Коза (John Koza). Запропонований метод назвали генетичним програмуванням (Genetic Programming) [34, 62, 66, 67]. Еволюційні алгоритми відрізняються один від одного. Але всі вони базуються на принципах еволюції:

1. Особі мають скінчений час життя; розмноження необхідне для продовження роду.
2. В деякій мірі нащадки відрізняються від батьків.
3. Особі існують в середовищі, в якому виживання є боротьбою за існування, і їх зміни сприяють кращій адаптації до умов зовнішнього середовища.
4. За допомогою природної селекції краще адаптовані особі мають тенденцію до довшого життя і виробництва більшої кількості нащадків.
5. Нащадкам властиво успадковувати корисні характеристики своїх батьків, що спричиняє збільшенню пристосованості особі в часі.

Розвиток теорії еволюційних обчислень в нашій країні почався ще з початку 80-х років, проте отримані результати довгий час залишалися, в основному, мало відомими. Ці дослідження в значній мірі базуються на раніше опубліковані роботи по самонавчальних системах (Івахненко, Ципкін), по стохастичній оптимізації (Растрігин). Кожна з цих «шкіл» взяла за основу ряд принципів, що існують в природі, і спростила їх до такого ступеня, щоб їх можна було реалізувати на комп'ютерах того часу. Були розроблені цікаві теорії, щоб описати переваги запропонованих підходів, але всі вони були змушені розділити і загальну проблему того часу - комп'ютери були недостатньо потужними, щоб вирішувати прикладні задачі такої розмірності, які не могли бути розв’язані традиційними методами. Це пояснюється тим, що еволюційні методи не здатні «забиратися» на найближчий «пагорб», виявляти оптимальні навчальні правила і тому подібне з такою ж швидкістю, як це можуть робити інші методи. Проте, коли задача не може бути вирішена іншими, простішими методами, еволюційні обчислювальні методи можуть знайти оптимальні або близькі до них рішення. Необхідний при цьому обсяг підрахунків може виявитися великим, але швидкість, з якою цей обсяг зростає при збільшенні «розмірності» задачі, часто менше, ніж для інших методів. Тому, як тільки обчислювальні потужності стали відносно доступними і недорогими, еволюційні обчислення перетворилися на важливий інструмент пошуку субоптимальних рішень тих задач, які до останнього часу вважалися за нерозв'язні.

В даний час відомою світовою школою, що представляє новий напрям в еволюційному моделюванні, є школа Кандіда Феррера (Candida Ferreira) у Великобританії. Основний напрям її досліджень зосереджений в програмуванні генетичних виразів. Нові алгоритми, які розробляються представниками школи, використовують специфічні оператори комбінаторного пошуку, що включають інверсію, вставку і видалення генів та їх послідовностей, обмеження і узагальнення перестановок, які збільшують їх ефективність. Самі автори визначають програмування генетичних виразів як мультигенне

генотип/фенотип кодування дерев виразів, зв'язаних приватною взаємодією. Іншою відомою школою, в якій досліджують генетичні алгоритми, еволюційні стратегії, генетичне програмування і еволюційне програмування є лабораторія еволюційних обчислень Департаменту комп'ютерних наук в університеті Джорджа Мейсона. У США керівництво школою здійснює учень Джона Холланда К. Де Йонг (K. De Jong). Лабораторія працює над проектами і додатками моделей еволюції (у дарвінівському сенсі). Такі моделі необхідні для кращого розуміння еволюційних систем, вони використовуються для забезпечення робастності, гнучкості і адаптивності обчислювальних систем. Головну увагу фахівці лабораторії приділяють вирішенню складних наукових і технічних проблем, таких, як інноваційне проектування, оптимізація і машинне навчання. У аналогічному напрямі, але з акцентом на генетичні алгоритми працює наукова школа Девіда Голдберга (David E. Goldberg). Лабораторія генетичних алгоритмів знаходиться в Іллінойському університеті США (рис. 5.1).

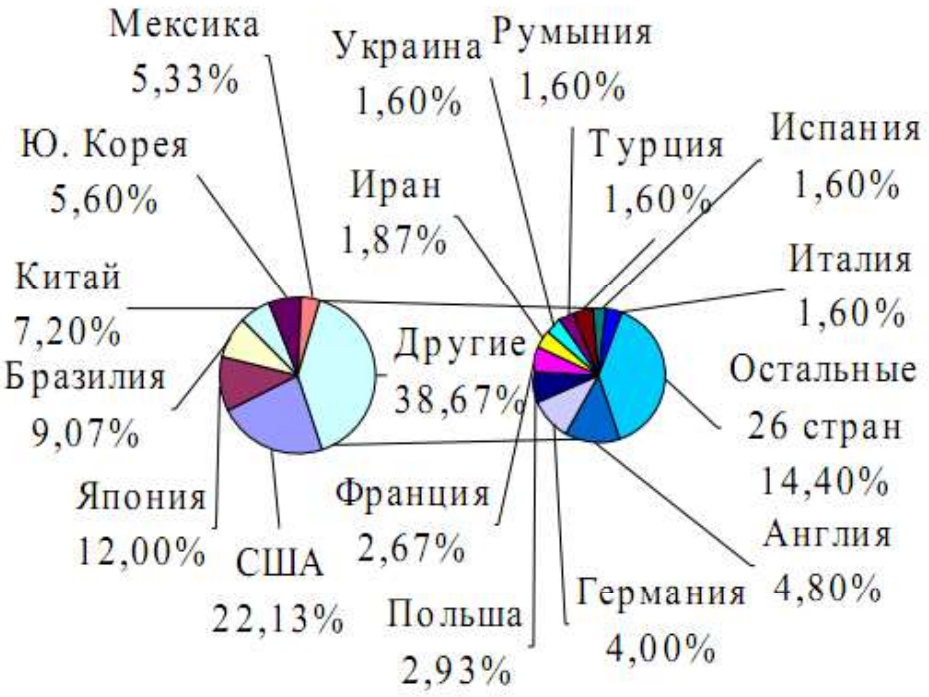


Рис. 5.1. Внесок фахівців різних країн в розвиток теорії еволюційних обчислень за 2000 – 2008 р.

Оскільки концепція еволюційних обчислень має бути заснована на деяких формалізованих принципах природного еволюційного процесу, то виникають питання про методологічні відмінності і сферу застосування основних форм еволюційних технологій. Результати порівняльного аналізу відомих форм еволюційних алгоритмів показують певні методологічні відмінності між ними. Ці відмінності стосуються форми представлення цільової функції і альтернативних рішень, операторів рекомбінації, мутації і ймовірності їх використання, стратегії селективного відбору і методів підвищення ефективності еволюційних обчислень шляхом адаптації.

Методологічні відмінності між різними формами еволюційних обчислень дозволяють говорити про базові постулати, такі як універсальність і фундаментальність, властивих еволюції незалежно від форми і рівня абстракції моделі. Вказана спільність може бути виражена у вигляді наступної схеми абстрактного еволюційного алгоритму:

1. Установка параметрів еволюції;
2. Ініціалізація початкової популяції $P(0)$;
3. $t = 0$;
4. Оцінка рішень, що входять в популяцію;
5. $t = t + 1$;
6. Селекція (відбір);
7. Реплікація (повторення, копіювання, аутосинтез);
8. Варіація (видозміна);
9. Оцінка рішень-нащадків;
10. Утворення нової популяції $P(t)$;
11. Виконання алгоритму до тих пір, поки параметр t не досягне заданого значення $t_{\max}$ або не будуть виконані інші умови зупинення;
12. Виведення результатів і зупинення.

Вирішальною обставиною для оцінки практичної придатності і результативності еволюційного алгоритму є швидкість (час, необхідний для виконання заданого користувачем числа ітерацій) і стійкість пошуку (здатність

постійно від покоління до покоління збільшувати якість популяції та стійкість до попадання в точки локальних екстремумів).

Спроба порівняльного аналізу конкуруючих евристичних алгоритмів з погляду їх результативності демонструє два важливі практичні наслідки:

1. Пошук еволюційного алгоритму, який перевершує всі алгоритми, що конкурують з ним, не має сенсу без точного опису конкретних задач і цільових функцій, для яких еволюційний алгоритм має перевагу перед іншими алгоритмами. Не можна розраховувати знайти один алгоритм, який буде результативніше останніх для будь-яких цільових функцій оптимізації.

2. Щоб знайти добре рішення для заданого класу задач, необхідно спочатку ідентифікувати характеристичні особливості класу задач і тоді на їх основі шукати відповідний алгоритм.

З цієї точки зору еволюційні обчислення володіють наступними достоїнствами і недоліками. Достоїнства еволюційних обчислень:

- широка сфера застосування;
- можливість проблемно-орієнтованого кодування рішень, підбору початкової популяції, комбінування еволюційних обчислень з не еволюційними алгоритмами, продовження процесу еволюції до тих пір, поки є необхідні ресурси;
- придатність для пошуку в складному просторі рішень великої розмірності;
- відсутність обмежень на вигляд цільової функції;
- ясність схеми і базових принципів еволюційних обчислень;
- інтегрованість еволюційних обчислень з іншими некласичними парадигмами штучного інтелекту, такими, як штучні нейромережі і нечітка логіка.

Недоліками еволюційних обчислень є:

- евристичний характер еволюційних обчислень не гарантує оптимальності отриманого рішення (правда, на практиці, часто, важливо за заданий час отримати одне або декілька субоптимальних альтернативних рішень, тим

більше, що початкові дані в задачі можуть динамічно мінятися, бути неточними або неповними);

- відносно висока обчислювальна трудомісткість, яка проте долається за рахунок розпаралелювання на рівні організації еволюційних обчислень і на рівні їх безпосередньої реалізації в обчислювальній системі;
- відносно невисока ефективність на завершальних фазах моделювання еволюції (оператори пошуку в еволюційних алгоритмах не орієнтовані на швидке попадання в локальний оптимум);
- невирішеність питань самоадаптації.

Таким чином, порівняльний аналіз показує, що еволюційні обчислення найменше підходять для вирішення задач, в яких потрібно знайти глобальний оптимум; є ефективний не еволюційний алгоритм; змінні, від яких залежить рішення незалежні; для задачі характерний високий ступінь епістазії (одна змінна пригнічує іншу); значення цільової функції в усіх точках, за винятком оптимуму, є приблизно однаковими. Принципово підходящими для вирішення за допомогою еволюційних обчислень є задачі багатовимірної оптимізації з мультимодальними цільовими функціями, для яких немає відповідних не еволюційних методів рішення; стохастичні задачі; динамічні задачі з блукаючим оптимумом; задачі комбінаторної оптимізації; задачі прогнозування і кластеризації.

Еволюційні обчислювальні методи сьогодні успішно застосовуються для вирішення ряду великих і економічно значущих задач в бізнесі і інженерних розробках. За їх допомогою були розроблені промислові проекти, що дозволили заощадити мільйони доларів. Фінансові компанії широко використовують еволюційні методи прогнозування розвитку фінансових ринків для управління портфелями цінних паперів і так далі Методи еволюційної теорії зазвичай використовуються для оцінки значень безперервних параметрів моделей великої розмірності, для вирішення NP-повних комбінаторних задач, для оптимізації моделей, що включають одночасно безперервні і дискретні параметри, в системах видобування нових знань з великих баз даних (data

mining), для побудови і навчання стохастичних мереж та байєсовських мереж довіри, для навчання нейронних мереж, для оцінки параметрів в багатовимірного статистичного аналізу, а також у поєднанні з різними евристичними процедурами. Сферами застосування таких технологій є проектування механічних пристроїв, розводка друкованих плат і розміщення на них електронних компонентів, проектування деталей реактивних двигунів, розробка складних планів і розкладів, оптимізація розміщення промислового устаткування, а також багаточисельні приклади вирішення оптимізаційних задач великої розмірності звеликим числом обмежень. Ніхто не стверджує, що еволюційні обчислення однаково «гарні» для вирішення всіх типів задач; при цьому є досить багато прикладів їх успішного застосування, хоча успіх зазвичай досягається тільки після ретельного налаштування структури і параметрів еволюційного методу.

Одним з найбільш відомих еволюційних алгоритмів є алгоритм на основі методу групового обліку аргументів (МГОА). Сутність цього методу в тому, що розробляється сімейство індуктивних алгоритмів для моделювання мультипараметричних даних. Метод заснований на рекурсивному селективному відборі моделей, на основі яких будуються складніші моделі. Точність моделювання на кожному наступному кроці рекурсії збільшується за рахунок ускладнення моделі. Згідно термінології теорії еволюції його в загальному вигляді можна .представити як слідуючий цикл:

1. Беремо найостанніший шар моделей.
2. Генеруємо з них по певних правилах новий шар моделей (які тепер самі стають останнім шаром).
3. Відбираємо з них F кращих, де F - ширина відбору (селекції).
4. Якщо не виконується умова припинення селекції (настання звироднілості), переходити на п. 1.
5. Сама краща модель оголошується шуканим рішенням задачі ідентифікації.

Як ми бачимо, в наявності всі ознаки еволюційного алгоритму - відбір (селекція) і генерація нового покоління. Розглянемо метод групового обліку аргументів більш докладніше.

Цей метод розробляється академіком НАН України А.Г. Івахненко і його школою і є типовим методом індуктивного моделювання і одним з найбільш ефективних методів структурно-параметричної ідентифікації складних об'єктів, процесів і систем за даними спостережень в умовах неповноти інформації. В цілому задача полягає в генеруванні за даними спостережень деякої множини F моделей різної структури виду

$$\tilde{y}_f = f(X, \tilde{\Theta}_f)$$

і відшуканні оптимальної моделі по умові мінімізації деякого критерію якості моделей

$$f^* = \arg\min C(y, f(X, \tilde{\Theta}_f)),$$

причому оцінки параметрів для кожної моделі є рішенням ще однієї екстремальної задачі виду

$$\tilde{\Theta}_f = \arg\min Q(y, X, \Theta_f),$$

де s_j називається складністю моделі f і дорівнює числу невідомих параметрів в моделі f^*, а Q - критерій якості рішення задачі параметричної ідентифікації кожної приватної моделі, що генерується в задачі структурної ідентифікації.

МГОА володіє певною різноманітністю можливостей на всіх етапах процесу моделювання в порівнянні з іншими методами побудови моделей. Це стосується перш за все генераторів моделей і вживаних критеріїв якості структур, а також класів моделей (базисних функцій). Метод відрізняється активним застосуванням принципів автоматичної генерації варіантів, послідовній селекції моделей і зовнішніх критеріїв для побудови моделей оптимальної складності. Він має оригінальну процедуру багаторядності автоматичної генерації структур моделей, що імітує процес біологічної селекції з попарним обліком послідовних ознак. Така процедура в сучасній термінології

називається поліноміальною нейронною мережею, причому її структура є явною і будується автоматично, в режимі самоорганізації.

Для порівняння і вибору кращих моделей застосовуються зовнішні критерії, засновані на розділенні вибірки на дві і більш частин, причому оцінювання параметрів і перевірка якості моделей виконується на різних підвибірках. Це дозволяє обійтися без обтяжливих апріорних припущень, оскільки розділення вибірки дозволяє неявно (автоматично) врахувати різні види апріорної невизначеності при побудові моделі. МГОА володіє перевагою при малих вибірках даних за рахунок вибору складності моделі, що оптимально враховує інформативність даних.

Ефективність методу багато разів підтверджувалася вирішенням множини конкретних задач з областей екології, економіки, гідрометеорології і так далі. Зокрема, на основі аналогії між задачею побудови моделі по зашумленим експериментальним даним і задачею проходження сигналу через канал з шумом побудовані начала теорія помехостійкого моделювання. Основний результат цієї теорії полягає в тому, що складність оптимальної прогнозуючої моделі залежить від рівня невизначеності в даних: чим він вищий - тим простіше (грубіше) має бути оптимальна модель (тим менше оцінюваних параметрів).

МГОА добре відомий і вельми активно розвивається у нас в країні і за кордоном. Розроблені основи теорії структурної ідентифікації моделей з мінімальною дисперсією помилки прогнозування. Ефективним апаратом цієї теорії є метод критичних дисперсій, що дозволяє вперше аналітично вирішувати актуальні задачі: порівняльний аналіз критеріїв структурної ідентифікації, планування експериментів, аналіз властивостей методів і тому подібне, причому як при обмеженій вибірці, так і в асимптотиці. При цьому досліджуються умови вибору оптимальної структури моделі залежно від дисперсії (рівня) шуму, довжини вибірки, вхідних дій (плану експерименту) і параметрів об'єкту, причому встановлений тісний взаємозв'язок між ними. Засобами цієї теорії встановлено, що МГОА є методом побудови моделей з

мінімальною дисперсією помилки прогнозування, і виконано його порівняння з іншими методами. Із цього виходить, що МГОА як основний інструмент теорії індуктивного моделювання відноситься до найбільш сучасних методів інтелектуального аналізу даних і м'яких обчислень. Цей метод є оригінальним і ефективним засобом вирішення широкого спектру задач штучного інтелекту, зокрема ідентифікації і прогнозування, розпізнавання образів і кластеризації, інтелектуального аналізу даних і пошуку закономірностей.

У останнє десятиліття інтерес до МГОА активно зростає у всьому світі, що можна пояснити, окрім відомої ефективності методу, також зростанням популярності технології штучних нейромереж. Річ у тому, що структуру МГОА можна інтерпретувати як нейромережу, оригінальність якої полягає в самоорганізації як її структури, так і параметрів. При цьому виявляється, що до явних переваг методу відносяться автоматичне формування структури мережі, простота і швидкодія настроювання параметрів, а також можливість «згорнути» настроєну мережу безпосередньо в явний математичний вираз [2, 9, 39].

Підхід індуктивного моделювання, побудований на принципах самоорганізації, розвивається протягом 40 років, застосовується в багатьох областях і присутній в таких поширених технологіях аналізу даних, як поліноміальні нейроні мережі, адаптивні і статистичні мережі, що навчаються. У нових розробках для побудови моделей на основі даних використовуються також еволюційні і генетичні алгоритми, ідея активних нейронів і багаторівнева самоорганізація, та інші ідеї. Комп’ютерний варіант системи МГОА реалізован в програмному комплексі NeuroShell компанії Ward Systems Group.

Спроби організувати штучну еволюцію в обчислювальному середовищі, ґрунтуючись на основних принципах біологічної еволюції - принципах мінливості і відбору, робилися ще в 70-х роках минулого століття. Досліджувалася еволюція машин Тьюрінга, клітинних автоматів, був розроблений генетичний алгоритм, але відсутність достатніх обчислювальних потужностей заважала поширенню еволюційних алгоритмів. Адже еволюція в природі «обраховується» на гігантській паралельній машині, що складається зі

всіх живих істот. Кожен новонароджений організм - це нове рішення для певного завдання.

Із зростанням продуктивності комп'ютерів еволюційні алгоритми набули широкої популярності. Спочатку параметри алгоритмів підстроювалися вручну. Відома історія про деякого аспіранта на ім'я Олівер, який задавав параметри і запускав еволюційний алгоритм оптимізації вагів для роботи штучної нейронної мережі, а сам йшов в паб. Потім з'явилися методи самоорганізації параметрів еволюційних алгоритмів. Тепер Олівер запускав алгоритм для пошуку оптимальних параметрів еволюційного алгоритму оптимізації вагів для роботи штучної нейронної мережі і йшов в паб. Олівер, напевно, незабаром спився б при такій «науковій роботі», але допомогли колеги, що довели так звану No Free Lunch Theorem. Теорема стверджує, що для будь-якого стохастичного алгоритму пошуку, до яких відносяться і еволюційні алгоритми, середня ефективність дорівнює простому випадковому пошуку. Іншими словами, якщо нам необхідно вирішувати задачі, які не схожі одна на одну, то ми можемо використовувати еволюційний алгоритм з тим же успіхом, що тикати пальцем в небо. Але ж нам не обов'язково потрібний алгоритм, щоб вирішувати будь-які задачі, зазвичай ми маємо справу із задачами, що відносяться до класу, який в теорії еволюційних обчислень отримав назву «Real World Problems» (реальні задачі). Деякі дослідники вважають, що для реальних задач універсальний алгоритм існувати може.

Останніми роками відмінним полігоном для еволюційних технологій став Інтернет. Так, в сумісному проекті Колумбійського університету і університету штату Айова розроблена мультиагентна система InfoSpiders призначена для пошуку інформації в Інтернеті. Інфопаучки сканують мережу у пошуках ресурсів, що задовольняють запиту користувача. Тим, хто знайде «правильні» ресурси, видається енергія. Чим більше у павука енергії, тим вище вірогідність того, що він виживе і залишить потомство - так популяція інфопаучків еволюціонує.

Ще одна сфера застосування еволюційних алгоритмів в Інтернеті - оптимізація мережевих екранів (firewall). Ідея застосувати еволюційні методи для аналізу мережевого трафіку виникла у фахівців компанії MASA Group , що займаються розробкою комп'ютерних адаптивних систем. На той час компанією була створена бібліотека алгоритмів по оптимізації, машинному навчанню, а також по визначенню змін в складних образах (novelty detection). Фахівці MASA Group вирішили, що при захисті мережі основний упор треба робити не на порівняння мережевої активності з шаблонами відомих атак, як в звичайних екранах, а на визначенні відхилень від норми в роботі мережі. Адже якщо вторгнення проводиться новим способом, невідомим системі, то звичайний підхід не зможе визначити атаку. В результаті співпраці MASA Group і MATRAnet ці ідеї втілилися в програмний продукт M>Detect, який, по завіренню представників компаній, повинен з'явитися на ринку восени цього року. Робота M>Detect складається з декількох етапів. На першому етапі система за допомогою еволюційного алгоритму навчається «нормальному» режиму роботи мережі. До кінця навчання формується набір фільтрів, що описують дозволені потоки даних. Тепер M>Detect може переходити до стежачого режиму, в якому використовується система розпізнавання новизни в образах (на основі тимчасових вікон різної тривалості, що дозволяє детектувати як «швидкі», так і «повільні» атаки). Як тільки в мережевому трафіку виникають аномалії, програма посилає системному адміністраторові попередження і лог активності. Все це дозволяє M>Detect виявляти мережеві атаки на ранній стадії, а також реагувати на невідомі типи атак. Якщо попередження було помилковим, то оператор перемикає систему в режим «дообучання», що дозволяє уникнути повторного помилкового спрацьовування. По оцінках розробників, помилкових попереджень буде не більше одного-двох в добу, при цьому об'єм трафіку може досягати 30 Гбайт/с, що, поза сумнівом, є добрим результатом.

Використовують еволюційні алгоритми в боротьбі з комп'ютерними вірусами. В цьому випадку антивірусна система схожа на імунну систему

людини, яка постійно генерує антитіла. При виникненні в машині шкідливої активності штучна імунна система «збуджується», що означає зростання числа випадково створених модифікацій шаблонів вірусів, які були відомі системі. Комп'ютер починає «хворіти». Як і в хворому організмі, під «гарячу руку» збудженої імунної системи можуть потрапляти не тільки чужорідні елементи, але і те, що необхідне для нормальної роботи самого «хворого».

Одна з областей еволюційного комп’ютинга, що найактивніше розвивається, - еволюційна робототехніка. Створення еволотів (еволюційних роботів) зазвичай засноване на еволюції комп'ютерної моделі. Спочатку прототип майбутнього евобота вчиться пересуватися. Якщо це крокуючий робот, то спочатку його ноги заплітають, він часто падає і не може встати, але вже декілька поколінь через окремі віртуальні прототипи майбутнього евобота жваво бігають. Потім наступає етап формування системи управління поведінкою. Залежно від цілей конструктора в процесі штучної еволюції відбираються нейроні мережі, що забезпечують лавірування між перешкодами або, наприклад, збирання об'єктів. Після того, як прототип успішно витримав всі випробування, програма «заливається» в «залізо» реального робота.

Інше цікаве застосування технологія побудови еволотів знайшла в створенні комп'ютерних ігор. Сучасні комп'ютерні ігри насичені масою персонажів, які взаємодіють з гравцем і між собою. Як зробити їх поведінку непередбачуваною, індивідуальною і навіть осмисленою? Надайте можливість попрацювати природному відбору на віртуальному ігровому світі, і ось вона - нова реальність, гра стала набагато багатша.

Сьогодні наші технології управляють навколишнім світом прямо. Створюючи всілякі пристрої і програми, людина обмежує міри свободи середовища. Але вже з'являються системи, в яких управління перенесене на наступний рівень - рівень управління процесами самоорганізації. Це і роботи, що самоконфігуруються, мікророботи, що самозбираються, і, звичайно ж, еволюційні алгоритми.

Бурхливий розвиток еволюційних алгоритмів привів до появи великої кількості програмних комплексів, що мають в своєму складі компоненту еволюційного пошуку. Аналіз таких програмних засобів, показує, що майже всі комерційні програмні пакети, що випускаються, є вбудовуваними в інші системи, так що всі деталі перетворення форматів даних і технології кодування, вибору типів еволюційних операторів, оціночних функцій, способу організації обчислень і так далі приховані від користувача. У існуючих гібридних системах інтеграція здійснюється тільки на рівні окремих модулів. Хоча у багатьох випадках це дає позитивний ефект, в цілому можливості закладених в інтегровані системи алгоритмів повною мірою не використовуються [17, 25].

Розглянемо деякі сучасні концепцій програмної реалізації основних технологій теорії еволюційних обчислень. Одним з відомих програмних комплексів, призначених для комп'ютерного використання технологій інтелектуального аналізу даних є система PolyAnalyst. У цій системі значна увага приділена однієї з найбільш молодих і багатообіцяючих технологій Data Mining - еволюційному програмуванню. Основна ідея методу полягає у формуванні гіпотез про залежність цільової змінної від інших змінних у вигляді автоматично синтезуємих спеціальним модулем програм на внутрішній мові програмування. Використання універсальної мови програмування теоретично дозволяє виразити будь-яку залежність, причому вид цієї залежності заздалегідь не відомий [79].

Процес виробництва внутрішніх програм організовується як еволюція в просторі програм, що нагадує генетичні алгоритми. Коли система знаходить перспективну гіпотезу, яка описує досліджувану залежність досить добре по цілому ряду критерій, в роботу включається механізм так званих "узагальнених перетворень" (GT-search). За допомогою цього механізму в «гарну» програму вводяться незначні модифікації, не погіршуючи її якість, і проводиться відбір кращої дочірньої програми. До нової популяції потім знову застосовуються механізми синтезу нових програм, і цей процес рекурсивно повторюється. Таким чином, система створює деяке число генетичних ліній програм, що

конкурують одна з однією по точності, статистичній значущості і простоті виразу залежності.

Спеціальний модуль безперервно перетворює «кращу» на даний момент програму з внутрішнього уявлення в зовнішню мову PolyAnalyst - мову символьних правил (Symbolic Rule Language), зрозумілу людині: математичні формули, умовні конструкції і так далі. Це дозволяє користувачеві зрозуміти суть отриманої залежності, контролювати процес пошуку, а також отримувати графічну візуалізацію результатів. Контроль статистичної значущості отриманих результатів здійснюється цілим комплексом ефективних і сучасних статистичних методів, включаючи методи рандомізованого тестування (рис. 5.2).

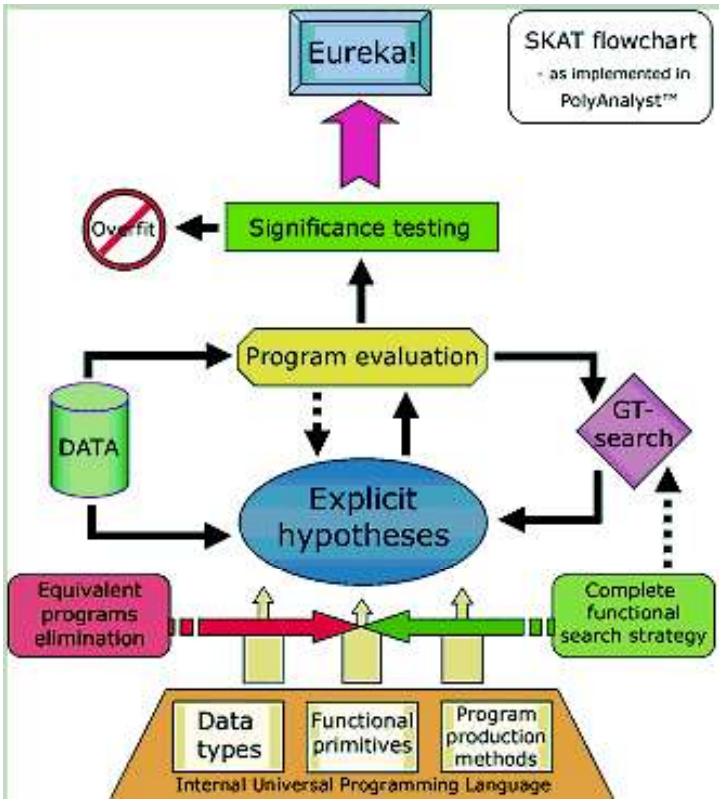


Рис. 5.2. Модуль еволюції програм системи PolyAnalyst.

Іншою сучасною концепцією є еволюційне проектування баз даних. У останні декілька років спостерігається розвиток нового підходу до розробки

програмного забезпечення, а саме застосування так званих гнучких методологій. Такий підхід диктує нові вимоги до проектування баз даних. Одна з центральних вимог пов'язана з ідеєю еволюційного проектування. У гнучкому проекті приймається той факт, що не можна заздалегідь виявити і зафіксувати вимоги до системи. В результаті цього фаза детального проектування на початковій стадії стає непродуктивною. Альтернативний варіант - розробка системи за допомогою багатьох ітерацій. Гнучкі методології, зокрема, екстремальне програмування, пропонують ряд методів, що роблять еволюційне проектування здійсненним.

Одна з ключових особливостей гнучких методик – це ставлення до змін. Більшість міркувань щодо процесу розробки програмного забезпечення зводяться до наступного: на ранньому етапі виявити вимоги, узгодити їх, спроектувати на їх основі систему, узгодити проект і потім приступити до кодування. Такий, керований планом, життєвий цикл часто називають моделлю водопаду. Подібна модель має на увазі мінімізацію змін за рахунок виконання величезної попередньої роботи. Проте після закінчення цієї роботи вартість змін різко зростає. Як наслідок, значні труднощі з'являються при подальшій зміні вимог, а текучість вимог створює вже майже нерозв'язні проблеми.

Гнучкі процеси відносяться до змін інакше. Вони прагнуть управляти змінами, допускаючи їх навіть на пізніх стадіях проектування. Зміни тримаються під контролем, але принципова позиція - максимально спростити можливість змін. Це частково є відповіддю на принципову нестабільність вимог в багатьох проектах, а також забезпечує кращу підтримку бізнес-середовища, що динамічно змінюється. Щоб все це працювало, необхідно змінити відношення до проектування. Замість того щоб розуміти проектування як фазу, що в основному завершується на початок написання коду, слід розглядати проектування як безперервний процес, який перемежається з кодуванням, тестуванням і навіть постачанням замовникові. У цьому проявляється зіставлення планового і еволюційного підходів до проектування. Замість звичного хаосу, який часто виникає за відсутності попереднього планування,

гнучкі методології дозволяють зробити еволюційне проектування здійсненним і керованим.

Важлива частина цього підходу - ітераційна розробка, при якій багато разів повторюється весь життєвий цикл програмного продукту в рамках одного проекту. Гнучкі процеси реалізують повний життєвий цикл в кожній ітерації, завершуючи кожну з них працюючим кодом, що відтестован, інтегрованим, для невеликої підмножини вимог кінцевого продукту. Ітерації короткі: від тижня до пари місяців, причому коротші переважні. Впродовж останніх декількох років був реалізований в крупний проект Atlas, в якому було використано еволюційне проектування баз даних. У проекті брало участь майже 1000 чоловік на декількох майданчиках по всьому світу (у США, Австралії, Індії). Об'єм розробки - приблизно півмільйона рядків коду і більше 2000 таблиць. База даних розроблялася впродовж півтора року, і в даний час вона експлуатується декількома замовниками і продовжує розвиватися.

Ще одним сучасним напрямом є технологія еволюційної побудови нейромереж. Як вже відомо, при побудові нейронної мережі виникає необхідність експериментувати з великою кількістю мереж, порівнюя отримані результати. Для кожної нової досліджуваної мережі визначається її придатність для рішення задачі, при необхідності воно змінюється або відбраковується. Простір всіляких нейронних мереж, в середині якого виконується пошук оптимальної мережі, дуже великий. Тому доводиться застосовувати евристичні методи. Процес – багатократне повторення циклу «вибір нової мережі – оцінка – відбраківка або модифікація» вельми нагадує природний процес еволюції, якщо в кожен момент часу спостерігати тільки за однією особою. Еволюційні методи добре зарекомендували себе в задах пошуку оптимальних рішень в складному просторі. Тому перехід до застосування еволюційних методів для пошуку хороших структур нейронної мережі є закономірним.

При застосуванні еволюційних алгоритмів для побудови нейронних мереж (Evolutionary design of neural architectures - EDNA) спочатку необхідно визначити фенотип і генотип нейронної мережі. Під фенотипом зазвичай

розуміють тільки структуру нейронної мережі (структуру зв'язків між нейронами). Такі параметри як вид та функція активації задаються в генотипі. Вагові коефіцієнти зв'язків для побудови мережі найчастіше задаються випадковим чином, після чого коректуються в процесі навчання нейронної мережі по одному з відомих методів. Проте багато дослідників використовують еволюційні методи і для корректировки вагів зв'язків (рис. 5.3).

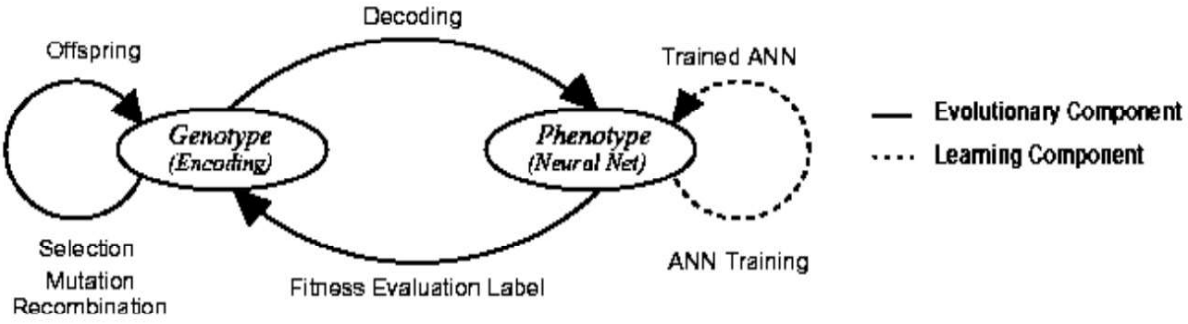


Рис. 5.3. Взаємодія між нейронною мережею та її записом в гені.

На рисунку 5.3 схематично зображена взаємодія між основними об'єктами, які беруть участь у еволюційному процесі. Таких об'єктів всього два - генотипи та фенотипи. Над генотипами проводяться всі еволюційні операції. По генотипу відновлюється фенотип, який проходить процес навчання. По результатах тестування фенотипу розраховується придатність відповідного генома.

5.2. Основні положення теорії генетичних алгоритмів.

Еволюційна теорія стверджує, що кожен біологічний вид цілеспрямовано розвивається і змінюється для того, щоб якнайкраще пристосуватися до навколишнього середовища. В процесі еволюції багато видів комах і риб придбали захисне забарвлення, їжак став невразливим завдяки голкам, людина стала володарем складної нервової системи. Можна сказати, що

еволюція - це процес оптимізації всіх живих організмів. Розглянемо, якими ж засобами природа вирішує цю задачу оптимізації [7, 12,18, 31].

Основний механізм еволюції - це природний відбір. Його суть полягає в тому, що більш пристосовані особини мають більше можливостей для виживання і розмноження і, отже, приносять більше потомства, чим погано пристосовані особини. При цьому завдяки передачі генетичній інформації (генетичному спадкоємству) нащадки успадковують від батьків основні їх якості. Таким чином, нащадки сильних індивідуумів також будуть відносно добре пристосованими, а їх частка в загальній масі особин зростатиме. Після зміни декількох десятків або сотень поколінь середня пристосованість особин даного вигляду помітно зростає.

Щоб зробити зрозумілими принципи роботи генетичних алгоритмів, пояснимо також, як влаштовані механізми генетичного спадкоємства в природі. У кожній клітці будь-якої тварини міститься вся генетична інформація цієї особини. Ця інформація записана у вигляді набору дуже довгих молекул ДНК (Дезоксирибонуклеїнова Кислота). Кожна молекула ДНК - це ланцюжок, що складається з молекул нуклеотидів чотирьох типів, що позначаються A, T, C і G. Власне, інформацію несе порядок проходження нуклеотидів в ДНК. Таким чином, генетичний код індивідуума - це просто дуже довгий рядок символів, де використовуються всього 4 букви. У тваринній клітці кожна молекула ДНК оточена оболонкою – таке утворення називається **хромосомою**.

Кожна природжена якість особини (колір очей, спадкові хвороби, тип волосся і так далі) кодується певною частиною хромосоми, яка називається **геном** цієї властивості. Наприклад, ген кольору очей містить інформацію, що кодує певний колір очей. Різні значення гена називаються його **алелями**.

При розмноженні тварин відбувається злиття двох батьківських статевих кліток і їх ДНК взаємодіють, утворюючи ДНК нащадка. Основний спосіб взаємодії - **кросовер** (cross-over, схрещування). При кросовері ДНК предків діляться на дві частини, а потім обмінюються своїми половинками.

При спадкоємстві можливі мутації із-за радіоактивності або інших впливів, в результаті яких можуть змінитися деякі гени в статевих клітинах одного з батьків. Змінені гени передаються нащадкові і наділяють його новими властивостями. Якщо ці нові властивості корисні, вони, швидше за все, збережуться в даному виді - при цьому відбудеться стрибкоподібне підвищення пристосованості виду.

Як вже було відмічено, еволюція - це процес постійної оптимізації біологічних видів. Знаючи, як вирішується задача оптимізації видів в природі, застосуємо схожий метод для вирішення різних реальних задач.

Хай дана деяка складна функція (цільова функція), залежна від декількох змінних, і потрібно знайти такі значення змінних, при яких значення функції максимальне. Один з найбільш наглядних прикладів - задача розподілу інвестицій. У цієї задачі змінними є обсяги інвестицій в кожен проект (наприклад, 10 змінних), а функцією, яку потрібно максимізувати, - сумарний дохід інвестора. Також задані значення мінімального і максимального обсягу вкладення в кожен з проектів, які задають область зміни кожній із змінних. Спробуємо вирішити цю задачу, застосовуючи відомі нам природні способи оптимізації. Розглядатимемо кожен варіант інвестування (набір значень змінних) як індивідуума, а прибутковість цього варіанту - як пристосованість цього індивідуума. Тоді в процесі еволюції пристосованість індивідуумів зростатиме, а значить, з'являтимуться все більш і більш прибуткові варіанти інвестування. Зупинивши еволюцію в деякий момент і вибравши самого кращого індивідуума, ми отримаємо досить хороше рішення задачі.

Генетичний алгоритм (ГА) - це проста модель еволюції в природі, реалізована у вигляді комп'ютерної програми. Він широко використовується, для проектування структури мостів, для пошуку максимального відношення міцність/вага, для визначення найменш марнотратного розміщення при нарізці форм з тканини. ГА можуть також використовуватися для інтерактивного управління процесом, наприклад на хімічному заводі, або балансуванні завантаження на багатопроцесорному комп'ютері. Наприклад, ізраїльська

компанія Schema розробила програмний продукт Channeling для оптимізації роботи стільникового зв'язку шляхом вибору оптимальної частоти, на якій вестиметься розмова. У основі цього програмного продукту використовуються генетичні алгоритми.

У генетичному алгоритмі використовується як аналог механізму генетичного спадкоємства, так і аналог природного відбору. При цьому зберігається біологічна термінологія в спрощеному вигляді. От як моделюється генетичне спадкоємство:

Хромосома	Вектор (послідовність) з нулів і одиниць. Кожна позиція (біт) називається геном.
Індивідуум = генетичний код	Набор хромосом = варіант рішення задачі.
Кросовер	Операція, при якій дві хромосоми обмінюються своїми частинами.
Мутація	Випадкова зміна однієї або декількох позицій в хромосомі.

Щоб змоделювати еволюційний процес, згенеруємо спочатку випадкову популяцію - декілька індивідуумів з випадковим набором хромосом (числових векторів). Генетичний алгоритм імітує еволюцію цієї популяції як циклічний процес схрещування індивідуумів і зміни поколінь (рис. 5.4).

Життєвий цикл популяції - це декілька випадкових схрещувань (за допомогою кросовера) і мутацій, в результаті яких до популяції додається якась кількість нових індивідуумів. Відбір в генетичному алгоритмі - це процес формування нової популяції із старої, після чого стара популяція гине. Після відбору до нової популяції знову застосовуються операції кросовера і мутації, потім знову відбувається відбір, і так далі.

Рис. 5.4. Алгоритм генетичних обчислювань.

Відбір в генетичному алгоритмі тісно пов'язаний з принципами природного відбору в природі таким чином:

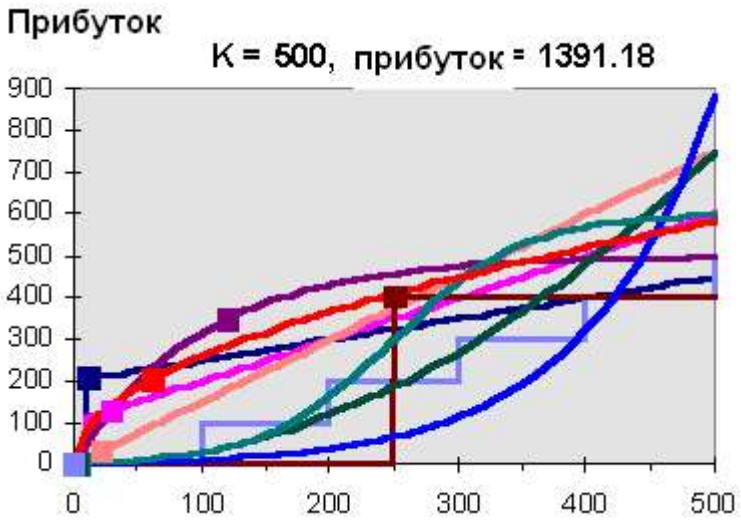
Пристосованість індивідуума	Значення цільової функції на цьому індивідуумі
Виживання найбільш пристосованих	Популяція наступного покоління формується відповідно до цільової функції. Чим пристосованіше індивідуум, тим більше вірогідність його участі в кросовері, тобто розмноженні.

Таким чином, модель відбору визначає, яким чином слід будувати популяцію наступного покоління. Як правило, вірогідність участі індивідуума в схрещуванні береться пропорційній його пристосованості. Часто використовується так звана стратегія елітизма, при якій декілька кращих індивідуумів переходять в наступне покоління без змін, не беручи участь в кросовері і відборі. У будь-якому випадку кожне наступне покоління буде в середньому краще попереднього. Коли пристосованість індивідуумів перестає помітно збільшуватися, процес зупиняють і як рішення задачі оптимізації беруть якнайкращого із знайдених індивідуумів.

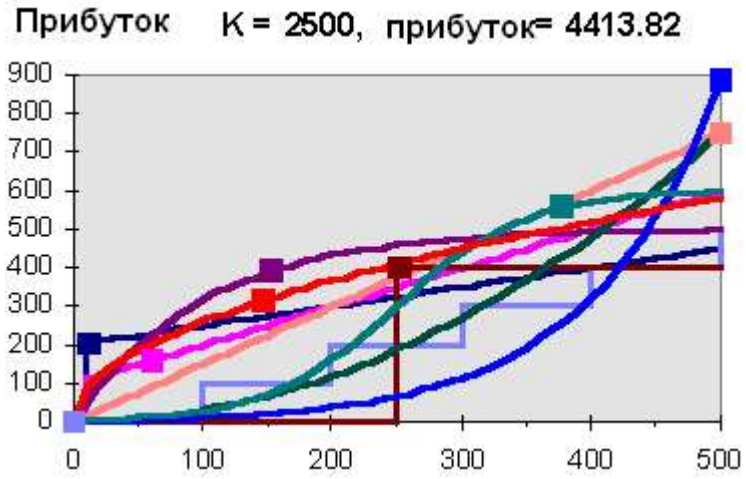
Повертаючись до задачі оптимального розподілу інвестицій, пояснимо особливості реалізації генетичного алгоритму в цьому випадку.

- Індивідуум = варіант рішення задачі = набір з 10 хромосом X_j.
- Хромосома X_j = об'єм вкладення в проект j = 16-розрядний запис цього числа
- Оскільки обсяги вкладень обмежені, не всі значення хромосом є допустимими. Це враховується при генерації популяцій.
- Оскільки сумарний обсяг інвестицій фіксований, то реально варіюються лише 9 хромосом, а значення 10-ої визначається по ним однозначно.

Нижче приведені результати роботи генетичного алгоритму для трьох різних значень сумарного обсягу інвестицій K (рис. 5.5). З рисунка 5.5. (а) видно, що при малому значенні K інвестуються тільки ті проекти, які прибуткові при мінімальних вкладеннях. Якщо збільшити сумарний обсяг інвестицій, стає прибутковим вкладати гроші і в дорожчі проекти.



а) капіталовкладення в проект;



б) капіталовкладення в проект.

Рис. 5.5. Розрахунок обсягу інвестицій на основі генетичного алгоритму.

При подальшому збільшенні K досягається поріг максимального вкладення в прибуткові проекти, і інвестування в малоприбуткові проекти знову набуває сенсу (рис. 5.5. б)Природа приголомшує своєю складність і багатством всіх своїх проявів. Серед прикладів можна назвати складні соціально-економічні системи, імунні і нейроні системи, складні взаємозв'язки між видами. Вони - всього лише деякі з чудес, які сталі очевидніші, коли вчені стали глибшими досліджувати світ довкола нас. Багато що з того, що ми бачимо і спостерігаємо, можна пояснити єдиною теорією: теорією еволюції через спадковість, мінливість і відбір.

Перша публікація, яку можна віднести до генетичних алгоритмів з'явилася в 1957 році, автором якої був Н.А. Барічеллі (N.A. Barricelli), і називалася вона «Symbiogenetic Evolution Processes realised by artificial methods». У 1962 році з'явилася ще одна його робота «Numerical testing of evolution theories». Приблизно в той же час ще один дослідник А.С.Фрейзер (A.S.Fraser) також опублікував дві статті: «Simulation of genetic systems by automatic digital computers: S-linkage, dominance, and epistasis» (1960) і «Simulation of genetic systems» (1962). Не дивлячись на те, що роботи обох авторів були направлені перш за все на розуміння природного феномену

спадковості, робота Фрейзера має багато спільного з сьогоднішнім баченням генетичних алгоритмів. Він моделював еволюцію 15-бітових рядків і підраховував процентний зміст особин з вдалим фенотипом в успішних поколіннях. Його роботи нагадують оптимізацію функцій, проте в роботах Фрейзера немає жодної згадки про можливість використовувати ГА для штучних задач.

У багатьох джерелах саме Холланда називають батьком сучасної теорії генетичних алгоритмів. Проте, Холланд займався ними не із самого початку. Його цікавила, перш за все, здібність природних систем до адаптації, а його мрією було створення такої системи, яка могла б пристосуватися до будь-яких умов довкілля. Заслуга Холланда в тому, що він усвідомив значення еволюційних принципів в адаптації і розвинув свої припущення. Разом зі своїми студентами, що слухали курси по адаптивних системах в університеті штату Мічіган, він розробляє те, що згодом назве «генетичним планом», а нам це відомо як «генетичний алгоритм». Про те, наскільки серйозно велися роботи в цьому напрямі говорить вже те, що один із студентів Холланда - Кенет Де Йонг (Kenneth De Jong) захистив дисертацію «An Analysis of the Behavior of a Class of Genetic Adaptive Systems» на ступінь доктора філософії в 1975 році. У тому ж році виходить знаменита книга Холланда «Адаптація в природних і штучних системах». Після цього ГА починають привертати до себе все більше уваги, ними займається все більше дослідників, які знаходять їм нові сфери вживання.

В даний час генетичні алгоритми в різних формах застосовуються до багатьох наукових і технічних проблем, зокрема для інтелектуального аналізу даних. Слід зазначити, що Data Mining не основна сфера застосування генетичних алгоритмів. Їх потрібно розглядати швидше як потужний засіб вирішення всіляких комбінаторних задач і задач оптимізації. Проте генетичні алгоритми увійшли зараз до стандартного інструментарію методів Data Mining.

Можливо найбільш популярним додатком генетичних алгоритмів є оптимізація багатопараметричних функцій. Багато реальних задач можуть бути

сформульовані як пошук оптимального значення, де значення - складна функція, залежна від деяких вхідних параметрів. В деяких випадках, представляє інтерес знайти ті значення параметрів, при яких досягається найкраще точне значення функції. У інших випадках, точний оптимум не потрібний - рішенням може вважатися будь-яке значення, яке краще за деяку задану величину. В цьому випадку, генетичні алгоритми - часто найбільш прийнятний метод для пошуку «хороших» значень. Сила генетичного алгоритму поміщена в його здатності маніпулювати одночасно багатьма параметрами, ця особливість ГА використовувалося в сотнях прикладних програм, включаючи проектування літаків, налаштування параметрів алгоритмів і пошуку стійких станів систем нелінійних диференціальних рівнянь. Таким чином, перевагами генетичних алгоритмів є:

- не вимагання жодної інформації про поверхню відповіді;
- розриви, що існують на поверхні відповіді мають незначний ефект та не впливають на ефективність оптимізації;
- стійкість до попадання в локальний оптимум;
- добре працюють при вирішенні великомасштабних проблем оптимізації;
- Можуть бути використані для широкого класу задач, зокрема; з середовищем, що змінюється.

Недоліками генетичних алгоритмів слід вважати:

- складність роботи у разі, коли необхідно знайти точний глобальний оптимум;
- час виконання функції оцінки великий;
- конфігурація є не простою (кодування рішення).

Мета інтелектуального аналізу даних за допомогою ГА полягає в тому, аби знайти краще можливе рішення задачі по одному або декільком критеріям. Аби реалізувати генетичний алгоритм, потрібно спочатку вибрати відповідну структуру для представлення цих рішень. У постановці задачі пошуку об'єкт цієї структури даних представляється точкою в просторі пошуку всіх можливих рішень. Властивості об'єктів представлені значеннями параметрів, що

об'єднуються в запис, названий в еволюційних методах **хромосомою**. У генетичних методах оперують хромосомами, що відносяться до множини об'єктів - **популяції**. Імітація генетичних принципів - **імовірнісний вибір** батьків, серед членів популяції, схрещування їх хромосом, відбір нащадків для включення в нові покоління об'єктів на основі оцінки цільової функції - веде до еволюційного поліпшення значень **цільової функції** (функції пристосованості) від покоління до покоління.

Найчастіше хромосома - це бітовий рядок. Проте ГА не обмежені бінарним представленням даних. Деякі реалізації використовують цілочисельне або вещественне кодування. Не дивлячись на те, що для багатьох реальних задач, мабуть, більше личать рядки змінної довжини, в даний час структури фіксованої довжини найбільш поширені і вивчені. Спочатку і ми обмежимося лише структурами, які є одиночними рядками по n біт.

Кожна **хромосома** (рядок) є конкатенацією ряду підкомпонентів, названих генами. Гени розташовуються в різних позиціях або локусах хромосоми і набувають значень, званих алелями. У представленнях з бінарними рядками **ген** - біт, **локус** - його позиція в рядку і алель - його значення (0 або 1). Біологічний термін «**генотип**» відноситься до повної генетичної моделі особини і відповідає структурі в ГА. Термін «**фенотип**» відноситься до зовнішніх спостережуваних ознак і відповідає вектору в просторі параметрів. Надзвичайно простий, але ілюстративний приклад - задача максимізації наступної функції двох змінних

$$f(x_1, x_2) = x_1 x_2,\ 0 \le x_1 \le 1,\ 0 \le x_2 \le 1.$$

Зазвичай методика кодування реальних змінних x_1 і x_2 полягає в їх перетворенні в двійкові цілочисельні рядки достатньої довжини - достатньою для того, щоб забезпечити бажану точність. Передбачимо, що 10-розрядне кодування достатнє і для x_1, і x_2. Встановити відповідність між генотипом і фенотипом закодованих особин можна розділивши відповідне двійковому представленню ціле число на значення $2^{10} - 1$. Наприклад, код [0000000000] відповідає вещественному значенню 0/1023 або 0, тоді як код [1111111111]

відповідає 1023/1023 або 1. Структура даних, що оптимізується - 20-бітовий рядок, що представляє конкатенацію кодувань x_1 і x_2. Змінна x_1 розміщується в крайніх лівих 10-розрядах, тоді як x_2 розміщується в правій частині генотипу особини (20-бітовому рядку). Генотип - точка в 20-вимірному бінарному просторі, досліджуваному ГА. Фенотип - точка в двовимірному просторі параметрів.

Після того, як вибрані параметри, їх число і розрядність, необхідно вирішити, як безпосередньо записувати дані. Можна використовувати звичайне кодування, коли, наприклад, $<1011>_2=<11>_{10}$, або **коди Грея**, коли $<1011>_G=<1110>_2=<14>_{10}$. Не дивлячись на те, що коди Грея потребують неминучого кодування/декодування даних, вони дозволяють уникнути деяких проблем, які з'являються в результаті звичайного кодування. Можна лише сказати, що перевага кода Грея в тому, що якщо два числа є послідовними при кодуванні, то і їх двійкові коди розрізняються лише на один розряд, а в двійкових кодах це не так. Варто відзначити, що кодувати і декодувати в коди Грея можна так:

- спочатку копіюється самий старший розряд, потім:
- з двійкового кода в код Грея $G[i] = XOR(B[i+1], B[i])$;
- з кода Грея в двійковий код $B[i] = XOR(B[i+1], G[i])$.

Тут, $G[i]$ - і-й розряд кода Грея, а $B[i]$ - і-й розряд бінарного кода. Наприклад, послідовність чисел від 0 до 7 в двійковому коді {000, 001, 010, 011, 100, 101, 110, 111}, а в кодах Грея {000, 001, 011, 010, 110, 111, 101, 100}.

Загальна схема такого алгоритму може бути записана таким чином.

1. Формування початкової популяції.
2. Оцінка особин популяції.
3. Відбір (селекція).
4. Схрещування.
5. Мутація.
6. Формування нової популяції.

7. Якщо популяція не зійшлася, то 2. Інакше - останов.

Зупинимося детальніше на всіх етапах цього алгоритму.

Формування початкової популяції. Стандартний генетичний алгоритм починає свою роботу з формування початкової популяції I_0 - кінцевого набору допустимих рішень задачі. Ці рішення можуть бути вибрані випадковим чином або отримані за допомогою простих наближених алгоритмів. Вибір початкової популяції не має значення для збіжності процесу в асимптотиці, проте формування «хорошої» початкової популяції (наприклад, з множини локального оптимуму) може помітно скоротити час досягнення глобального оптимуму. Якщо відсутні припущення про місце розташування глобального оптимуму, то індивіди з початкової популяції бажано розподілити рівномірно по всьому простору пошуку рішення.

Оцінка особин популяції. Аби оптимізувати яку-небудь структуру з використанням ГА, потрібно задати міру якості для кожного індивіда в просторі пошуку. Для цієї мети використовується **функція пристосованості**. У задачах максимізації цільова функція часто сама виступає як функція пристосованості (наприклад, як в розглянутому раніше двовимірному прикладі); для задач мінімізації цільову функцію слід інвертувати. Якщо вирішувана задача має обмеження, виконання яких неможливо контролювати алгоритмічно, то функція пристосованості, як правило, включає також штрафи за невиконання обмежень (вони зменшують її значення).

Відбір (селекція). Існує декілька підходів до вибору батьківської пари. Найбільш поширеними операторами вибору батьків є наступні.

Панміксія - найпростіший оператор відбору. Відповідно до його кожному членові популяції зіставляється випадкове ціле число на відрізку [1; n], де n - кількість особин в популяції. Розглядатимемо ці числа як номери особин, які візьмуть участь в схрещуванні. При такому виборі якісь з членів популяції не братимуть участь в процесі розмноження, оскільки утворюють пару самі з собою. Якісь члени популяції візьмуть участь в процесі відтворення не однократно з різними особинами популяції. Не дивлячись на простоту, такий

похід універсальний для вирішення різних класів задач. Проте він досить критичний до чисельності популяції, оскільки ефективність алгоритму, що реалізовує такий підхід, знижується із зростанням чисельності популяції.

Інбридинг є таким методом, коли перший з батьків вибирається випадковим чином, а другим з родини є член популяції найближчий до першого. Тут «найближчий» може розумітися, наприклад, в сенсі мінімальної відстані Хеммінга (для бінарних рядків) або евклідова відстані між двома вещественними векторами. Відстань Хеммінга дорівнює числу локусов (розрядів), що розрізняються, в бінарному рядку. При *аутбридинге* також використовують поняття схожості особин. Проте тепер шлюбні пари формують з максимально далеких особин.

Останні два способи по різному впливають на поведінку генетичного алгоритму. Так, інбридинг можна охарактеризувати властивістю концентрації пошуку в локальних вузлах, що фактично приводить до розбиття популяції на окремі локальні групи довкола підозрілих на екстремум ділянок ландшафту. Аутбридинг же направлений на попередження збіжності алгоритму до вже знайдених рішень, заставляючи алгоритм переглядати нові, недосліджені області. Інбридинг і аутбридинг буває генотипом (коли в якості відстані береться різниця значень цільової функції для відповідних особин) і фенотипним (в якості відстані береться відстань Хеммінга).

Селекція полягає в тому, що батьками можуть стати лише ті особини, значення пристосованості яких не менше порогової величини, наприклад, середнього значення пристосованості по популяції. Такий похід забезпечує швидшу збіжність алгоритму. Проте із-за швидкої збіжності селективний вибір батьківської пари не підходить тоді, коли ставиться задача визначення декількох екстремумів, оскільки для таких задач алгоритм, як правило, швидко сходиться до одного з рішень. Крім того, для деяких багатовимірних задач із складним ландшафтом цільової функції швидка збіжність може перетворитися на передчасну збіжність до квазіоптимального рішення. Цей недолік може бути частково компенсований використанням відповідного механізму відбору, який

би «гальмував» дуже швидку збіжність алгоритму. Порогова величина в селекції може бути обчислена різними способами. Тому виділяють різні варіації селекції. Серед операторів селекції найбільш поширеними є два імовірнісні оператори **пропорційної (рулеточної)** і **турнірної** селекції. В деяких випадках також застосовується **відбір усіканням**.

Пропорційний відбір (Proportional selection). При пропорційній селекції вірогідність на k-му кроці є вибір рішення i як одного з батьків, яке задається формулою $P\{i-вибрано\}=\dfrac{f(i)}{\sum\limits_{j\in I_k} f(j)}$, у припущенні, що $f(i)>0$ для всіх $i\in I_k$ (I_k - популяція на k-му кроці).

Простий пропорційний відбір – *рулетка (roulette-wheel selection)* - відбирає особини за допомогою n «запусків» рулетки. Колесо рулетки містить по одному сектору для кожного i-го члена популяції. Розмір i-го сектора пропорційний відповідній величині $P(i)$. При такому відборі члени популяції з вищою пристосованістю з більшою вірогідністю частіше вибиратимуться, чим особини з низькою пристосованістю.

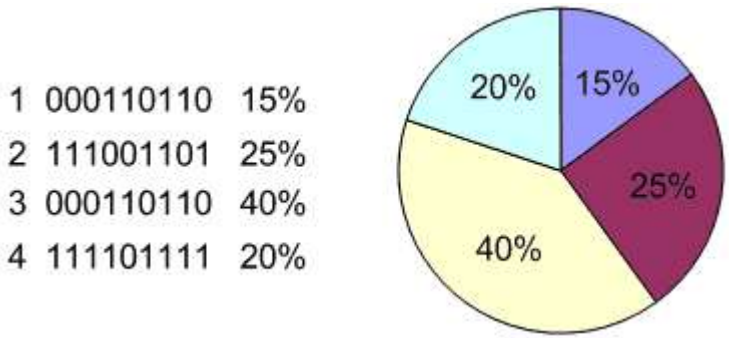


Рис. 5.6. Оператор селекції типу колеса рулетки.

Турнірний відбір (Tournament selection). Турнірний відбір може бути описаний таким чином: з популяції, що містить m рядків (особин), вибирається випадковим чином t рядків і кращий рядок записується в проміжний масив (між вибраними рядками проводиться турнір). Ця операція повторюється m разів. Рядки в отриманому проміжному масиві потім використовуються для схрещування. Розмір групи рядків, що відбираються для турніру, часто рівний

2. В цьому випадку говорять про двоїчним/парний турнір. Взагалі ж t називається чисельністю турніру (рис. 5.7).

Відбір усіканням (Truncation selection). Ця стратегія використовує відсортовану по убуванню популяцію. Число особин для схрещування вибирається відповідно до порогу $T \in [0;1]$. Поріг визначає, яка доля особин, починаючи з найпершої (самої пристосованої), братиме участь у відборі. В принципі, поріг можна задати і рівним 1, тоді всі особини поточної популяції будуть допущені до відбору. Серед особин, допущених до схрещування випадковим чином $m/2$ разів вибираються батьківські пари, нащадки яких утворюють нову популяцію.

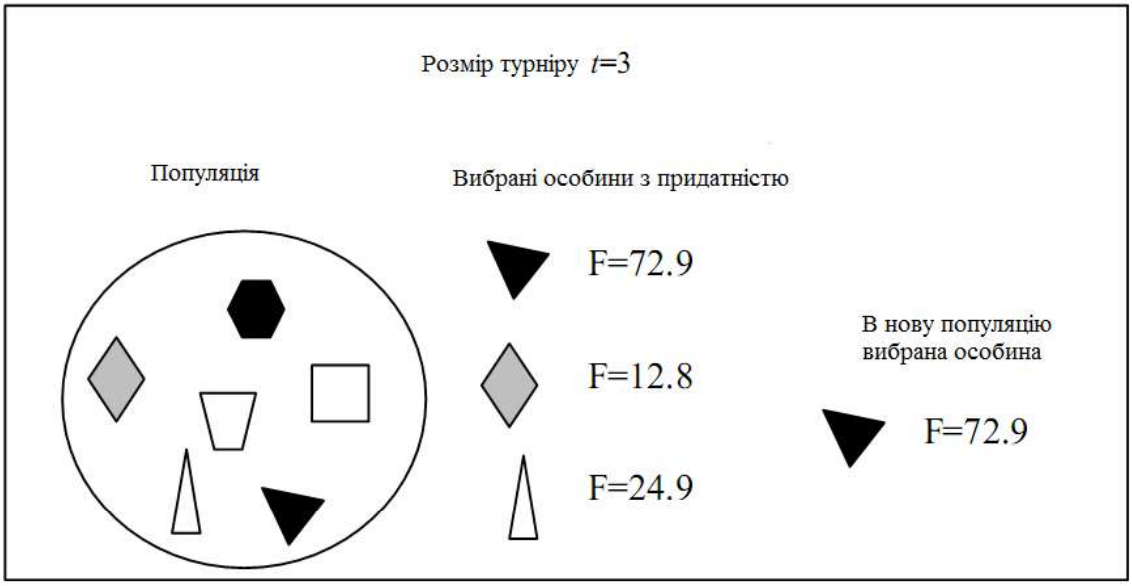


Рис.5.7. Турнірний відбір.

Схрещування. Оператори рекомбінації (схрещування) застосовують відразу ж після оператора відбору батьків для здобуття нових особин-нащадків. Сенс рекомбінації полягає в тому, що створені нащадки повинні успадковувати генну інформацію від обох батьків. Розрізняють **дискретну рекомбінацію** і **кросинговер**.

Дискретна рекомбінація (Discrete recombination) в основному застосовується до хромосом з вещественними генами. Основними способами

дискретної рекомбінації є власне дискретна рекомбінація, проміжна, лінійна і розширена лінійна рекомбінації.

Дискретна рекомбінація відповідає обміну генами між особинами. Для ілюстрації даного оператора порівняємо дві особини з трьома генами

Особина 1	12	25	7
Особина 2	116	4	34

Для створення двох нащадків з рівною імовірністю випадково виберемо номер особини для кожного гена

Схема 1	2	2	1
Схема 2	1	2	1

Метод рулетки. Сумарна придатність = 200, сумарна імовірність = 1.

Популяція з 5 особин	Придатність	Імовірність вибору
C_1	52	52/200 = 0.26
C_2	85	85/200 = 0.425
C_3	37	37/200 = 0.185
C_4	3	3/200 = 0.015
C_5	23	23/200 = 0.115

Згідно схемі створимо нащадків

Нащадок 1	116	4	7
Нащадок 2	12	4	7

Дискретна рекомбінація може застосовуватися для будь-якого типа генів (двоїчних, вещественних і символьних).

Проміжна рекомбінація (Intermediate recombination) застосовується лише до вещественних змінних, але не до бінарних. У даному методі заздалегідь визначається числовий інтервал значень генів нащадків, який повинен містити значення генів батьків. Нащадки створюються за наступним правилом: Нащадок = Батько 1 + α(Батько 2 — Батько 2), де множник α - випадкове число на відрізку $[-d,1+d]$, $d>0$. Як відзначають прибічники цього методу, найбільш оптимальне відтворення виходить при $d=0.25$. Для кожного гена створюваного нащадка вибирається окремий множник α. Розглянемо вживання оператора на прикладі. Хай двоє батьків мають наступні значення генів

Особина 1	12	25	7
Особина 2	116	4	34

Випадково виберемо значення $\alpha \in [-0.25;1.25]$ для кожного гена обох нащадків

Схема 1	0.5	1.1	-0.1
Схема 2	0.1	0.8	0.5

Обчислимо значення генів нащадків за запропонованою вище формулою

Нащадок 1	12+0.5(116–12) = 64	25+1.1(4-25) = 1.9	4.3
Нащадок 2	12+0.1(116-12) = 2.4	25+0.8(4-25) = 8.2	20.5

При проміжній рекомбінації виникають значення генів, відмінні від значення генів особин-батьків. Це приводить до виникнення нових особин, придатність яких може бути краще, ніж придатність батьків. У літературі такий оператор рекомбінації інколи називається *диференціальним схрещуванням*.

Лінійна рекомбінація (Line recombination) відрізняється від проміжної тим, що множник α вибирається для кожного нащадка один раз. Розглянемо гени приведених вище батьків. Хай значення α визначається таким чином

Схема 1	0.5
Схема 2	0.1

Тоді гени створених нащадків набудуть наступних значень

Нащадок 1	12+0.5(116–12) = 64	25+0.5(4-25) = 14.5	20.5
Нащадок 2	12+0.1(116-12) = 2.4	25+0.1(4-25) = 22.9	9.7

Якщо розглядати особині популяції як точки в k-мірному просторі, де k - кількість генів в одній особині, то можна сказати, що при лінійній рекомбінації точки нащадків, що генеруються, лежать на прямій, заданій двома точками - батьками.

Кросинговер (Crossover). Рекомбінацію бінарних рядків прийнято називати кросинговером або схрещуванням. Як тільки два рішення-батька вибрано, до них застосовується імовірнісний оператор схрещування, який будує на їх основі нові (1 або 2) рішення-нащадка. Відібрані особини піддаються кросинговеру із заданою вірогідністю P_c. Якщо кожна пара батьків породжує двох нащадків, для відтворення популяції необхідно схрестити $m/2$ пари. Для кожної пари з вірогідністю P_c застосовується кросинговер. Відповідно, з вірогідністю 1-P_c кросинговер не відбувається і тоді незмінені особини переходять на наступну стадію (мутації). Існує велика кількість різновидів оператора схрещування.

Одноточковий кросинговер (Single-point crossover) працює таким чином. Спочатку випадковим чином вибирається одна з можливих точок розриву. Точка розриву це ділянка між сусідніми бітами в рядку. Обоє батьківські

структури розриваються на два сегменти по цій точці. Потім відповідні сегменти різних батьків склеюються і виходять два генотипи нащадків (рис. 5.8)).

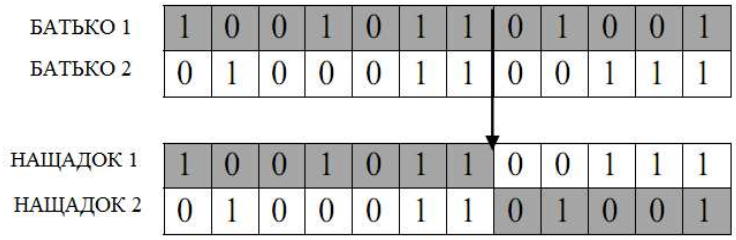
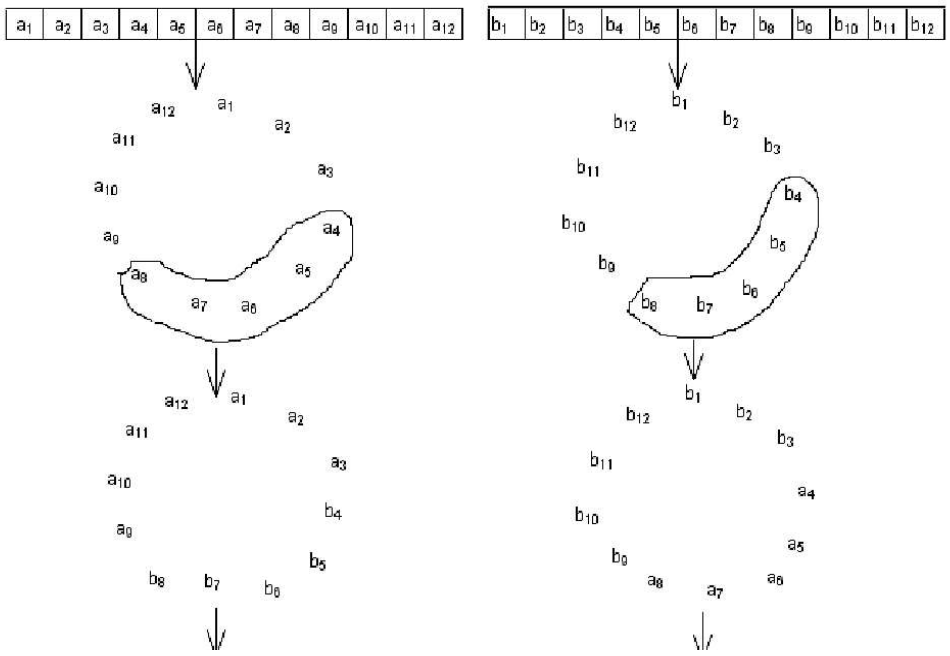


Рис. 5.8. Приклад роботи одноточкового кросинговера.

У двоточковому кросинговері хромосоми розглядаються як цикли, які формуються з'єднанням кінців лінійної хромосоми разом. Для заміни сегменту одного циклу сегментом іншого циклу потрібний вибір двох точок розрізу. В такому представленні, одноточковий кросинговер може бути розглянутий як кросинговер з двома точками, але з однією точкою розрізу, зафіксованою на початку рядка. Отже, двоточковий кросинговер вирішує ту ж саму задачу, що і одноточковий, але більш повно. Хромосома, що розглядається як цикл, може містити більшу кількість стандартних блоків, оскільки вони можуть зробити «циклічне повернення» в кінці рядка (рис. 5.9). Зараз багато дослідників погоджуються, що двоточковий кросинговер взагалі краще, ніж одноточковий.



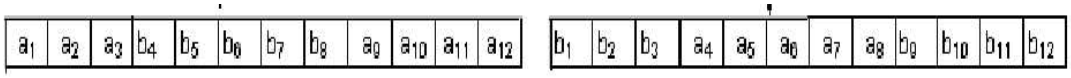


Рис. 5.9. Двоточковий кросинговер.

Для *багатоточкового кросинговера (Multi-point crossover)*, вибираємо m точок розрізу $k_i \in \{1,2,...,N_{\text{var}}\}$, $i=\overline{1,m}$, N_{var} - кількість змінних (генів) в особини. Точки розрізу вибираються випадково без повторень і сортуються в порядку зростання. При кросинговері відбувається обмін ділянками хромосом, обмеженими точками розрізу і таким чином отримують двох нащадків. Ділянка особини з першим геном до першої точки розрізу в обміні не бере участь. Порівняємо наступні дві особини по 11 двійковим генам.

Особина 1	0	1	1	1	0	0	1	1	0	1	0
Особина 2	1	0	1	0	1	1	0	0	1	0	1

Виберемо точки розрізу кросинговера

Точка розрізу ($m=3$)	2	6	10

Створимо двох нових нащадків

Нащадок 1	0	1	1	0	1	1	1	1	0	1	1
Нащадок 2	1	0	1	1	0	0	0	0	1	0	0

Вживання багатоточкового кросинговера вимагає введення декілька змінних (точок розрізу), і для відтворення вибираються особини з найбільшою пристосованістю.

Однорідний кросинговер (Uniform crossover) створює маску (схему) особини, в кожному локусі якою знаходиться потенційна точка кросинговера. Маска кросинговера має таку же довжину, що і особини, які схрещуються.

Створити маску можна таким чином: введемо деяку величину $0 < p_0 < 1$, і якщо випадкове число більше p_0, то на n позицію маски ставимо 0, інакше - 1. Таким чином, гени маски є випадковими двійковими числами (0 або 1). Згідно цим значенням, геном нащадка стає перша (якщо ген маски = 0) або друга (якщо ген маски = 1) особина-батько. Наприклад, розглянемо особині

Особина 1	0	1	1	1	0	0	1	1	0	1	0
Особина 2	1	0	1	0	1	1	0	0	1	0	1

Для кожного створюваного нащадка створимо маску з 11 випадково вибраних елементів з множини {0; 1}

Маска 1	0	1	1	0	0	0	1	1	0	1	0
Маска 2	1	0	0	1	1	1	0	0	1	0	1

Створимо нащадків за наступним правилом: якщо на i-му місці у відповідній масці стоїть 1, то ген 1 батька переходить нащадкові, інакше успадковується ген другого батька. Отримаємо наступні особини

Нащадок 1	1	1	0	1	1	1	1	1	1	1	1
Нащадок 2	0	0	1	1	0	0	0	0	0	0	0

Однорідний кросинговер дуже схожий на багатоточковий, але рядок випадкових бітових значень в ньому довше. Це гарантує, що в нащадках чергуватимуться короткі рядки особин-батьків.

Тріадний кросинговер (Triadic crossover). Даний різновид кросинговера відрізняється від однорідного тим, що після відбору пари батьків з останніх членів популяції випадковим чином вибирається особина, яка надалі використовується як маска. Далі 10 % генів маски мутують. Потім гени першого батька порівнюються з генами маски: якщо гени однакові, то вони

передаються першому нащадкові, інакше на відповідні позиції хромосоми нащадка переходять гени другого батька. Генотип другого нащадка відрізняється від генотипу першого тим, що на тих позиціях, де у першого нащадка стоять гени першого батька, у другого нащадка стоять гени другого батька і навпаки.

Як було показано вище, кросинговер генерує нове рішення (у вигляді особини-нащадка) на основі двох інших, комбінуючи їх частини. Тому число різних рішень, які можуть бути отримані кросинговером при використанні однієї і тієї ж пари готових рішень, обмежене. Відповідно, простір, який ГА може покрити, використовуючи лише кросинговер, жорстко залежить від генофонду популяції. Чим різніше генотипи рішень популяції, тим більше простір покриття. При виявленні локального оптимуму відповідний йому генотип прагнутиме зайняти всі позиції в популяції, і алгоритм може зійтися до помилкового оптимуму. Тому в генетичному алгоритмі важливу роль грають мутації.

Мутація (mutation). Після того, як закінчиться стадія кросинговера, нащадки можуть піддаватися випадковим модифікаціям, названим **мутаціями** (рис. 5.10). У простому випадку в кожній хромосомі, яка піддається мутації, кожен біт з вірогідністю P_m змінюється на протилежний (це так звана *одноточкова мутація*).

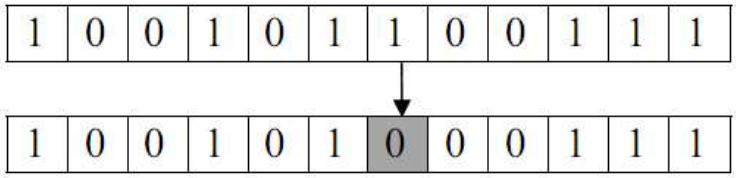


Рис. 5.10. Приклад дії мутації.

Складнішим різновидом мутації є оператори *інверсії* і *транслокації*. Інверсія - це перестановка генів в зворотному порядку усередині навдогад вибраної ділянки хромосоми (рис. 5.11).

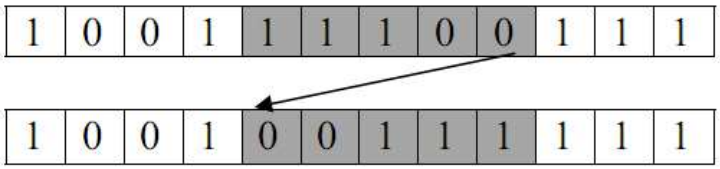


Рис. 5.11. Приклад дії інверсії.

Транслокація - це перенесення якої-небудь ділянки хромосоми, в інший сегмент цієї ж хромосоми (рис. 5.12).

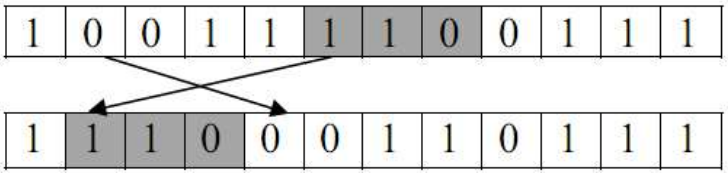


Рис. 5.12. Приклад дії транслокації.

Слід зазначити, що всі перераховані генетичні оператори (одноточковий і багатоточковий кросинговер, одноточкова мутація, інверсія транслокація) мають схожі біологічні аналоги.

У деяких роботах пропонується використовувати стратегію *інцеста* як механізму самоадаптації оператора мутації. Вона полягає в тому, що вірогідність мутації кожного гена P_m визначається для кожного нащадка на підставі генетичної близькості його батьків. Наприклад, це може бути відношення числа співпадаючих генів батьків до загального числа генів хромосоми. Це приводить до цікавого ефекту - при високій різноманітності генофонду популяції наслідки мутації будуть мінімальними, що дозволяє операторові схрещування працювати без стороннього втручання. У разі ж пониження різноманітності, що виникає в основному при застряванні алгоритму в локальному оптимумі, наслідки мутації стають відчутнішими, а при повному сходженні популяції алгоритм просто стає стахостичним, що збільшує вірогідність виходу популяції з локального оптимуму.

Інколи (з метою підвищення середньої пристосованості популяції) допустимо здійснювати *направлені мутації*, тобто після кожної зміни

хромосоми перевіряти, чи підвищилася в результаті цієї мутації її пристосованість і, якщо немає, повертати хромосому до вихідного стану.

Формування нового покоління. Після схрещування і мутації особин необхідно вирішити проблему про те, які з нових особин увійдуть до наступного покоління, а які - ні, і що робити з їх предками. Є два найбільш поширених способи.

1. Нові особини (нащадки) займають місця своїх батьків. Після чого настає наступний етап, в якому нащадки оцінюються, відбираються, дають потомство і поступаються місцем своїм «дітям».
2. Наступна популяція включає як батьків, так і їх нащадків.

У другому випадку необхідно додатково визначити, які з особин батьків і нащадків попадуть в нове покоління. У простому випадку, в нього після кожного схрещування включаються дві кращі особини з четвірки батьків і їх нащадків. Ефективнішим є механізм *витіснення*, який реалізується таким чином, що прагне видаляти «схожі» хромосоми з популяції і залишати ті, що відрізняються.

Елітарний відбір (Elite selection). Створюється проміжна популяція, яка включає як батьків, так і їх нащадків. Члени цієї популяції оцінюються, а потім з них вибираються N найкращих, які і увійдуть до наступного покоління. Часто даний метод комбінують з іншими - вибирають в нову популяцію, наприклад, 10% «елітних» особин, а останні 90% - одним з традиційних методів селекції. Інколи ці 90% особини створюють випадково, як при створенні початкової популяції перед запуском роботи генетичного алгоритму. Використання стратегії елітизма виявляється вельми корисним для ефективності ГА, оскільки не допускає втрату кращих рішень. Наприклад, якщо популяція зійшлася в локальному максимумі, а мутація вивела один з рядків в область глобального, то при попередній стратеги досить імовірно, що ця особина в результаті схрещування буде втрачена, і рішення задачі не буде отримано. Якщо ж використовується елітизм, то отримане хороше рішення залишатиметься в популяції до тих пір, поки не буде знайдено ще краще.

Відбір витісненням (Exclusion selection). У даному відборі вибір особини в нову популяцію залежить не лише від величини її придатності, але і від того, чи є вже у формованій популяції особина з аналогічним хромосомним набором. Відбір проводиться з числа батьків і їх нащадків. Зі всіх особин з однаковою пристосованістю перевага спочатку віддається особинам з різними генотипами. Таким чином, досягаються дві мета: по-перше, не втрачаються кращі знайдені рішення, що володіють різними хромосомними наборами, по-друге, в популяції постійно підтримується генетична різноманітність. Витіснення в даному випадку формує нову популяцію швидше з видалених особин, замість особин, що групуються біля поточного знайденого рішення. Даний метод найбільш придатний для багатоекстремальних задач, при цьому окрім визначення глобальних екстремумів з'являється можливість виділити і ті локальні максимуми, значення яких близькі до глобальних.

Метод Больцмана, або метод відпалу (Bolzman selection). У даному методі вірогідність відбору в нову популяцію залежить від управляючого параметра - температури T. Зазвичай імовірність попадання в нову популяцію обчислюється за наступною формулою

$$p = \frac{1}{1 + e^{\frac{f(i) - f(j)}{T}}},$$

де $f(i)$ та $f(j)$ - значення цільової функції i та j особин, відповідно. Номери особин i та j вибираються випадково. Якщо значення p виявиться більше випадкового числа на інтервалі (0; 1), то в нову популяцію попаде особина $f(i)$, інакше $f(j)$.

В деяких випадках застосовується альтернативна формула

$$p = \frac{\exp(f(i)/T)}{\{\exp(f(j)/T)\}}$$

де $\{\}$ - середнє по популяції на ітерації з номером t. Якщо p виявиться більше випадкового числа на інтервалі (0; 1), то особина $f(i)$ попаде в нову популяцію.

У даному методі першим поколінням відповідають високі температури, і вірогідність відбору особин велика (підтримується різноманіття у новій популяції). Поступово із зростанням кількості поколінь ГА температура знижується, імовірність відбору зменшується і в нову популяцію потрапляють ті особини, пристосованість яких мінімальна.

Останов алгоритму. Робота ГА є ітераційним процесом, який продовжується до тих пір, поки не пройде задане число поколінь або не виконається який-небудь інший критерій останову. У оптимізаційних задачах традиційними критеріями останову алгоритму є, наприклад, тривала відсутність прогресу в сенсі поліпшення значення середньої (або кращої) пристосованості популяції, мала різниця між кращим і гіршим значенням пристосованості для поточної популяції і тому подібне

5.3. Моделі генетичних алгоритмів.

Канонічний ГА (Canonical GA - J. Holland).

Ця модель алгоритму є класичною. Вона була запропонована Джоном Холландом в його знаменитій роботі «Адаптація в природних і штучних середовищах» [62]. Часто можна зустріти опис простого ГА (Simple GA), він відрізняється від канонічного тим, що використовує або рулеточний, або турнірний відбір. Модель канонічного ГА має наступні характеристики.

- Фіксований розмір популяції.
- Фіксована розрядність генів.
- Пропорційний відбір.
- Одноточечний кросовер і одноточечна мутація.
- Наступне покоління формується з нащадків поточного покоління без «елітизму».

Алгоритм роботи ГА (*репродуктивний план Холланда*) має наступний вигляд.

1. Ініціалізація початкової популяції. Покласти номер епохи $t = 0$. Ініціалізувати випадковим чином m генотипів особин і сформувати з них випадкову популяцію. Обчислити пристосованість особин популяції $F(0) = (f_1(0),...,f_m(0))$, а потім - середню пристосованість по популяції $f_{ср}(0) = \sum_{i=1}^{m} f_i(0)/m$.

2. Вибір батьків для схрещування. Збільшити номер епохи на одиницю: $t = t + 1$. Визначити випадковим чином номер першого батька $l \in \{1,...,m\}$, призначивши імовірність випадання будь-якого номера h пропорційній величині $f_h(t)/f_{ср}(t)$. Повторним випробуванням визначити номер другого батька k.

3. Формування генотипу нащадків. Із заданою ймовірністю p_c провести над генотипами вибраних батьків одноточковий кросовер. Далі до кожного з отриманих нащадків з імовірністю p_M застосувати оператора мутації.

4. Оновлення популяції. Помістити нащадків в популяцію, заздалегідь видаливши з неї батьків. Обчислити пристосованості нащадків і відновити значення середньої пристосованості популяції $f_{ср}(t)$. Якщо формування популяції не завершене, перейти до кроку 2.

Генітор (Genitor).

У моделі генітор використовується специфічний спосіб відбору [31]. Спочатку, як і завжди, популяція ініціалізується, і її особини оцінюються. Потім вибираються випадковим чином дві особини, схрещуються, причому виходить лише один нащадок, який оцінюється і займає місце менш пристосованої особини в популяції (а не одного з батьків). Після цього знову випадковим чином вибираються дві особини, і їх нащадок займає місце батьківської особини з найнижчою пристосованістю. Таким чином, на кожному кроці в популяції оновлюється лише одна особина. Процес продовжується до тих пір, поки придатності хромосом не стануть однаковими. У даний алгоритм можна додати мутацію нащадка після його створення. Критерій закінчення

процесу, як і вигляд кросинговера і мутації, можна вибирати різними способами. Підводячи підсумки, можна виділити наступні характерні особливості.

- Фіксований розмір популяції.
- Фіксована розрядність генів.
- Особини для схрещування вибираються випадковим чином.
- Обмежень на типа кросовера і мутації немає.
- В результаті схрещування особин виходить один нащадок, який займає місце найменш пристосованої особини.

Метод переривистої рівноваги.

Даний метод заснований на палеонтологічній теорії переривистої рівноваги, яка описує швидку еволюцію за рахунок вулканічних і інших змін земної кори [18]. Для вживання даного методу в задачах інтелектуального аналізу даних пропонується після кожної генерації проміжного покоління випадковим чином перемішувати особині в популяції, а потім застосовувати основний ГА. У даній моделі для відбору батьківських пар використовується панміксія. Нащадки, що вийшли в результаті кросинговера, і найбільш придатні батьки випадковим чином змішуються. Із загальної маси в нове покоління попадуть лише ті особини, придатність яких вище середньою. Тим самим досягається управління розміром популяції залежно від наявності кращих особин. Така модифікація методу переривистої рівноваги може дозволити скоротити неперспективні популяції і розширити популяції, в яких знаходяться кращі індивідуальності.

Гібридний алгоритм (Hybrid Algorithm - L. «Dave» Davis).

Ідея гібридних алгоритмів полягає в поєднанні генетичного алгоритму з деяким іншим класичним методом пошуку, відповідним до даної задачі [34]. У кожному поколінні всі згенеровані нащадки оптимізуються вибраним методом і потім заносяться в нову популяцію. Тим самим виходить, що кожна особина в популяції досягає локального оптимуму, поблизу якого вона знаходиться. Далі виробляються звичайні для ГА дії: відбір батьківських пар, кросинговер і

мутації. На практиці гібридні алгоритми виявляються дуже вдалими. Це пов'язано з тим, що вірогідність попадання однієї з особин в область глобального максимуму досить велика. Після оптимізації така особина буде рішенням задачі. Відомо, що генетичний алгоритм здатний швидко знайти у всій зоні пошуку хороші рішення, але він може зазнавати труднощі в здобутті з них найкращих. Звичайний оптимізаційний метод може швидко досягти локального максимуму, але не може знайти глобальний. Поєднання двох алгоритмів дозволяє використовувати переваги обох.

CHC (Eshelman).

CHC розшифровується як Cross-population selection, Heterogeneous recombination and Cataclysmic mutation [7]. Даний алгоритм досить швидко сходиться через те, що в ньому немає мутацій, наступних за оператором кросинговера, використовуються популяції невеликого розміру, і відбір особин в наступне покоління ведеться і між батьківськими особинами, і між їх нащадками. У даному методі для кросинговера вибирається випадкова пара, але не допускається, аби між батьками була маленька хеммінгова відстань або мала відстань між крайніми бітами. Для схрещування використовується різновид однорідного кросовера HUX (Half Uniform Crossover), при якому нащадкові переходить рівно половина бітів кожного батька. Для нового покоління вибираються N кращих різних особин серед батьків і дітей. При цьому дублювання рядків не допускається. У моделі CHC розмір популяції відносно малий - близько 50 особин. Це виправдовує використання однорідного кросинговера і дозволяє алгоритму зійтися до рішення. Для здобуття більш менш однакових рядків CHC застосовує cataclysmic mutation: всі рядки, окрім самого пристосованого, піддаються сильній мутації (змінюється біля третини бітів). Таким чином, алгоритм перезапускається і далі продовжує роботу, застосовуючи лише кросинговер.

ГА з нефіксованим розміром популяції (Genetic Algorithm with Varying Population Size - GAVaPS).

У генетичному алгоритмі з нефіксованим розміром популяції кожній особини приписується максимальний вік, тобто число поколінь, після яких особина гине [9]. Впровадження в алгоритм нового параметра - віку - дозволяє виключити оператор відбору в нову популяцію. Вік кожної особини індивідуальний і залежить від її пристосованості. У кожному поколінні t на етапі відтворення звичайним способом створюється додаткова популяція з нащадків. Розмір додаткової популяції (AuxPopsize(t)) пропорційний розміру основної популяції (Popsize(t)) і рівний AuxPopsize(t) = [Popsize(t) p_c], p_c - вірогідність відтворення. Для відтворення особині вибираються з основної популяції з однаковою імовірністю незалежно від їх пристосованості. Після вживання мутації і кросинговера нащадкам приписується вік згідно значенню їх пристосованості. Вік є константою впродовж всієї еволюції особини (від народження до загибелі). Потім з основної популяції видаляються ті особини, термін життя яких витік, і додаються нащадки з проміжної популяції. Таким чином, розмір після однієї ітерації алгоритму обчислюється за формулою Popsize(t+1) = Popsize(t)+AuxPopsize(t)-D(t), де D(t) - число особин, які вмирають в поколінні t.

Простий паралельний ГА (Parallel implementations).

Генетичні алгоритми застосовуються і при паралельних обчисленнях [47] При цьому формується декілька популяцій, що живуть окремо. На етапі формування нового покоління за деяким правилом відбираються особини з різних популяцій. Згенерована таким чином популяція, в деяких випадках замінює всі інші. Таким чином, так або інакше, відбувається міграція особин однієї популяції в інші популяції. Тому паралельні ГА називають *міграціями*.

Спершу розглянемо створення простого паралельного ГА з класичної моделі Холланда. Для цього використовуватимемо турнірний відбір. Заведемо $N/2$ процесів (тут і далі процес розглядається як деяка машина, процесор, якої може працювати незалежно). Кожен з них вибиратиме випадково з популяції 4 особини, проводити 2 турніри, і переможців схрещувати. Отримані нащадки

записуватимуться в нове покоління. Таким чином, за один цикл роботи одного процесу змінюється ціле покоління.

Міграція (Migration).

Модель міграції представляє популяцію як множину підпопуляцій. Кожна підпопуляція обробляється окремим процесором. Ці підпопуляції розвиваються незалежно одна від одної протягом однакової кількості поколінь T (час ізоляції). Після закінчення часу ізоляції відбувається обмін особинами між підпопуляціями (міграція). Кількість особин, що піддалися обміну (вірогідність міграції), метод відбору особин для міграції і схема міграції визначає частоту виникнення генетичного різноманіття в підпопуляціях і обмін інформацією між популяціями. Відбір особин для міграції може відбуватися таким чином:

- випадкова одноманітна вибірка з числа особин;
- пропорційний відбір: для міграції беруться найбільш придатні особини.

Окремі підпопуляції в паралельних ГА можна умовно прийняти за вершини деякого графа. У зв'язку з цим можна розглядати топологію графа міграційного ГА. Найбільш поширеною топологією міграції є повний граф (рис. 5.13), при якій особині з будь-якої підпопуляцій можуть мігрувати в будь-яку іншу підпопуляцію. Для кожної підпопуляції повна кількість потенційних іммігрантів будується на основі всіх підпопуляцій. Мігруюча особина випадковим чином вибирається з цього загального числа.

При використанні в необмеженій міграції пропорційного відбору спочатку формується масив з найбільш придатних особин, відібраних по всіх підпопуляціях. Випадковим чином з цього масиву вибирається особина, і нею замінюють найменш придатну особину в підпопуляції 1. Аналогічні дії проробляємо з останніми підпопуляціями. Можливо, що якась популяція отримає дублікат своєї «хорошої» особини.

Інша основна міграційна схема - це топологія кільця (рис. 5.14). Тут особини передаються між сусідніми (по напряму обходу) підпопуляціями.

Таким чином, особини з однієї підпопуляції можуть мігрувати лише в одну - сусідню підпопуляцію.

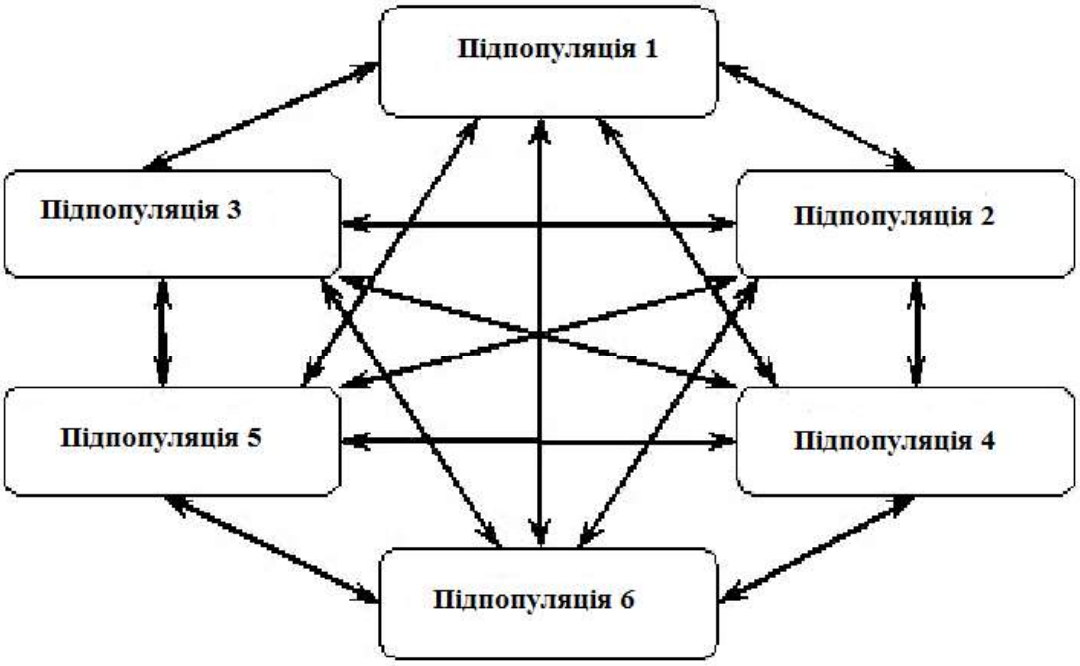


Рис. 5.13. Міграція з топологією повної мережі.

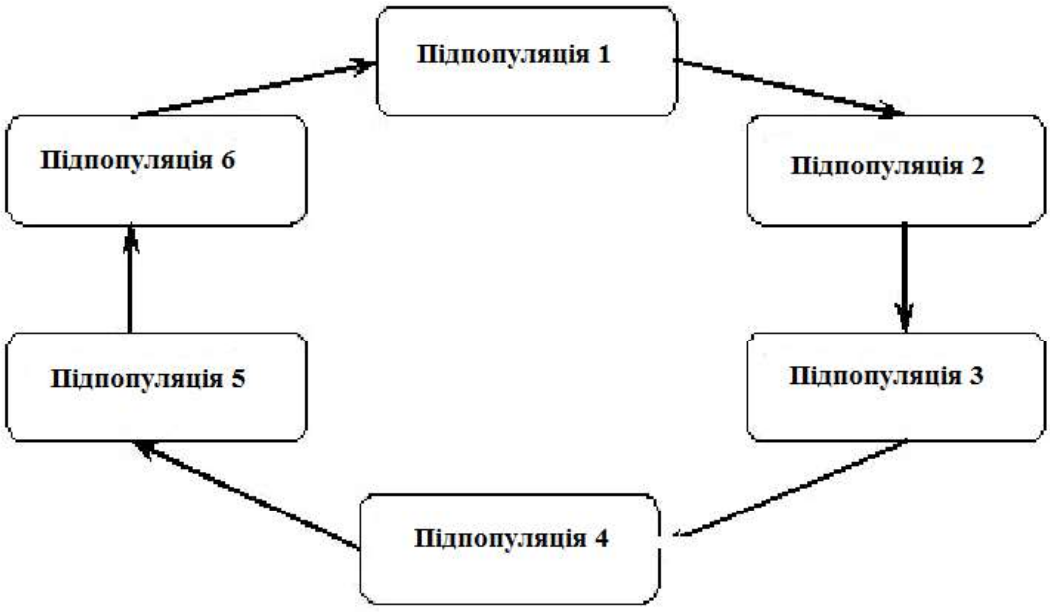


Рис. 5.14. Міграція з топологією кільця.

Модель дифузії, або острівна модель ГА (Island Model GA).

Острівна модель є найбільш поширеною моделлю паралельного ГА. Її суть полягає в тому, що популяція, яка як правило складається з дуже великого

числа особин, розбивається на однакові за розміром підпопуляції. Кожна підпопуляція обробляється окремим процесором за допомогою одного з різновидів непаралельного ГА. Зрідка, наприклад, через п'ять поколінь, підпопуляції обмінюватимуться декількома особинами. Такі міграції дозволяють підпопуляціям спільно використовувати генетичний матеріал.

Хай виконуються 16 незалежних генетичних алгоритмів, використовуючи підпопуляції з 1000 особин на кожному процесорі. Якщо міграцій немає, то відбувається 16 незалежних пошуків рішення. Всі пошуки ведуться на різних початкових популяціях і сходяться до певних особин. Дослідження підтверджують, що генетичний дрейф схильний приводити підпопуляції до різних домінуючих особин. Це пов'язано з тим, що, по-перше, кількість островів, що приймають домінуючих «емігрантів» з острова, обмежена (2 - 5 островів). По-друге, обмін особинами однобічний. Тому у великій популяції з'являться групи островів з різними домінуючими особинами. Якщо популяція невеликого розміру, то можлива швидка міграція помилкових домінуючих особин. Наприклад, дійсне рішення знаходиться лише на одному острові, а декілька помилкових домінант - на інших островах. Тоді при міграції кількість помилкових особин на островах зросте (на кожен острів міграції відбуваються з не менш 2 островів), генетичним алгоритмом вірне рішення буде зруйновано. Тим самим в маленькій популяції при генетичному дрейфі можлива поява помилкових домінуючих особин і сходження алгоритму до помилкового оптимуму.

Введення міграцій в острівній моделі дозволяє знаходити різні особини-домінанти в підпопуляціях, що сприяє підтримці різноманіття в популяції. Кожну підпопуляції можна прийняти за острів. Під час міграції підпопуляції обмінюються своїм домінуючим генетичним матеріалом. При частій міграції великої кількості особин відбувається перемішування генетичного матеріалу. Тим самим усуваються локальні відмінності між островами. Дуже рідкі міграції не дозволяють запобігти передчасній збіжності алгоритму в маленьких підпопуляціях. У даній моделі з кожного острова міграції можуть відбуватися

лише на певну відстань: 2 - 5 островів залежно від кількості підпопуляцій. Таким чином, кожен острів виявляється майже ізольованим. Кількість островів, на які можуть мігрувати особини однієї підпопуляції, називають відстанню ізоляції. Слід зауважити, що взаємоміграції виключені (рис. 5.15).

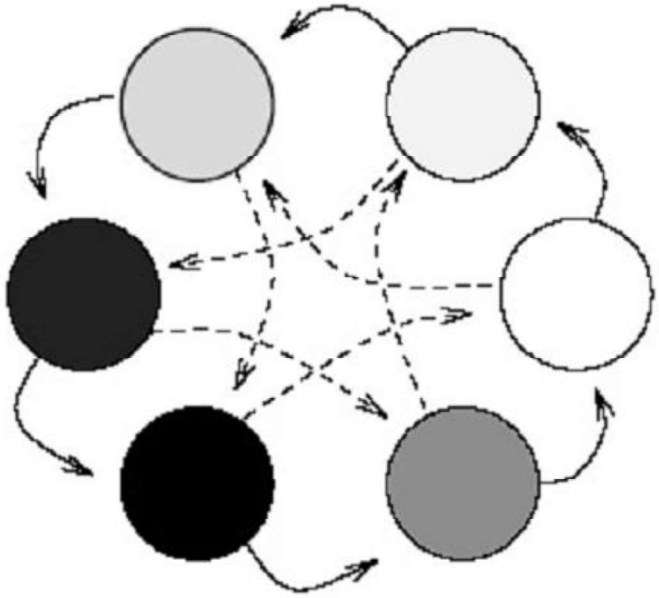


Рис. 5.15. Острівна модель.

Шима (Schema).

Хоча зовні здається, що ГА обробляє рядки, насправді при цьому неявно відбувається обробка шим, які представляють шаблони подібності між рядками. ГА практично не може займатися повним перебором всіх представлень в просторі пошуку. Проте він може виробляти вибірку значного числа гіперплощин в зонах пошуку з високою пристосованістю. Кожна така гіперплощина відповідає множині схожих рядків з високою пристосованістю. Шима - це рядок довжини l (як і довжина будь-якого рядка популяції), що складається із знаків алфавіту {0; 1; *}, де {*} - невизначений символ. Кожна шима визначає множину всіх бінарних рядків довжини l, що мають у відповідних позиціях або 0, або 1, залежно від того, який біт знаходиться у відповідній позиції самої шими. Шима, що не містить жодного невизначеного символу, є деяким рядком. Шима з одним невизначеним символом описує два бінарні рядки, а з двома — чотири рядки. Наприклад, шима, 10 * * 1, визначає

собою множину з чотирьох п'ятибітових рядків {10001; 10011; 10101; 10111}. Неважко відмітити, що шима з r невизначеними символами описує 2^r бінарних рядків. З іншого боку, кожен рядок довжини m описується 2^m шимами. Отже, в популяції з n таких рядків число можливих шим може досягати $n2^m$! При цьому велика частина шим ймовірно буде менш пристосованою чим інші, що може привести до епістазу. Тому рекомендується створювати початкову популяцію з шим з високою пристосованістю. Всі шими різні між собою. Основними характеристиками шин є порядок і довжина [7, 31].

Порядок шими $o(S)$ (order) - це число фіксованих бітів (0 або 1) в шимі S.

Визначальна довжина $\delta(S)$ (defining length) - це відстань між першим і останнім фіксованими бітами в шимі S. Довжина шими визначає концентрацію інформації в шимі. Вважається, що шима з однією фіксованою позицією має нульову довжину. Наприклад, шима $S = (***001*110)$ має порядок $o(S) = 6$ і довжину $\delta(S) = 10 - 4 = 6$. Порядок і довжина шим використовуються для визначення вірогідності мутації і кросинговера відповідно.

У зв'язку з тим, що більш пристосовані особини (хромосоми) описуються шимою з більшою пристосованістю, сенс роботи ГА полягає в пошуку двійкового рядка певного вигляду зі всієї множини бінарних рядків довжини m. Тоді простір пошуку складає 2^m рядків, а його розмірність рівна m. Шима відповідає деякій гіперплощині в цьому просторі. Дане ствердження можна проілюструвати таким чином. Хай розрядність хромосоми дорівнює 3, тоді всього можна закодувати $2^3 = 8$ рядків. Представимо куб в тривимірному просторі. Позначимо вершини цього куба трьохрозрядними бінарними рядками так, щоб мітки сусідніх вершин відрізнялися рівно на один розряд, причому вершина з міткою «000» знаходилася б на початку координат (рис. 5.16). Якщо узяти шиму вигляду «**0», то вона опише ліву грань куба, а шима «*10» - верхнє ребро цієї грані. Вочевидь, що шима «***» відповідає всьому простору.

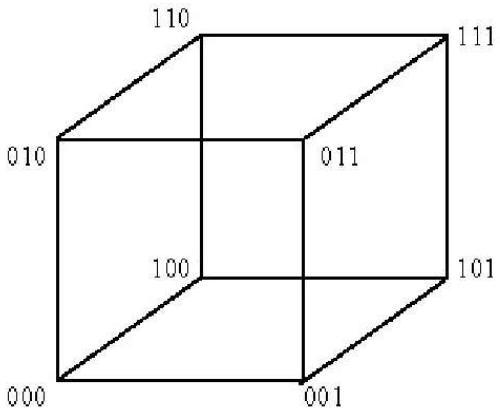


Рис. 5.16. Тривимірний куб.

Тут шимі «*1**» відповідає гіперплощина, що включає задні грані зовнішнього і внутрішнього куба, а шимі «**10» — гіперплощина з верхніми ребрами лівих граней обох кубів. Таким чином терміни «гіперплощина» і «шима» взаємозамінні.

Будівельні блоки - це шими, що володіють:

- високою придатністю;
- низьким порядком;
- короткою певною довжиною.

Придатність шими визначається як середнє придатності рядків-особин, які її містять. Після процедури відбору залишаються лише рядки з вищою придатністю. Отже, рядки, які є прикладами шим з високою придатністю, вибираються частіше. Кросинговер рідше руйнує шими з коротшою певною довжиною, а мутація рідше руйнує шими з низьким порядком. Тому, такі шими мають більше шансів переходити з покоління в покоління. Холланд показав, що в той час, як ГА явним чином обробляє n рядків на кожному поколінні, неявно обробляються порядка n^3 таких коротких шим низького порядку і з високою пристосованістю (корисних шим - useful schemata). Він називав це явище *неявним паралелізмом* (implicit parallelism). Для рішення реальних задач, присутність неявного паралелізму означає, що велика популяція має більше можливостей локалізувати рішення експоненціально швидше за популяцію з меншим числом особин.

Теорема шим (The schema theorem).

Теорема шим показує, яким чином простий ГА експоненціально збільшує число прикладів корисних шим або будівельних блоків, що приводить до знаходження рішення вихідної задачі.

Хай $m(H,t)$ - число прикладів шими H в t поколінні. Обчислимо очікуване число прикладів H в наступному поколінні або $m(H,t+1)$ в термінах $m(H,t)$. Простий ГА кожному рядку при відборі ставить у відповідність імовірність її «виживання» пропорційно її пристосованості (наприклад, як в методі рулетки). Очікується, що шима H може бути вибрана $m(H,t)(f(H)/f_{ср})$ разів, де $f_{ср}$ - середня придатність популяції, а $f(H)$ - середня придатність тих рядків в популяції, які являються прикладами H. Ймовірність того, що одноточечний кросинговер зруйнує шиму дорівнює ймовірності того, що точка розриву попаде між певними бітами. Ймовірність же того, що H «переживає» кросинговер не менше $1-p_c(\delta(H)/l-1)$, де p_c - ймовірність кросинговера. Ця ймовірність - нерівність, оскільки шима зможе вижити, якщо в кросинговері також брав участь приклад подібної шими.

Ймовірність того, що H переживе точкову мутацію - $(1-p_m)^{o(H)}$, де p_m - ймовірність мутації. Цей вираз можна апроксимувати як $(1-o(H))$ для малих p_m і $o(H)$. Множення очікуваного число відборів і ймовірності виживання відоме як *теорема шим*

$$\langle m(H,t+1)\rangle \geq m(H,t)\frac{f(H,t)}{\overline{f(t)}}[1-p_c\frac{\delta(H)}{l-1}](1-p_m)^{o(H)}.$$

Теорема шим показує, що будівельні блоки зростають по експоненті, у той же час шими з пристосованістю нижче середньою розпадаються з тією ж швидкістю. Голдберг в своїх дослідженнях теореми шим висуває гіпотезу будівельних блоків, яка полягає в тому, що «будівельні блоки об'єднуються, аби сформувати кращі рядки». Тобто рекомбінація і експоненціальне зростання будівельних блоків веде до формування кращих будівельних блоків.

Тоді як теорема шим передбачає зростання прикладів хороших шим, сама теорема вельми спрощено описує поведінку ГА. Перш за все, $f(H)$ і $f_{ср}$ не залишаються постійними від покоління до покоління. По-друге, теорема шим пояснює втрати шим, але не появу нових. Нові шими часто створюються кросинговером і мутацією. Крім того, в результаті еволюції члени популяції стають все більш і більш схожими один на одного так, що зруйновані шими будуть відразу ж відновлені. Нарешті, доведення теореми шим побудоване на елементах теорії вірогідності і, отже, не враховує розкид значень. У багатьох задачах розкид значень придатності шими може бути досить великий, роблячи процес формування шим дуже складним.

Істотна різниця придатності шими може привести до збіжності та неоптимального рішення. Не дивлячись на простоту, теорема шим описує декілька важливих аспектів поведінки ГА. Мутації з більшою вірогідністю руйнують шими високого порядку, тоді як кросинговер з більшою вірогідність руйнує шими з більшою певною довжиною. Коли відбувається відбір, популяція сходиться пропорційно відношенню пристосованості кращої особини, до середньої пристосованості в популяції: це відношення - *міра тиску відбору («selection pressure»)*. Збільшення p_c чи p_m, або зменшення тиску відбору веде до збільшеного здійснення вибірки або дослідження простору пошуку, але не дозволяє використовувати все хороші шими, які має в своєму розпорядженні ГА. Зменшення p_c, чи p_m, або збільшення тиску вибору веде до поліпшення використання знайдених шим, але гальмує дослідження простору у пошуках нових хороших шим. Моделювання ГА передбачає збереження рівноваги ГА між тим і іншим, що зазвичай відоме як проблема «балансу дослідження і використання».

Деякі дослідники критикують зазвичай швидку збіжність ГА, заявляючи, що випробування величезних кількостей шим, що перекриваються, вимагає більшої вибірки і повільнішої, більш керованій збіжності. Методологія управління збіжністю простого ГА до цих пір не вироблена.

Недоліками теореми шим є те, що вона:

- застосовується лише до канонічного ГА;
- не враховує ту обставину, що кросинговер і мутація можуть не лише руйнувати шиму, але створювати її з інших шим. Тому в теоремі шим присутній знак нерівності;
- дозволяє розрахувати долю шим в популяції лише для наступного покоління, тобто при спробі підрахувати число рядків, відповідних даній шимі, через декілька поколінь з використанням теореми шим до успіху не приведе.

Неперервні генетичні алгоритми.

При роботі з оптимізаційними задачами в неперервних просторах цілком природно представляти гени безпосередньо дійсними числами. В цьому випадку хромосома є вектор дійсних чисел. Довжина хромосоми збігатиметься з довжиною вектора-рішення оптимізаційної задачі, інакше кажучи, кожен ген відповідатиме за одну змінну. Генотип об'єкту стає ідентичним його фенотипу. Вищесказане визначає список основних переваг неперервних алгоритмів.

1. Використання неперервних генів робить можливим пошук у великих просторах (навіть у невідомих), що важко робити в разі двійкових генів, коли збільшення простору пошуку скорочує точність вирішення при незмінній довжині хромосоми.

2. Однією з важливих рис неперервних ГА є їх здібність до локального налаштування рішень.

3. Використання неперервних алгоритмів для представлення рішень зручно, оскільки близько до постановки більшості прикладних задач. Крім того, відсутність операцій кодування/декодування, які необхідні в класичному ГА, підвищує швидкість роботи алгоритму.

Як відомо, появу нових особин в популяції канонічного ГА забезпечують декілька біологічних операторів: відбір, схрещування і мутація. Як оператори відбору особин в батьківську пару тут підходять будь-які відомі : пропорційний, турнірний, відбір усіканням. Проте оператори схрещування і мутації не годяться: у класичних реалізаціях вони працюють з бітовими рядками. Потрібні власні реалізації, що зважають на специфіку неперервних

алгоритмів. Оператор схрещування неперервного ГА породжує одного або декількох нащадків від двох хромосом. Власне кажучи, потрібно з двох векторів дійсних чисел отримати нові вектори за якими-небудь законами. Більшість неперервних алгоритмів генерують нові вектори в околиці батьківських пар. Спершу розглянемо прості і популярні кросовери.

Хай $C_1 = (c_1^1, c_2^1, ..., c_n^1)$ та $C_2 = (c_1^2, c_2^2, ..., c_n^2)$ - дві хромосоми, вибрані оператором селекції для схрещування.

Плоский кросовер (flat crossover). Створюється нащадок $H = (h_1, ..., h_k, ..., h_n)$, де h_k - випадкове число з інтервалу $[c_k^{\min}, c_k^{\max}]$, $c_k^{\min} = \min(c_k^1, c_k^2)$, $c_k^{\max} = \max(c_k^1, c_k^2)$.

Простий кросовер (simple crossover). Випадковим чином вибирається число k з інтервалу $\{1,2,\ldots,n\text{-}1\}$ і генеруються два нащадки $H_1 = (c_1^1, c_2^1, ..., c_k^1, c_{k+1}^2, ..., c_n^2)$ та $H_2 = (c_2^2, c_2^2, ..., c_k^2, c_{k+1}^1, ..., c_n^1)$.

Арифметичний кросовер (arithmetical crossover). Створюються два нащадки $H_1 = (h_1^1, ..., h_n^1)$, $H_2 = (h_1^2, ..., h_n^2)$, де $h_k^1 = w^* c_k^1 + (1-w) * c_k^2$, $h_k^2 = w * c_k^2 + (1-w) * c_k^1$. Величина w або константа (рівномірний арифметичний кросовер) з інтервалу [0;1], або змінюється із збільшенням епох (нерівномірний арифметичний кросовер).

Геометричний кросовер (geometrical crossover). Створюються два нащадки $H_1 = (h_1^1, ..., h_n^1)$, $H_2 = (h_1^2, ..., h_n^2)$, де $h_k^1 = (c_k^1)^w (c_k^2)^{1-w}$, $h_k^2 = (c_k^2)^w (c_k^1)^{1-w}$. Величина w - випадкове число з інтервалу [0;1]. (Застосовується в тому випадку, якщо вектори C_1 і C_2 не негативні)

Дискретний кросовер (discrete crossover). Кожен ген h_k вибирається випадково по рівномірному закону з кінцевої множини $\{c_k^1, c_k^2\}$.

Евристичний кросовер (Wright's heuristic crossover). Хай C_1 - один з двох батьків з кращою пристосованістю. Тоді $h_k = w * \left| c_k^1 - c_k^2 \right| + c_k^1$, де w - випадкове число з інтервалу [0;1].

В якості оператора мутації найбільшого поширення набула *випадкова мутація*. При випадковій мутації ген, що підлягає зміні, набуває випадкового значення з інтервалу своєї зміни. Розглянуті кросовери історично були запропоновані першими, проте в багатьох задачах їх ефективність виявляється невисокою. Пізніше були розроблені покращені оператори схрещування, аналітична формула яких і ефективність обґрунтовані теоретично. Розглянемо детальніше один з таких кросоверів - SBX.

SBX кросовер (Simulated Binary Crossover). SBX – кросовер, імітуючий двійковий. Він був розроблений в 1995 році дослідницькою групою під керівництвом K. Deb'a. Як випливає з його назви, цей кросовер моделює принципи роботи двійкового оператора схрещування.

SBX кросовер був отриманий наступним способом. Автором було введено поняття *сили пошуку кросовера* (search power). Це кількісна величина, що характеризує розподіл вірогідності появи будь-якого нащадка від двох довільних батьків. Спочатку була розрахована сила пошуку для одноточкового двійкового кросовера, а потім був розроблений вещественний кросовер SBX з такою ж силою пошуку. У ньому сила пошуку характеризується розподілом вірогідності випадкової величини β

$$p(\beta) = \begin{cases} 0.5(n+1)\beta^n, \beta \le 1 \\ 0.5(n+1)\beta^{-(n+2)}, \beta > 1 \end{cases}.$$

Для генерації нащадків використовується наступний алгоритм, що використовує наступний вираз для $p(\beta)$. Створюються два нащадки $H_k = (h_1^k, ..., h_j^k, ..., h_n^k)$, $k = 1,2$, де $h_1^1 = 0.5[(1-\beta)c_1^1 + (1+\beta)c_1^2]$, $h_1^2 = 0.5[(1+\beta)c_1^1 + (1-\beta)c_1^2]$, β - число, отримане по формулі

$$\beta(u) = \begin{cases} (2u)^{\frac{1}{n+1}}, u(0,1) \le 0.5 \\ (\frac{1}{2(1-u)})^{\frac{1}{n+1}}, u(0,1) > 0.5 \end{cases}.$$

У формулі $u(0,1)$ - випадкове число, розподілене по рівномірному закону, $n \in [2,5]$ - параметр кросовера. На рисунку 5.17 приведена геометрична

інтерпретація роботи SBX кросовера при схрещуванні двох хромосом, відповідних дійсним числам 2 і 5. Видно, як параметр n впливає на кінцевий результат: збільшення n спричиняє за собою збільшення ймовірності появи нащадка в околиці батька і навпаки.

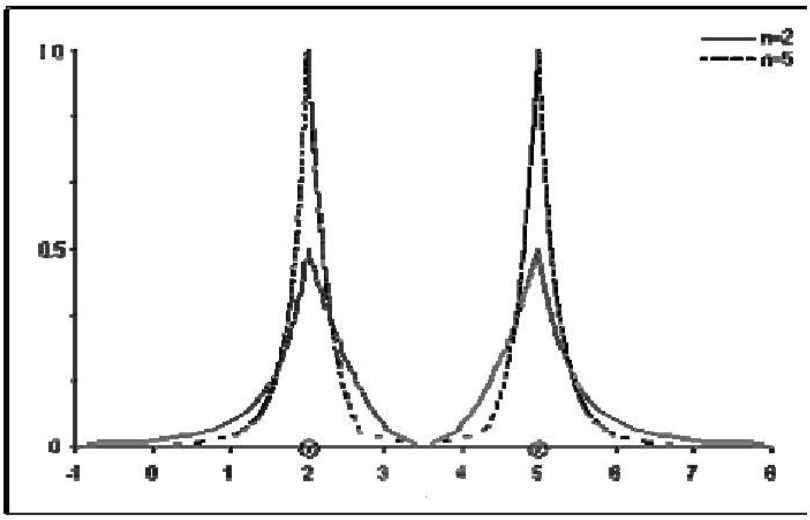


Рис. 5.17. Геометрична інтерпретація роботи SBX кросовера.

Експерименти автора SBX кросовера показали, що він у багатьох випадках ефективніше змішаного кросовера, хоча не існує жодного кросовера, ефективного у всіх випадках. Дослідження показують, що використання декількох операторів кросовера дозволяє зменшити ймовірність передчасної збіжності, тобто поліпшити ефективність алгоритму оптимізації в цілому. Для цього можуть використовуватися спеціальні стратегії, що змінюють ймовірність вживання окремого еволюційного оператора залежно від його «успішності», або використання гібридних кросоверів. В будь-якому разі, якщо стоїть задача оптимізації в безперервних просторах з використанням еволюційних стратегій, то слід зробити вибір на користь безперервного генетичного алгоритму.

5.4. Програмне забезпечення та сфери застосування генетичних алгоритмів.

Доля нових наукових напрямів - при всій їх революційності, незвичності і парадоксальності - зазвичай розвивається за канонами класичної мелодрами, повторюючи вічну казку про Попелюшку. Цей нехитрий сюжетний хід прослідкується в розвитку практично всіх нових наук - і теорії нейронних мереж, і нечіткої логіки, і системної динаміки, і інших. Всіх, окрім однієї. Цього не сталося з напрямом artificial life (або A-life, як охрестили нову науку американські журналісти), основу якого складають генетичні алгоритми [2, 25].

Отже, що таке artificial life? Найзагальніше визначення дає журнал MIDRANGE Systems, трактуючи A-life як «комп'ютерне моделювання живих об'єктів». Проте на практиці до A-life прийнято відносити комп'ютерні моделі, що володіють рядом конкретних особливостей. По-перше, центральна модель системи - будь то самохідний робот або інтелектуальний агент Internet - володіє здатністю адаптуватися до умов зовнішнього світу, поповнюючи знання про нього шляхом взаємодії з іншими об'єктами і середовищем. По-друге, компоненти системи, розвиваючись в процесі еволюції, здатні передавати свої характерні риси по спадку. Відповідно, присутній механізм породження нових поколінь - шляхом ділення, схрещування або дублювання існуючих об'єктів. По-третє, навколишній світ досить жорстокий і зводить до мінімуму шанси на виживання і появу потомства у слабких і погано пристосованих особин. І, нарешті, присутній механізм породження нових форм (аналог мутацій на реальному світі), що зазвичай містить елемент випадковості. Іншими словами, аби вирішити реальну задачу методами artificial life (наприклад, створити винищувач ідеальної форми, або розробити оптимальну стратегію біржової гри), треба побудувати динамічну модель середовища, в якому належить існувати проектованому об'єкту, населити її множиною різновидів цього об'єкту і дати їм «пожити» декілька поколінь. Слабкі особини відмиратимуть, сильні - схрещуватися, закріплюючи в нових поколіннях свої кращі риси. Через

декілька десятків (інколи - сотень і навіть тисяч) циклів така селекція породить «цивілізацію» практично невразливих особин, ідеально пристосованих до заданої моделі світу. Можна з упевненістю сказати, що «життєва сила» рішення, що пройшло жорстокий відбір, буде достатньою, аби успішно протистояти будь-яким діям конкурентів.

Багато представників великого бізнесу (у тому числі такі гіганти, як Ford) заявляють, що вже давно використовують елементи A-life в ситуаційному моделюванні і ділових іграх для керівників. Наприклад, моделюється ринок США, населений сонмом акул-конкурентів і працюють фірми, віддані в управління кожному з учасників. За один такт гри треба оцінити позиції своєї фірми на різних ринках декількох міст, закрити збиткові філії, реорганізувати неефективні і розширити прибуткові, тобто прийняти в цілому близько п'ятдесяти управлінських рішень. Через півгодини центральний комп'ютер підводить загальні підсумки такту (рівного кварталу роботи фірми) і переходив до наступного «покоління».

Промисловці застосовують методи A-life в першу чергу при створенні всіляких роботів, маніпуляторів і автоматизованих виробництв. Одне з визначень A-life навіть трактує її як теорію управління співтовариством роботів, що вирішують навігаційні завдання шляхом адаптації до зовнішніх умов. Розробники БІС серйозно захоплені розробкою нового покоління мікросхем, що реалізовують базові алгоритми A-life «в кремнії». Піонером тут виступає легендарний професор Mead (який створив п'ятнадцять років назад перший «кремнієвий компілятор», - повністю автоматичну САПР замовлених БІС). Він розробив модель асинхронного нейроподібного кристала, на якому збирається спершу реалізувати моделі ока і вуха, і вже отримав перший патент на однотранзисторну модель синапсу. А на конференції ECAL була продемонстрована перша програмована мікросхема (ППМ), здібна до самоудосконалення. У схему «зашитий» генетичний алгоритм, що оптимізує програмний код обробки зовнішніх дій.

Бізнесменам добре знайома фірма Flavors Technology, яка вирішує велику кількість складноформалізуємих фінансових і управлінських задач, використовуючи методи A-life і теорії хаосу, які реалізовані на спеціалізованій багатопроцесорній системі. Менш відомо, що фірма Thinking Machines, знаменитий розробник суперкомп'ютерів, також має власні програмні напрацювання по втіленню A-life для вирішення ряду оборонних завдань. Осібно коштує така екзотична область, як комп'ютерна вірусологія. Дослідники, що використовують методи A-life, підійшли до цієї проблеми, трактуючи комп'ютерні віруси як специфічний різновид штучного життя, здібний до мутацій, розмноження, інфікування місця існування і самоудосконалення. При такому трактуванні стає реальністю давня ідея про універсальний антивірус, який виявлятиме не конкретні види вірусів (як більшість існуючих програм) а прояви нових форм комп'ютерного життя - зміна дисципліни обробки переривань, нетипова поведінка відомих програм, незнайомий «почерк» спілкування програм з файловою системою і тому подібне Фірма IBM вже анонсувала перший комерційний антивірус, побудований на подібних принципах. Що ж до таких областей, як моделювання катастроф, надзвичайних ситуацій і військових конфліктів, то тут історія ситуаційних центрів, що фактично використовують елементи A-life, налічує вже декілька десятиліть. Багаточисельні приклади таких застосувань можна знайти в «NASA Technology», з якого недавно був знятий гриф секретності.

Скільки завгодно красива і цікава теорія опановує маси лише тоді, коли на ринок виходять прості і зручні інструментальні засоби, що роблять експерименти з новими концепціями загальнодоступними. На щастя, для A-life такий інструмент вже існує. Основним компонентом систем A-life є пакети, що реалізовують генетичні алгоритми. На ринку програмних засобів представлено декілька таких пакетів:

- Arlequin - аналіз генетичних для популяції даних;
- Genepop - генетичний для популяції аналіз;

- Genetix - генетичний для популяції аналіз (програма доступна лише французькою мовою);
- MacClade - комерційна програма для інтерактивного еволюційного аналізу даних;
- MEGA - молекулярно-еволюційний генетичний аналіз;
- Populations - генетичний для популяції аналіз.

Найбільш поширеними вважаються два програмних пакета: пакет Evolver (перший з масових пакетів GA), а також пізніший і потужніший пакет GeneHunter фірми Ward Systems Group. Останній особливо популярний, оскільки входить до складу нейромережевого пакету Ward, активно вживаного банкірами для портфельної гри і валютного ділінга.

GeneHunter. Потужний програмний комплекс під назвою GeneHunter призначений для рішення оптимізаційних задач будь-якої складності за допомогою генетичних алгоритмів і складається з наступних трьох частин: надбудови для MS Excel, динамічної бібліотеки функцій GALIB і прикладів задач, вирішених на пакеті.

Використання надбудови GeneHunter для Excel. Для того, щоб створити модель задачі в GeneHunter, необхідно внести дані, що підлягають обробці, в робочий аркуш Excel і визначити параметри рішення задачі в діалоговому вікні GeneHunter (рис. 5.18).

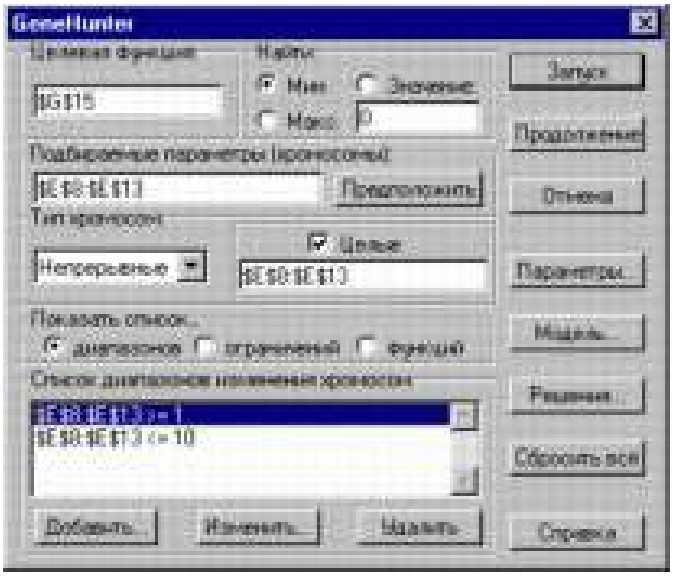


Рис. 5.18. Діалогове вікно GeneHunter.

Діалогове вікно GeneHunter дає можливість вказати ячейки в робочому аркуші, які використовуються при рішенні задачі. Тут також визначається метод рішення і створюється список обмежень, які мають бути дотримані в процесі рішення.

Вікно «Цільова функція» передає в GeneHunter повідомлення про положення ячейки з формулою, по якій визначається, наскільки хороше рішення задачі знайшов GeneHunter. Формула може бути створена за допомогою будь-якої функції Excel, доступної з меню «Вставка». Для створення формул також можна використовувати макроси Excel або функції Excel Visual Basic, які дозволяють вирішувати дуже складні задачі.

Діалогове вікно GeneHunter «Хромосоми» використовується для задання змінних, значення яких треба підібрати для того, щоб вирішити задачу. Їх значення зрештою визначають величину цільової функції. GeneHunter використовує два типи хромосом:

1. Безперервні хромосоми використовуються у тому випадку, коли параметр може набувати значень з деякої безперервної області, наприклад, значення 1,5 усередині діапазону від 0 до 2. Безперервні хромосоми можуть також бути цілими числами, якщо необхідно обмежити простір пошуку.
2. Перерахуємі хромосоми використовуються в задачах пошуку оптимальних комбінацій типа вибору маршруту, складання розкладу занять або послідовності процесів і тому подібне

Діалогове вікно GeneHunter «Діапазони, обмеження і додаткові цільові функції» дозволяє зробити наступне.

- Вказати діапазони значень кожної хромосоми, в яких GeneHunter шукатиме рішення.
- Додати до первинної цільової функції додаткові умови. Такі умови називаються обмеженнями. GeneHunter намагатиметься шукати рішення, які задовольняють обмеженням, але і одночасно оптимізують цільову функцію.
- Вказати ячейки додаткових цільових функцій, значення яких оптимізуватимуться одночасно із значенням головної цільової функції.

Динамічні бібліотеки функцій GALIB. Багато користувачів хочуть використовувати потужність генетичних алгоритмів в своїх додатках, проте вважають за краще розробити власний інтерфейс або скоротити час обчислень складної цільової функції в порівнянні з часом, який ця процедура займає в Excel. Для задоволення цих потреб до складу GeneHunter включена повна динамічна бібліотека функцій генетичних алгоритмів - GALIB.DLL. У бібліотеку входять функції створення популяції, визначення параметрів еволюції (таких як вірогідність схрещування, мутації, різноманітності), визначення значень цільової функції індивідуумів, оновлення популяції і переходу в наступне покоління. Користувач має можливість створювати індивідуумів з безперервними або перерахуємими хромосомами (рис. 5.19).

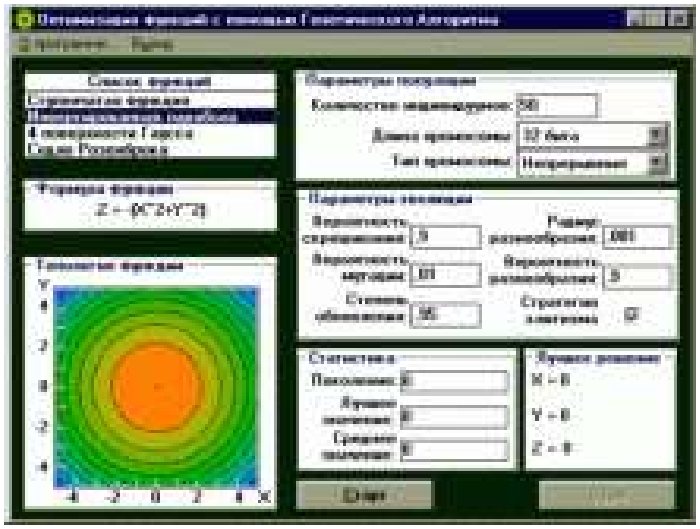


Рис. 5.19. Діалогове вікно динамічної бібліотеки.

Динамічна бібліотека GALIB.DLL дозволяє створювати додатки, в яких можуть розвиватися одночасно до 128 популяцій. Наприклад, функція MakeChromosomePool дає можливість швидко створювати множину схожих між собою хромосом, що буває зручно для таких застосувань, як оптимізація вагів нейронної мережі. GeneHunter дозволяє використовувати навіть суміш безперервних та перерахуємих хромосом в одній популяції.

Приклади використання GeneHunter. До складу пакету GeneHunter входять ряд прикладів використання пакету з робочих листів Microsoft Excel і із зовнішніх програм. Зупинимося коротко на деяких з них.

1. Аналіз портфеля акцій.

«Портфель акцій» є характерним прикладом множини задач, які вимагають групування різних по своїм властивостях об'єктів оптимальним способом. У даному прикладі фінансовий менеджер хоче розподілити наявні акції різної вартості на шість груп так, щоб вартість кожної групи складала заданий відсоток від вартості всього портфеля.

2. Оптимізація інвестиційного портфеля.

Цей приклад має більше відношення до задач, що реально виникають в житті. У реальних задачах оптимізації портфеля трейдер або керівник фондами старається мінімізувати ризик, одночасно намагаючись максимізувати дохід. Він може зробити це, намагаючись отримати портфель, вартість якого моделює поведінку вартості існуючого оптимального портфеля, такого, як S&P 500. Трейдер вибирає деяку кількість акцій, наприклад, ті, які він вважає найбільш прибутковими. ГА використовується для мінімізації ризику, допомагаючи скласти портфель так, щоб його вартість поводилася подібно до вартості більш диверсифікованого портфеля S&P 500.

3. Прогнозування індексу NYSE.

В даному прикладі GeneHunter створює правила для прогнозування зростання індексу NYSE. Формулювання цього завдання для GeneHunter декілька відрізняється від звичайного формулювання завдання для ГА. Кожен індивідуум в популяції представляє правило наступного вигляду: «якщо індекс close 4 дні тому назад менше, ніж індекс high 9 днів тому назад і якщо індекс high 3 дні тому назад більше, ніж індекс close 4 дні тому назад, тоді індекс close завтра зросте» (рис. 5.20). Цільова функція в цьому випадку має бути доходом, який можна отримати в разі використання даного правила при торгівлі. Концепція використання генетичних алгоритмів для пошуку правил може бути

поширена на задачі пошуку оптимальних правил для обробки даних, складання розкладів руху літаків або доставки вантажів і так далі.

Рис. 5.20. Прогнозування індексу NYSE.

4. Задача комівояжера.

Задача комівояжера - це широко відома задача, що стало тестом для перевірки і порівняння різних алгоритмів рішення комбінаторних задач оптимізації. Комівояжер повинен зробити замкнутий маршрут через задану кількість міст. Всі міста зв'язані між собою дорогами, і кожне місто комівояжер повинен відвідати лише один раз. GeneHunter вирішує цю задачу, вибираючи порядок відвідин міст і мінімізуючи довжину маршруту.

5. Створення оптимального графіка робіт.

Одним з найбільш важливих вживань генетичних алгоритмів є їх використання для складання оптимального розкладу для задач практично будь-яких розумних розмірів. ГА дають прекрасну можливість підприємцям планувати доручення для своїх співробітників, а технологам складати розклад технологічних процесів в конкретних умовах виробництва.

У даному прикладі ми представимо себе в ролі керуючого невеликим заводом по виробництву печатних плат, який щодня здійснює збірку, монтаж, паяння і тестування деякої кількості печатних плат різних типів. Кожна з плат повинна пройти через п'ять робочих станцій, проте на кожній з них можуть працювати по декілька фахівців (або верстатів), кожен з яких в змозі незалежно від інших повністю виконувати всі роботи, що входять в обов'язки даної

робочої станції. На заводі виготовляються декілька типів печатних плат, і кожна з них вимагає свого часу завантаження для кожної робочої станції. Тому є небезпека простою фахівців або верстатів, оскільки плати можуть зажадати довгого часу обробки на попередніх робочих станціях. GeneHunter створює розклад, що дозволяє виробити всі плати за мінімальний час.

6. Генетичне тренування нейронної мережі.

Штучні нейроні мережі спочатку були розроблені для моделювання здатності мозку розпізнавати образи. В даний час вони широко застосовуються для вирішення багатьох практичних задач передбачення і класифікації. Існують багато способів тренування нейромереж, і одним з них є використання генетичних алгоритмів. Приклад «Нейрона мережа» демонструє, як це можна зробити. У даному прикладі показаний один із способів, як можна використовувати GeneHunter для створення і тренування нейронної мережі, не застосовуючи жодних інших нейромережевих програм і алгоритмів (рис. 5.21).

Рис. 5.21. Генетичне тренування нейронної мережі.

7. Поліноміальна апроксимація функціональних залежностей.

GeneHunter можна використовувати для пошуку математичних моделей даних, багато в чому подібних до моделей, які можуть бути побудовані за допомогою нелінійного регресійного аналізу. В даному прикладі GeneHunter використовується для знаходження коефіцієнтів при п'яти незалежних змінних, а також показника міри при кожній змінній. Приклад легко доопрацювати для побудови складніших математичних моделей.

Auto2Fit 3. Програмний комплекс є потужним і в той же час легким у використанні інструментом, призначеним для вирішення оптимізаційних задач і проведення складних розрахунків. Програма Auto2Fit має в своєму розпорядженні вісім оптимізаційних алгоритмів, включаючи генетичні, і їх багаточисельні модифікації. Якщо говорити про ГА, то пакет дозволяє кодувати рішення для більшої ефективності, має шість типів операторів схрещування (кросинговера) і сім можливих варіантів відбору. Програма може працювати в одному з наступних режимів: швидкому і програмному. З додаткових можливостей Auto2Fit можна відзначити зручну навігацію по файлах, що реалізовується за допомогою дерева каталогів, робочу область, представлену у вигляді таблиць Excel, побудова 3D-графиков і інше. У комплект також включені багаточисельні приклади, у тому числі і класичні приклади задач оптимізації.

GeneBase. Бібліотека компонент для Delphi, що реалізовує генетичні алгоритми - потужний засіб рішення задач багатовимірної непараметричної оптимізації. У наборі є компонент, що реалізовує генетичний алгоритм, який може бути використаний для рішення задач багатовимірної непараметричної оптимізації. Зокрема він дозволяє знаходити субоптимальне рішення NP-повних задач (рис. 5.22).

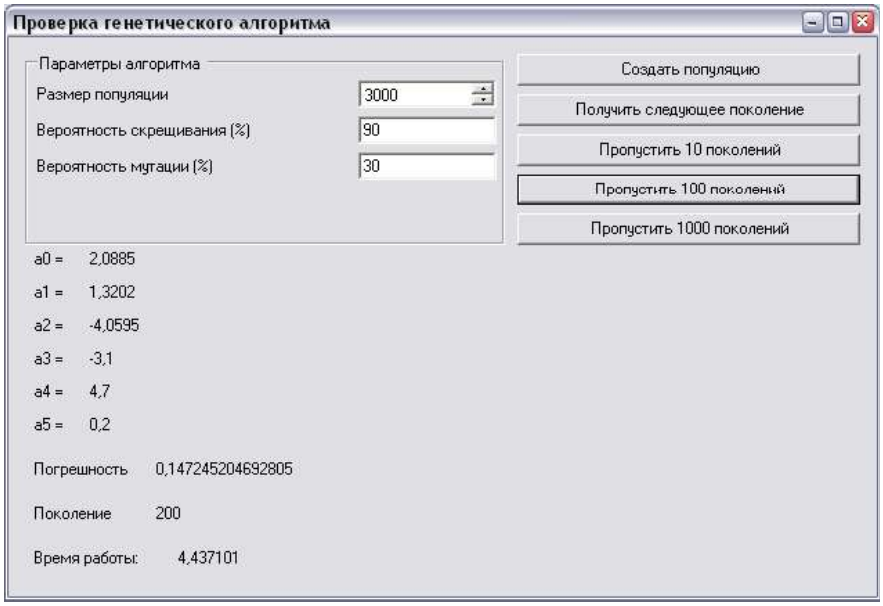


Рис. 5.22. Реалізація генетичного алгоритму на основі GeneBase.

Genetic Algorithm and Direct Search Toolbox. Genetic Algorithm and Direct Search Toolbox призначений для розширення функціональних можливостей пакету MATLAB і, зокрема, Optimization Toolbox новим виглядом алгоритмів оптимізації. У даному інструментарії містяться нові можливості по вживанню відомих алгоритмів оптимізації для такого класу задач, який представляє певні труднощі при вирішенні звичайними методами оптимізації. Подібні методи і алгоритми найчастіше використовуються у разі, коли шукана цільова функція є переривистою, істотно нелінійною, стохастичною і не має похідних або ці похідні є недостатньо визначеними. Цей модуль може розглядатися і як деяке доповнення до інших методів оптимізації як деякий засіб для пошуку прийнятної початкової точки розрахунку по основному алгоритму оптимізації. Алгоритм може служити і як засіб для подальшого уточнення раніше використаних основних алгоритмів (рис. 5.23).

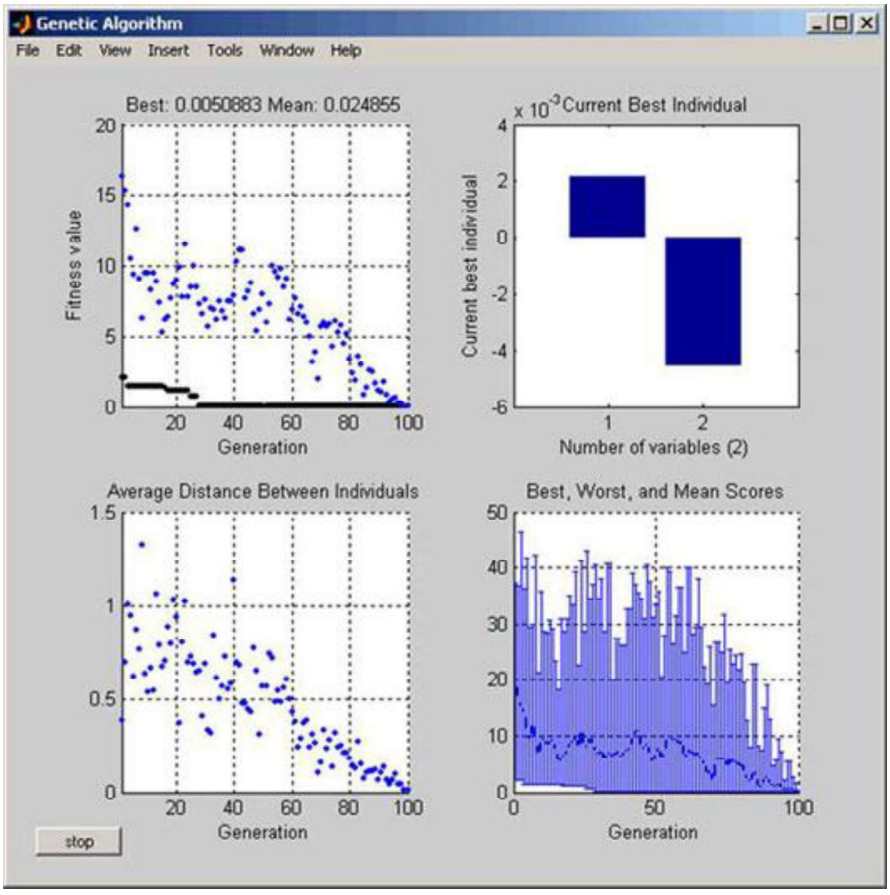


Рис. 5.23. Реалізація генетичного алгоритму засобами MATLAB.

Звернення до функцій Genetic Algorithm and Direct Search Toolbox можливо за допомогою основного графічного інтерфейсу або через командний рядок MATLAB з використанням відкритої алгоритмічної мови. Також є відповідні інструментарії для управління процесом оптимізації і контролю ефективності виконання і введення критеріїв останову виконання програми. Під час виконання процесу оптимізації з метою уточнення рішення і коректування отримуваних результатів є можливість оперативної зміни опцій виконуваного завдання. Такий підхід означає, що будь-який користувач має можливість відстежування алгоритму, модифікації вихідних кодів і створення власних оригінальних програм.

Алгоритм допоможе вирішити задачі, які або неможливо вирішити звичайними методами або виникають нестійкі рішення при вживанні стандартних математичних методів. Графічний інтерфейс користувача допоможе швидко встановити тип вирішуваної задачі і отримати її рішення (рис. 5.24).

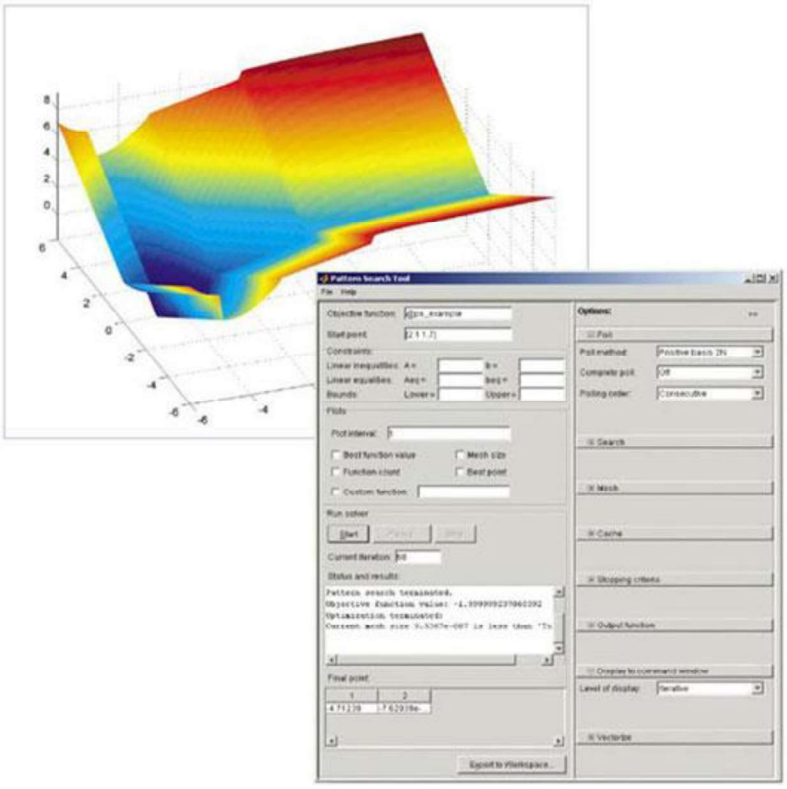


Рис. 5.24. Графічний інтерфейс Genetic Algorithm.

Genetic Algorithm and Direct Search Toolbox тісно інтегрований з пакетом MATLAB і Optimization Toolbox. Така інтеграція дозволяє використовувати генетичний алгоритм і методи безпосереднього пошуку для визначення найкращої стартової точки і, відповідно, більш повно використовувати можливості вирішувачів Optimization Toolbox або програм пакету MATLAB для підвищення точності вирішення задач оптимізації (рис. 5.25).

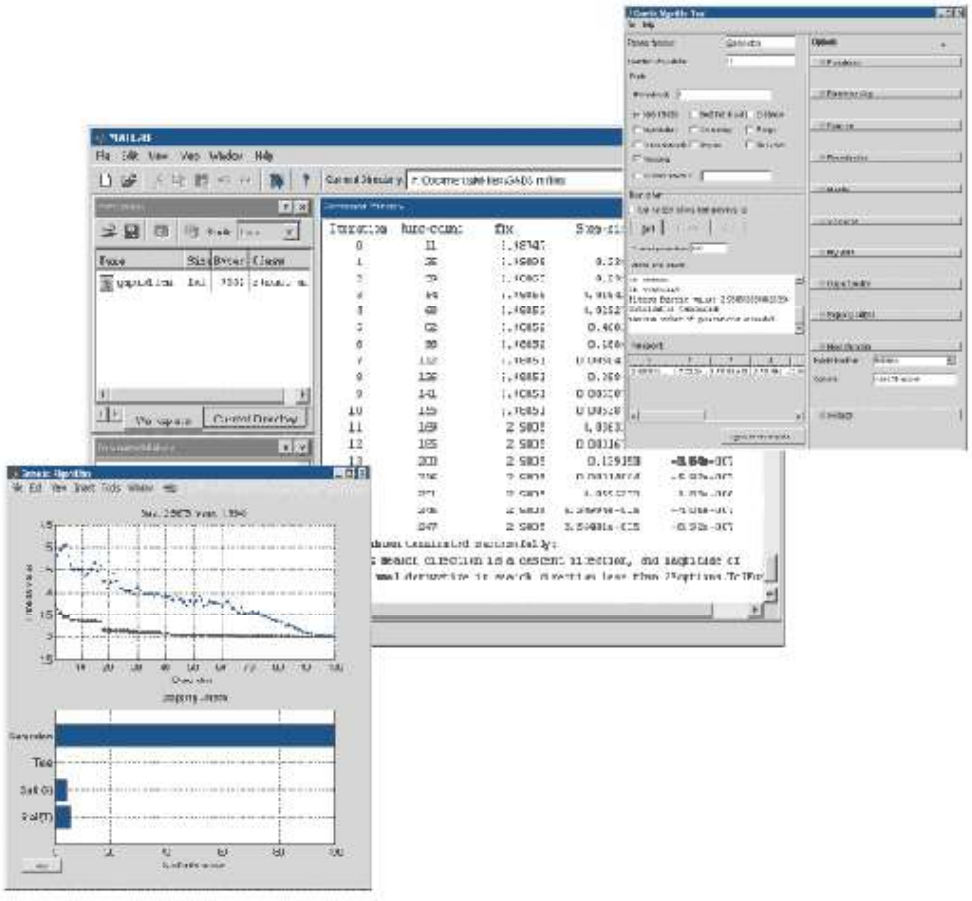


Рис. 5.25. Інтеграція генетичного алгоритму з іншими програмами MATLAB.

У даний інструмент включений ряд графічних функцій для відображення результатів оптимізації. Подібна візуалізація дає можливість встановити динамічний зворотній зв'язок з процесом оптимізації і проводити необхідні модифікації під час виконання програм. Спеціалізовані графічні функції є як для генетичного алгоритму, так і для методу прямого пошуку. Крім того, є можливість поєднання графіків, виділення окремих графіків для

ретельнішого аналізу і введення власних графічних відображень користувача. Так само є можливість експорту алгоритмічних опцій виконаних задач з подальшим їх імпортом в графічний інтерфейс.

Апаратна реалізація генетичних алгоритмів. В даний час спостерігається зростаючий інтерес до вживання генетичних алгоритмів в автономних системах і розвитку нового напряму в області проектування апаратних засобів, який отримав назву еволюційні апаратні засоби. Вперше ця ідея була запропонована С. Луїсом [Louis] і Д. Раулінсом [Rawlins] в 1991 році та експериментально перевірена в області цифрових схем. Надалі, запропоновані методи були розвинені і доопрацьовані, і отримали вживання в багатьох автоматизованих системах проектування апаратури. Зараз бурхливо розвивається один з найбільш перспективних напрямів в даної області, пов'язаний з проектуванням самореконфігуруємих апаратних засобів і автоматичного проектування схем, реалізації динамічної реконфігурації в мобільних системах, побудови реконфігуруємих апаратних засобів на одному кристалі БІС. Використання генетичних алгоритмів як механізму для автоматичного проектування схем на реконфігуруємих платформах, отримало назву еволюційні апаратні засоби (Evolvable Hardware), яке також використовується синонімом для загального напряму, відомого як еволюційна електроніка (Evolutionary Electronics).

Новий напрям використання генетичних алгоритмів - побудова динамічно реконфігуруємих апаратних засобів, в яких виробляється еволюційна зміна архітектури системи в режимі реального часу відповідно до зміни зовнішніх чинників. При реалізації подібних систем, ГА як правило виступає як зовнішній апаратний модуль або вбудовується в один кристал з реконфігурованою апаратною системою.

Апаратна реалізація компактного генетичного алгоритму передбачає перехід від програмної реалізації компактного ГА до апаратного виконання на прикладі реалізації алгоритму на FPGA фірми Xilinx. В результаті, апаратно реалізований компактний ГА виконує одну генерацію за три тактові цикли для

задачі one-max. Даний алгоритм був також розглянутий Джоном Галлагхером і доопрацьований для вживання в еволюційних апаратних засобах, на прикладі використання алгоритму в схемі управління, в якому реконфігурована аналогова нейронна мережа змінюється в інтерактивному режимі, для управління фізичними процесами. У структуру компактного ГА були додані деякі генетичні оператори, наприклад стратегія елітизма, оператор мутації і схема перевиборки кращого рішення (champion resampling), а також були внесені зміни в структуру алгоритму, що дозволило добитися здобуття якісно нових результатів при тестуванні алгоритмами ДеДжонга [DeJong]. Основний ухил при апаратній реалізації робився в області зменшення споживаної потужності і необхідного простору кристала для розміщення алгоритму.

Іншим рішенням підвищення ефективності генетичних алгоритмів і сфери їх застосування є апаратна реалізація генетичного алгоритму UMDA (Univariate Marginal Distributional Algorithm) (рис. 5.26).

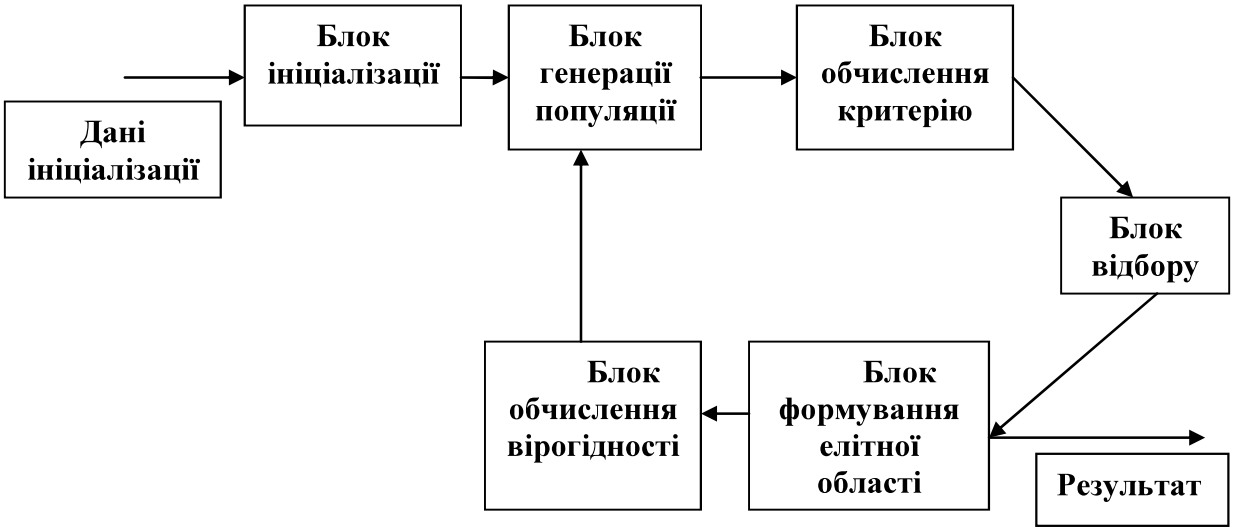


Рис.5.26. Структурна схема функціонування імовірнісного генетичного алгоритму UMDA.

Функціонування алгоритму полягає в послідовному виконанні генетичних операторів. При ініціалізації виробляється налаштування параметрів функціонування алгоритму. Блок генерації популяції відповідає за формування

популяції за допомогою послідовної генерації особин (хромосом). Для кожної хромосоми виробляється обчислення критерію відбору (функції придатності), на основі якого виконується відбір кращих особин і формування елітної області. На основі сформованої елітної області виробляється обчислення вірогідності, необхідної для генерації наступного покоління. Ознакою знаходження рішення є рішення задачі one-max або виконання заданої кількості ітераційних циклів.

Розглянемо деякі найбільш значущі сфери практичного застосування генетичних алгоритмів.

Задача комівояжера. Задача комівояжера в теорії дискретної оптимізації вважається класичною задачею. Вперше вона була сформульована ще в 1759 році. Суть задачі полягає в тому, аби знайти найкоротшу замкнуту дорогу обходу декількох міст, заданих своїми координатами (або за допомогою матриці відстаней між ними). Міста можуть відвідуватися лише один раз. Відомо, що вже для 50 міст пошук оптимальної дороги є складним завданням, що спонукало розвиток різних нових методів (у тому числі нейромережевих і генетичних алгоритмів). Це задача NP-повна (задача з експоненціальною оцінкою числа ітерацій, необхідних для відшукання точного рішення) і мультимодальна (має локальні екстремуми). ГА використовується для знаходження околооптимального шляху за лінійний час.

Функція пристосованості. Значення функції пристосованості повинне відповідати відстані, яку проходить комівояжер згідно шляху, що представляється хромосомою. Оскільки це значення має бути мінімальне, то кінцева формула функції пристосованості j хромосоми часто виглядає таким чином: $f_j = 1.1 \cdot d_{\max} - d_j$, де $d_{\max}$ - довжина максимального маршруту в поточній популяції, d_j - довжина маршруту, що представляється j хромосомою. Значення цієї функції чим більше, тим краще.

Кодування рішень. В даний час існує чотири основні представлення маршруту комівояжера у вигляді хромосоми: сусідське, порядкове, шляхове і матричне. Оскільки класичні оператори схрещування і мутації для них, як

правило, непридатні, кожне з цих представлень має власні «генетичні» оператори, всі вони дуже сильно розрізняються. Деякі з розглянутих нижче кросоверів можуть створювати потомство, що не має рішення (невалідні рішення). Невалідні рішення можуть бути корисні для створення і внесення різноманітності в популяцію, проте цим вони можуть уповільнити або навіть запобігти збіжності ГА. Одним з рішень цієї проблеми може стати ідея відновлення невалідних рішень.

Сусідське представлення. Сусідське представлення представляє маршрут як список з n міст. Місто j знаходиться на позиції i лише в тому випадку, якщо маршрут проходить з міста i в місто j. Наприклад, маршрут з наступним порядком обходу міст: $1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7 \to 1$ представляється як хромосома $P = [2\ 4\ 8\ 3\ 9\ 7\ 1\ 5\ 6]$. Кожен маршрут має лише одне сусідське представлення, але не кожен вектор в сусідському представленні відповідає якому-небудь маршруту. Наприклад, вектор [2 4 8 1 9 3 5 7 6] позначає послідовність $1 \to 2 \to 4 \to 1...$, тобто частина маршруту - це замкнутий цикл, що недопустимо.

Сусідське представлення не підтримує класичну операцію кросовера (він практично завжди породжує недопустимі маршрути). Для нього розроблено три власні оператора.

Кросовер альтернативних ребер. Кросовер alternating edges будує нащадків, вибираючи по черзі перше ребро від першого батька, потім друге ребро від другого, потім знову наступне від першого і так далі. Якщо нове ребро представляє замкнутий цикл, беруть ребро з того ж батька (порушуючи чередування). Якщо і воно утворює цикл, беруть випадкове ребро, яке ще не вибиралося і не утворює замкнутого циклу. Другий нащадок будується аналогічно, але перше ребро беруть від другого батька. Для прикладу розглянемо побудову одного з нащадків двох хромосом:

$$P_1 = [2\ 3\ 8\ 7\ 9\ 1\ 4\ 5\ 6],$$

$$P_2 = [7\ 5\ 1\ 6\ 9\ 2\ 8\ 4\ 3].$$

Крок 1. П1 = [2 _ _ _ _ _ _ _ _]. (У маршрут входить ребро $1 \to 2$).

Крок 2. П1 = [2 5 _ _ _ _ _ _]. (Маршрут містить фрагмент $1 \to 2 \to 5$).

Крок 3. П1 = [2 5 8 _ _ _ _ _]. (Маршрут містить фрагменти $1 \to 2 \to 5,3 \to 8$).

Крок 4. П1 = [2 5 8 6 _ _ _ _ _]. (Маршрут містить фрагменти $1 \to 2 \to 5,3 \to 8,4 \to 6$).

Крок 4. П1 = [2 5 8 6 9 _ _ _ _]. (Маршрут містить фрагменти $1 \to 2 \to 5 \to 9,3 \to 8,4 \to 6$).

Крок 5. П1 = [2 5 8 6 9 1 _ _ _]. (Маршрут містить фрагменти $4 \to 6 \to 1 \to 2 \to 5 \to 9,3 \to 8$).

Крок 6. П1 = [2 5 8 6 9 1 4 _ _]. (Маршрут містить фрагменти $7 \to 4 \to 6 \to 1 \to 2 \to 5 \to 9,3 \to 8$).

Крок 7. П1 = [2 5 8 6 9 1 4 7 _]. (Маршрут містить фрагмент $3 \to 8 \to 7 \to 4 \to 6 \to 1 \to 2 \to 5 \to 9$). На цьому етапі нащадок отримав ребро $8 \to 7$, не присутнє ні у кого з батьків, оскільки додавання в маршрут 5-го або 4-го батьківських міст привело б до утворення замкнутого підциклу.

Крок 8. П1 = [2 5 8 6 9 1 4 7 3]. (Маршрут має вигляд $3 \to 8 \to 7 \to 4 \to 6 \to 1 \to 2 \to 5 \to 9 \to 3$).

Кросовер фрагментів. Кросовер subtour chunks створює нащадків, вибираючи (випадково) фрагмент від одного з батьків, потім випадкової довжини фрагмент від іншого з батьків, і так далі. І знову, якщо на якомусь рівні утворюється замкнутий цикл, він вирішується аналогічним чином (вибирається інший випадковий фрагмент, який ще не вибирався).

Евристичний кросовер. Heuristic crossovers будує нащадків, вибираючи випадкове місто як стартову точку для маршруту - нащадка. Потім він порівнює два відповідні ребра від кожного з батьків і вибирає більше коротке. Потім кінцеве місто вибирається як початкове для вибору наступного коротшого ребра з цього міста. Якщо на якомусь кроці виходить замкнутий маршрут, він продовжується будь-яким випадковим містом, яке ще не відвідувалося.

Переваги сусідського представлення в тому, що воно дозволяє схематично аналізувати створювані маршрути. Це представлення має в основі

«будівельні блоки» – зв'язки між містами. Наприклад, схема (* * * 3 * 7 * * *) описує множину всіх маршрутів з ребрами $(4 \to 3)$ і $(6 \to 7)$. Основний же недолік даного представлення - в тому, що множина всіх його операцій дуже бідна. Зокрема, для нього не існує простих алгоритмів мутації. Кросовер alternating edges часто руйнує хороші маршрути, які були у обох батьків до вживання цієї операції. Кросовер subtour chunks має кращі характеристики, чим перший, завдяки тому, що його руйнівні властивості менші. Але все одно його експлуатаційні якості все ж досить низькі. Кросовер heuristic crossover, звичайно ж, найкращий оператор для даного представлення. Причина в тому, що попередні операції сліпі, вони не беруть в розрахунок справжню довжину ребер. З іншого боку, heuristic crossover вибирає краще ребро з двох можливих. Може вийти, що подальша дорога буде неможлива і доведеться вибирати ребро, довжина якого невиправдано велика.

Порядкове представлення. Порядкове представлення визначає маршрут як список з n міст; i елемент списку - номер від 1 до $n - i - 1$. Ідея порядкового представлення полягає в наступному. Є впорядкований список міст, який служить для зв'язку маршрутів з їх порядковим представленням. Передбачимо, що такий впорядкований список простий: $C = [1\ 2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$. Тоді маршрут $1 \to 2 \to 4 \to 3 \to 8 \to 5 \to 9 \to 6 \to 7 \to 1$ буде представлений як список $L = [1\ 1\ 2\ 1\ 4\ 1\ 3\ 1\ 1]$. Він може бути інтерпретований таким чином.

- Оскільки перший номер списку L дорівнює 1, беремо перше місто із списку C як перше місто маршруту (місто номер 1) і виключаємо його із списку. Поточний фрагмент маршруту - це 1.
- Наступний номер в списку L також 1, тому беремо перший номер з списку $C = [2\ 3\ 4\ 5\ 6\ 7\ 8\ 9]$, що залишився. Оскільки ми виключили із списку C 1-е місто, наступне місто - 2. Виключаємо і це місто із списку. Маршрут на даному кроці набуває вигляду: $1 \to 2$.
- Наступний номер в списку $L - 2$. Беремо із списку $C = [3\ 4\ 5\ 6\ 7\ 8\ 9]$ 2-ге по порядку місто, що залишилося. Це - 4. Виключаємо його із списку. Маршрут $1 \to 2 \to 4$.

- Наступний номер в списку $L-1$. Беремо із списку $C=[\ 3\ 5\ 6\ 7\ 8\ 9]$ 1-е місто - з номером 3. Маємо маршрут $1\to 2\to 4\to 3$.
- Наступний номер в списку $L-4$, беремо 4-е місто із списку C, що залишився =[5 6 7 8 9] - це 8. Маршрут $1\to 2\to 4\to 3\to 8$.
- Наступний номер в списку $L-1$. Беремо із списку $C=[\ 5\ 6\ 7\ 9]$ 1 -е місто - з номером 5. Маршрут $1\to 2\to 4\to 3\to 8\to 5$.
- Наступний номер в списку $L-3$. Беремо третє місто із списку $C=[\ 6\ 7\ 9]$ - це 9. Видаляємо його з C. Маршрут $1\to 2\to 4\to 3\to 8\to 5\to 9$.
- Наступний номер в списку $L-1$, тому беремо перше місто з поточного списку $C=[\ 6\ 7]$ - місто номер 6, і видаляємо його з C. Частинний маршрут має вигляд $1\to 2\to 4\to 3\to 8\to 5\to 9\to 6$.
- Останнім номером в списку L завжди буде 1, тому беремо останнє місто, що залишилося, з поточного списку $C=[7]$, і видаляємо його з C. Остаточний маршрут має вигляд $1\to 2\to 4\to 3\to 8\to 5\to 9\to 6\to 7\to 1$.

Основна перевага порядкового представлення в тому, що класичний кросовер в даному випадку працює. Будь-які два маршрути в порядковому представленні, що розрізаються в будь-якій позиції і склеєні разом, породять двох нащадків, кожен з яких буде правильним маршрутом. Наприклад, два батька $P_1=[1\ 1\ 2\ 1|\ 4\ 1\ 3\ 1\ 1]$ та $P_2=[5\ 1\ 5\ 5\ |\ 5\ 3\ 3\ 2\ 1]$, які позначають відповідно маршрути $1\to 2\to 4\to 3\to 8\to 5\to 9\to 6\to 7\to 1$ та $5\to 1\to 7\to 8\to 9\to 4\to 6\to 3\to 2\to 5$, з точкою розрізу, позначеною « | » породять наступних нащадків: П1 = [1 1 2 1 5 3 3 2 1] та П2 = [5 1 5 5 4 1 3 1 1]. Ці нащадки позначають маршрути $1\to 2\to 4\to 3\to 9\to 7\to 8\to 6\to 5\to 1$ та $5\to 1\to 7\to 8\to 6\to 2\to 9\to 3\to 4\to 5$. Слід відмітити, що частини маршруту зліва від лінії розрізу не змінилися, тоді як частини маршруту праворуч від лінії розрізу мають досить багато відмінностей від закінчень батьківських хромосом. В якості мутації використовується зміна з вірогідністю p_m i елементу порядкового представлення на випадковий номер від 1 до $n-i-1$.

Шляхове представлення. Шляхове представлення - це найбільш природне представлення маршруту. Наприклад, тур

$5 \to 1 \to 7 \to 8 \to 9 \to 4 \to 6 \to 2 \to 3 \to 5$ буде представлений як рядок [5 1 7 8 9 4 6 2 3].

Частково відображаємий кросовер. Кросовер, що частково відображується, бере спочатку частину дороги одного батька і зберігає послідовність і позиції як можна більшого числа міст іншого батька. Точка кросовера вибирається випадково (рис. 5.27).

$$V_1 = 12 \mid 543$$
$$V_2 = 35 \mid 421$$

Рис. 5.27. Точка перетину частково відображаємого кросовера.

Потім виконується відображення замін перших частин хромосом $\{1 \Leftrightarrow 3, 2 \Leftrightarrow 5\}$, яке застосовується поточково до батьків для здобуття потомства. Перевіряється кожен елемент першого батька: якщо є для нього заміна, то вона виконується і потім копіюється ($\Updownarrow$) в першого нащадка, інакше він просто копіюється без заміни ($\Downarrow$) (рис. 5.28). Подібна процедура може бути використана і для другого нащадка (рис. 5.29, 5.30, 5.31).

$$V_1 = 1\ 2\ 5\ 4\ 3$$
$$\Updownarrow\ \Updownarrow\ \Updownarrow\ \Downarrow\ \Updownarrow$$
$$V_6 = 3\ 5\ 2\ 4\ 1$$

Рис. 5.28. Частково відображаємий кросовер.

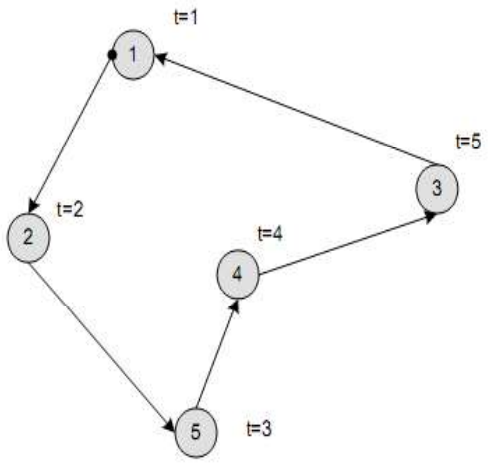


Рис. 5.29. Батько V_1.

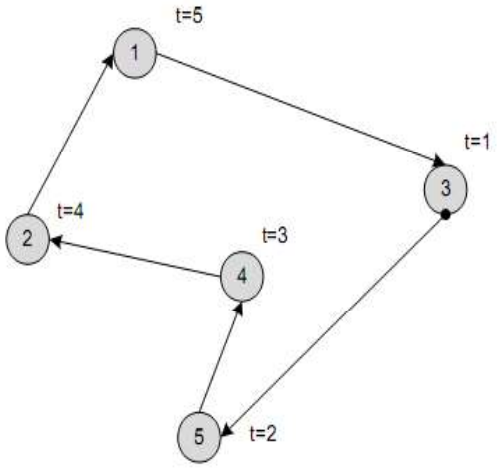


Рис. 5.30. Батько V_2.

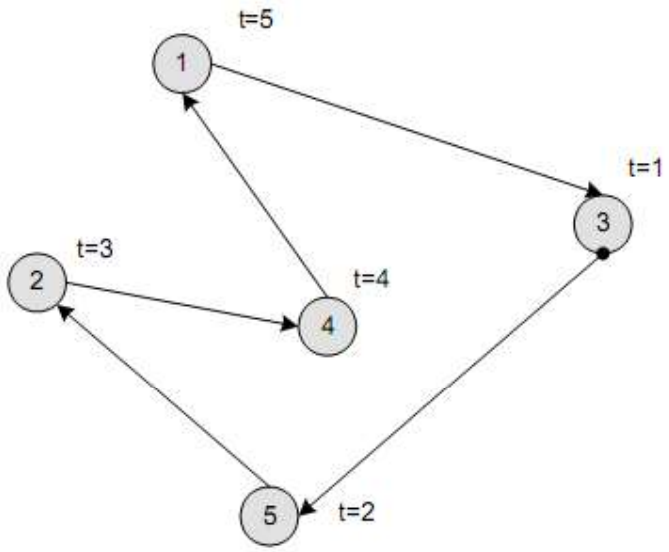


Рис. 5.31. Нащадок V_6.

Впорядкований кросовер. Впорядкований кросовер бере частину шляху одного батька і зберігає родинний порядок міст з іншого батька. Перші дві точки перетину кросовера вибираються випадково (рис. 5.32). Кожен елемент центральної секції першого батька копіюється в нащадка (рис. 5.33).

$V_1 = 1 \mid 2\ 5 \mid 43$

$V_2 = 3 \mid 54 \mid 21$

Рис. 5.32. Точки перетину кросовера.

$V_1 = 1\ 2\ 5\ 4\ 3$

$V_7 = \quad 2\ 5$

Рис. 5.33. Створення нового нащадка.

Потім елементи другого батька збираються в список (рис. 5.34), починаючи з другої точки кросовера. Нарешті, міста, вже представлені в нащадках, видаляються (рис. 5.35), і елементи, що залишилися, копіюються замість порожніх пропусків нащадка, починаючи з другої точки перетину кросовера (мал. 5.36).

[2 1 3 5 4]

Рис. 5.34. Список міст в 2 батьку.

[2 1 3 5 4] => [1 3 4]

Рис. 5.35. Видалення дубльованих міст.

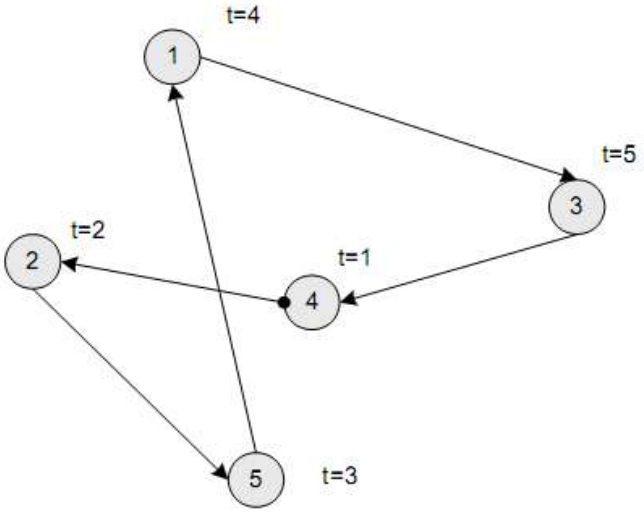


Рис. 5.36. Нащадок $V_7 = [4\ 2\ 5\ 1\ 3]$.

Кросовер рекомбінації ребер. Кросовер рекомбінації ребер робить потомство лише за допомогою ребер, представлених в обох батьках. Спочатку з двох батьків $V_1 = [1\ 2\ 5\ 4\ 3]$ та $V_2 = [3\ 5\ 4\ 2\ 1]$ будується список ребер (табл. 1). Міста в нащадках вибираються поодинці, при цьому вибирається місто з найменшою кількістю ребер. Першим елементом в хромосомі є місто з маленькою кількістю зв'язків. Після того, як місто вибране, воно видаляється з таблиці, і міста, приєднані до нього, розглядаються як кандидати при наступному виборі (рис. 5.37).

Табл. 1.

Список ребер.

Місто	Сполучений з		
1	2	3	
2	1	5	4
3	4	1	5
4	5	3	2
5	2	4	3

Хромосома на першому кроці - $V_9 = [1\ \_\ \_\ \_\ \_\ ]$ (табл. 2).

Табл. 2.

Список ребер після першого кроку.

Місто	Сполучений з		
2	5	4	
3	4	5	
4	5	3	2
5	2	4	3

Хромосома на другому кроці - $V_9 = [1\ 3\ \_\ \_\ \_\ ]$ (табл. 3).

Табл. 3.

Список ребер після другого кроку.

Місто	Сполучений з	
2	5	4
4	5	3
5	2	4

Хромосома на третьому кроці - $V_9 = [1\ 3\ 5\ \_\ \_\ ]$ (табл. 4).

Табл. 4.

Список ребер після третього кроку.

Місто	Сполучений з
2	4
4	2

Хромосома на четвертому кроці - $V_9 = [1\ 3\ 5\ 2\ \_\ ]$ (табл. 5).

Табл. 5.

Список ребер після четвертого кроку.

Місто	Сполучений з
4	

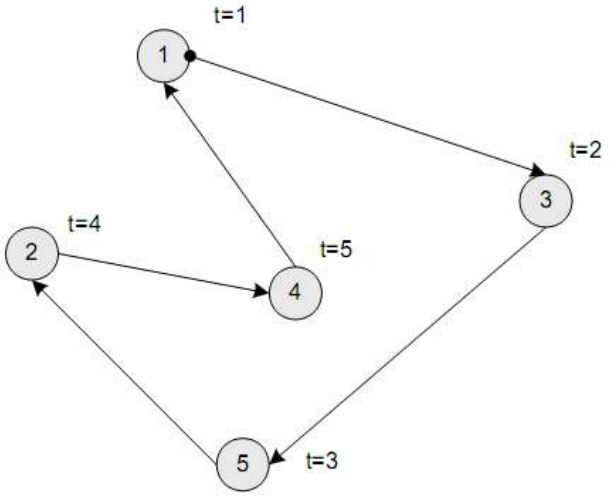


Рис. 5.37. Отриманий нащадок $V_9 = [1\ 3\ 5\ 2\ 4]$.

Мутація. Мутація працює за наступним принципом. Вибираються випадковим чином два міста в хромосомі і міняються місцями.

«Жадібна мутація». «Жадібна мутація» (greedy reconnection) полягає у впорядковуванні послідовності міст. Її вживання допомагає добре шукати локальний оптимум. Спочатку випадковим чином вибираються дві точки перетину в хромосомі так, щоб між ними було не менше чим 2 міста. Потім послідовність міст між точками перетину упорядковується: вони переставляються залежно від близькості один до одного. У нашому випадку серед міст 2, 5 і 4 визначається місто, найближче до міста 1. Хай, наприклад, це місто 5. Він ставиться слідом за містом 1. Потім серед міст 2 і 4 визначається найближчий до міста 5. Хай, наприклад, це місто 4. Ставимо його за містом 5. На останнє місце ставимо місто, що залишилося, 2. В результаті маємо модифіковану хромосому.

Крок 1. $V_1 = [1 \mid 2\ 5\ 4 \mid 3]$, $V_1^* = [1\ \_\ \_\ \_\ 3]$.

Крок 2. $V_1^* = [1\ 5\ \_\ \_\ 3]$.

Крок 3. $V_1^* = [1\ 5\ 4\ \_\ 3]$.

Нащадок, отриманий при мутації: $V_1^* = [1\ 5\ 4\ 2\ 3]$.

Матричне представлення. Для кодування хромосоми також може служити бінарна матриця V, де $V_{ct} = 1$, якщо місто c відвідане у момент часу t

(знаходиться в маршруті у позиції t), інакше $V_{ct} = 0$. Наприклад, шляхи ABEDC (V_1) і CEDBA (V_2) можуть бути представлені у вигляді наступних матриць (рядки-міста, стовпці-час) (рис. 5.38):

$$V_1 = \begin{vmatrix} & 1 & 2 & 3 & 4 & 5 \\ A & 1 & 0 & 0 & 0 & 0 \\ B & 0 & 1 & 0 & 0 & 0 \\ C & 0 & 0 & 0 & 0 & 1 \\ D & 0 & 0 & 0 & 1 & 0 \\ E & 0 & 0 & 1 & 0 & 0 \end{vmatrix} \qquad V_2 = \begin{vmatrix} & 1 & 2 & 3 & 4 & 5 \\ A & 0 & 0 & 0 & 0 & 1 \\ B & 0 & 0 & 0 & 1 & 0 \\ C & 1 & 0 & 0 & 0 & 0 \\ D & 0 & 0 & 1 & 0 & 0 \\ E & 0 & 1 & 0 & 0 & 0 \end{vmatrix}$$

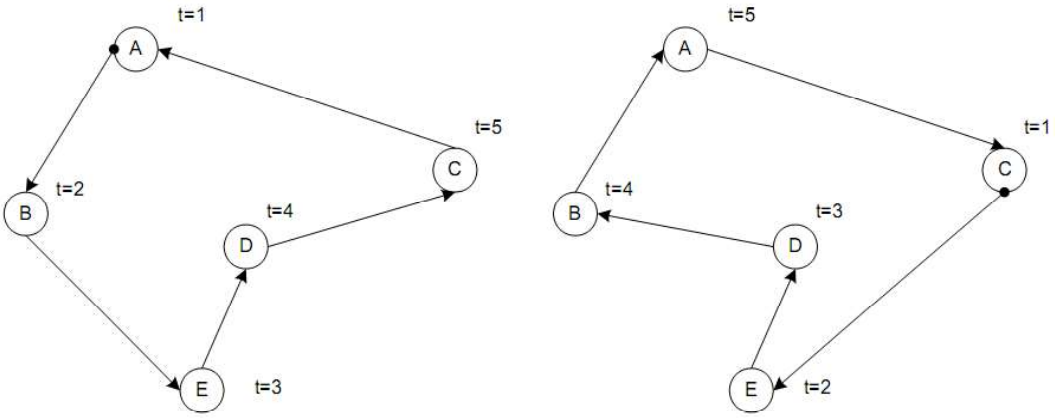


Рис. 5.38. Шляхи ABEDC та CEDBA.

Матриці можуть бути представлені у вигляді наступних хромосом:

$$V_1 = 100000100000010001000100$$

$$V_2 = 000010001010000010001000$$

Необхідно відзначити, що міста в однаковій послідовності, але з різними стартовими точками або з протилежним напрямом кодуються різними матрицями, тобто різними хромосомами. ГА може створити деякі хромосоми, що не мають шляхового представлення (невалідні рішення). Це може статися при створенні початкової популяції і при дії стандартних генетичних операторів.

Фокс (Fox) і Макмахон (McMahon) розглядали маршрут як двійкову матрицю, в якій елемент X_{ij} містить 1, якщо місто i стоїть в маршруті раніше, ніж місто j. Існує також альтернативне шляхове представлення , при якому в

матрицю в елемент X_{ij} записується 1 лише в тому випадку, якщо i місто безпосередньо передує j. Для цих представлень також розроблені способи схрещування і мутації, але вони досить складні, тому в алгоритмах частіше застосовуються розглянуті раніше способи символьного кодування хромосом (рис. 5.39).

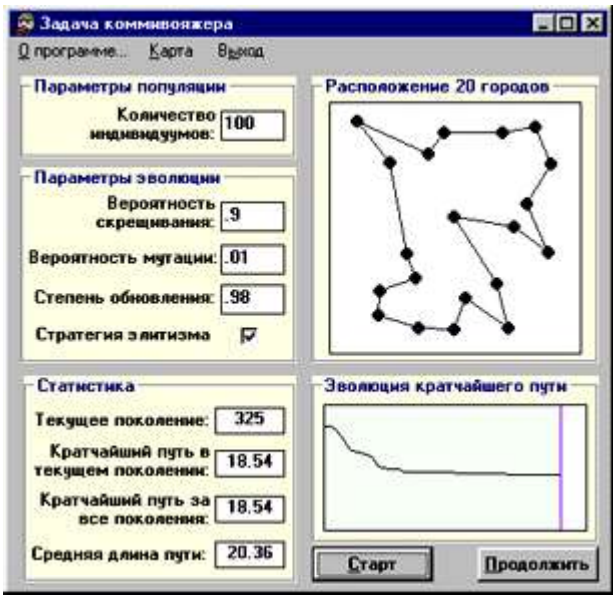


Рис. 5.39. Рішення задачі комівояжера в пакеті GeneHunter.

Аналіз ризиків проектів. В даний час підприємства, що займаються реорганізацією своєї діяльності і одночасно з цим впроваджують інформаційні системи управління, прагнуть скоротити терміни реалізації цих проектів до мінімуму, при цьому постійно корегуючи вимоги до створюваної інформаційної системи. Така ситуація приводить до багатократного зростання ризиків невдалого завершення цих проектів. Як правило, системні інтегратори, що виконують такі проекти, намагаються проводити аналіз можливих ризиків до початку, а також в ході реалізації проекту. Проте, на практиці часом складно, а деколи і неможливо визначити залежність різних процесів проекту від можливих ризиків, що діють на проект, ще складніше привести аналітичний опис такої залежності, тим більше на початкових етапах проекту. Навіть сама задача визначення ризиків, які можуть діяти на проект стає порівнянною по складності з вхідною задачею. Ці обставини значно ускладнюють вживання на

всіх етапах проекту класичних методів аналізу ризиків, оскільки більшість з них ґрунтується на використанні апріорної інформації про реакцію системи управління проектом на виникнення ризикових ситуацій в аналогічних, вже виконаних проектах. У зв'язку з цим встає задача побудови таких методів аналізу ризиків, які були б здатні виявляти ризики проекту практично при повній відсутності припущень про характер поведінки процесів проекту на різних етапах. Одними з таких методів є еволюційні методи пошуку, зокрема, генетичні алгоритми.

Генетичний алгоритм, побудований для такої задачі, володіє досить широкими можливостями. Відстроївши відповідним чином параметри системи можна управляти процесом пошуку залежно від поставленої задачі. Існує як мінімум три класи задач, які можуть бути вирішені представленим алгоритмом аналізу ризиків проекту:

- задачі визначення найбільшої вірогідності виникнення ризикової події. При рішенні цієї задачі встановлюється етап робіт проекту або стадія виконання процесу на якій існує найбільша вірогідність виникнення вказаної ризикової події;
- задачі визначення системи ризиків проекту. При рішенні цієї задачі встановлюються ризики проекту або процесів проекту, які мають найбільшу вірогідність передаватися з етапу в етап або з процесу в процес;
- задачі визначення профілів ризиків проекту або процесів проекту на всіх його етапах. При рішенні цієї задачі визначається взаємозв'язана система ризиків проекту що діє на даному етапі.

Генетичний алгоритм побудови розкладу для багатопроцесорних обчислювальних систем. Багатопроцесорна обробка, як спосіб підвищення загальної ефективності обчислень, є одним з перспективних напрямів розвитку обчислювальних систем. Задачі, які необхідно вирішувати при плануванні паралельних обчислень в загальній постановці є NP-повними. Для скорочення часу вирішення таких задач використовують евристичні підходи. Одним з

евристичних підходів до вирішення задач планування є використання генетичних алгоритмів.

Для однозначного розподілу процесів по процесорах необхідно для кожного процесу знати номер процесора, на якому він виконуватиметься. Інформацію про процес виконання програми зручно представляти у вигляді рядки наступного вигляду:

- кількість чисел в рядку (довжина рядка) відповідає кількості процесів;
- кожна позиція рядка відповідає одному процесу, порядковий номер якого дорівнює номеру позиції;
- число у позиції рядка показує, на якому процесорі виконуватиметься даний процес.

Залежно від вибраного критерію ефективності цільова функція може бути або час виконання програми або завантаження системи. Початкова популяція генерується випадковим чином, якщо передбачається побудова нового розкладу. Якщо є варіант розкладу і його необхідно поліпшити, за основу для початкової популяції береться наявний розклад. Селекція проводиться за пропорційною схемою. Число нащадків прямо пропорційно залежить від значення цільової функції для поточного рядка. Для проведення операції мутації в рядку випадковим чином вибирається позиція, число в якій замінюється вибраним випадковим чином числом з інтервалу.

Розроблений алгоритм представляє практичний інтерес при побудові планувальників багатопроцесорних систем. Дослідження алгоритму показали високу ефективність даного підходу порівняно з іншими методами. Додатковою перевагою даного методу є незалежність від структури обчислювальної системи, кількість процесорів в системі є вхідними даними для алгоритму.

5.5. Мурашині алгоритми та генетичне програмування

У останні два десятиліття при оптимізації складних систем дослідники все частіше застосовують природні механізми пошуку найкращих рішень. Ці механізми забезпечують ефективну адаптацію флори і фауни до довкілля впродовж мільйонів років. Останніми роками інтенсивно розробляється науковий напрям Natural Computing - «Природні обчислення», який об'єднує математичні методи з природними механізмами прийняття рішень, а саме:

- Genetic Algorithms - генетичні алгоритми;
- Evolution Programming - еволюційне програмування;
- Neural Network Computing – еволюційне нейромережеві обчислення;
- DNA Computing - ДНК обчислення;
- Cellular Automata - клітинні автомати;
- Ant Colony Algorithms – мурашині алгоритми.

Імітація самоорганізації мурашиної колонії складає основу мурашиних алгоритмів оптимізації - нового перспективного методу природних обчислень. Колонія мурах може розглядатися як багатоагентна система, в якій кожен агент (мурашка) функціонує автономно по дуже простих правилах. На противагу майже примітивній поведінці агентів, поведінка всієї системи виходить на подив розумною. Мурашині алгоритми серйозно досліджуються європейськими вченими з середини 90-х років. Вперше підхід був запропонований бельгійським дослідником Марко Доріго (Marco Dorigo). На сьогодні вже отримані важливі результати мурашиної оптимізації таких складних комбінаторних задач, як: задача комівояжера, задача оптимізації маршрутів вантажівок, задача розфарбовування графа, квадратичної задачі про призначення, оптимізації мережевих графіків, задача календарного планерування і інших. Особливо ефективні мурашині алгоритми при оптимізації процесів в розподілених нестаціонарних системах, наприклад трафіків в телекомунікаційних мережах [50, 78].

Кожен, хто хоч раз в житті спостерігав за мурахами, обов'язково повинен був відмітити: вся діяльність цих комах має яскраво виражене групове забарвлення. Працюючи разом, група мурах здатна затаскати в мурашник шматок їжі або будівельного матеріалу, в 10 разів більше самих працівників. Вчені давно знають про це, але лише останнім часом задумалися про корисне застосування мурав'їнного досвіду в повсякденному житті.

Сам по собі мурашка - досить примітивна істота. Всі його дії, по суті, зводяться до елементарних інстинктивних реакцій на навколишнє оточення і своїх побратимів. Проте декілька мурах разом утворюють складну систему, яку деякі вчені називають «колективним розумом». Тому алгоритми мурав'їнних колоній часто називають алгоритмами «роєвого інтелекту». Наприклад, група мурах прекрасно здатна знаходити найкоротшу дорогу до їжі. Якщо яка-небудь перешкода - палиця, камінь, нога людини - встає на дорозі, браві добувачі швидко знаходять новий оптимальний маршрут.

Принципи поведінки мурах витримали випробування впродовж 100 мільйонів років - саме стільки часу тому мурашки «колонізували» Землю. Вони відносяться до соціальних комах, що живуть усередині деякого колективу - колонії. На Землі близько двох відсотків комах є соціальними, половину з них складають мурав'ї - невеликі істоти масою від 1 до 5 міліграма. Число мурах в одній колонії вагається від 30 штук до декількох мільйонів. На Землі близько 10^{16} мурах із загальною масою, приблизно рівній масі людства.

Які ж механізми забезпечують настільки складну поведінку мурах, і що можна запозичити у цих істот для вирішення своїх глобальних задач? Основу «соціальної» поведінки мурашок складає самоорганізація - множина динамічних механізмів, що забезпечують досягнення системою глобальної мети в результаті низькорівневої взаємодії її елементів. Принциповою особливістю такої взаємодії є використання елементами системи лише локальній інформації. При цьому виключається будь-яке централізоване управління і звернення до глобального образу, що репрезентує систему на зовнішньому світі.

Мурахи використовують два способи передачі інформації: прямий - обмін їжею, мандибулярний, візуальний і хімічний контакти, і непрямий - стігмержи (stigmergy). Стігмержи - це рознесений в часі тип взаємодії, коли один суб'єкт взаємодії змінює деяку частину довкілля, а останні використовують інформацію про її стан пізніше, коли знаходяться в її околиці. Біологічно стігмержи здійснюється через феромон (pheromone) - спеціальний секрет, що відкладається як слід при переміщенні мурав'їв. Феромон - досить стійка речовина, він може сприйматися мурашками декілька діб. Чим вище концентрація феромону на стежці, тим більше мурах по ній рухатиметься. З часом феромон випаровується, що дозволяє мурашкам адаптувати свою поведінку під зміни зовнішнього середовища. Розподіл феромону по простору пересування мурах є свого роду динамічно змінною глобальною пам'яттю мурашника. Будь-яка мурашка у фіксований момент часу може сприймати і змінювати лише один локальний елемент цієї глобальної пам'яті.

Експерименти з Argentine ants, які проводилися Госсом в 1989 і Денеборгом в 1990 році послужили відправною точкою для дослідження роєвого інтелекту. Дослідження застосування отриманих знань для дискретної математики почалися на початку 90-х років XX століття, автором ідеї є Марко Доріго з Університету Брюсселю, Бельгія. Саме він вперше зумів формалізувати поведінку мурах і застосувати стратегію їх поведінки для вирішення задач про найкоротші шляхи. Пізніше за участю Гамбарделли, Тайлларда і Ді Каро були розроблені і багато інших підходів до вирішення складних оптимізаційних задач при допомогою мурашиних алгоритмів.

Ідея мурашиного алгоритму полягає в наступному: дві мурашки з мурашника повинні дістатися до їжі, яка знаходиться за перешкодою. Під час переміщення кожна мурашка виділяє трохи феромону, використовуючи його як маркер. При інших рівних умовах кожна мурашка вибере свою дорогу. Перша мурашка вибирає верхню дорогу, а друга - нижню. Оскільки нижня дорога в два рази коротше верхньої, друга мурашка досягне мети за час t_1. Перша мурашка у цей момент пройде лише половину дороги. Коли одна мурашка

досягає їжі, вона бере один з об'єктів і повертається до мурашника по тій же дорозі. За час t_2 друга мурашка повернеться в мурашник з їжею, а перша мурашка досягне їжі. При переміщенні кожної мурашки на дорозі залишається трохи феромону. Для першої мурашки за час t_0t_2 дорога була покрита феромонами лише один раз. У той же самий час друга мурашка покрила дорогу феромонами двічі. За час t_4 перша мурашка повернулася в мурашник, а друга мурашка вже встигла ще раз сходити до їжі і повернутися. При цьому концентрація феромону на нижній дорозі буде в два рази вище, ніж на верхній. Тому перша мурашка наступного разу вибере нижню дорогу, оскільки там концентрація феромону вища. У цьому і полягає базова ідея мурашиного алгоритму - оптимізація шляхом непрямого зв'язку між автономними агентами.

Розглянемо покроковий опис алгоритму. Передбачимо, що довкілля для мурашок є повний неорієнтований граф. Кожне ребро має вагу, яка позначається як відстань між двома вершинами, сполученими їм. Граф двонаправлений, тому мурашка може подорожувати по грані в будь-якому напрямку.

Імовірність включення ребра в маршрут окремої мурашки пропорційна кількості феромону на цьому ребрі, а кількість феромону, що відкладається, пропорційна довжині маршруту. Чим коротше маршрут тим більше феромону буде відкладено на його ребрах, отже, більша кількість мурашок включатиме його в синтез власних маршрутів. Моделювання такого підходу, що використовує лише позитивний зворотний зв'язок, приводить до передчасної збіжності - більшість мурашок рухаються по локально-оптимальному маршруту. Уникнути цього можна моделюючи негативний зворотний зв'язок у вигляді випару феромону. Причому, якщо феромон випаровується швидко, то це наводить до втрати пам'яті колонії і забуванню хороших рішень, з іншого боку, великий час випарів може привести до здобуття стійкого локального оптимального рішення.

Мурашка. Мурашка - це програмний агент, який є членом великої колонії і використовується для вирішення якої-небудь проблеми. Агент

забезпечується набором простих правил, які дозволяють йому вибирати дорогу в графі. Він підтримує список вузлів, які вже відвідав. Таким чином, мурашка повинна проходити через кожен вузол лише один раз. Дорога між двома вузлами графа, по якому мурашка відвідав кожен вузол лише один раз, називається шляхом Гамільтона.

Вузли в списку «поточної подорожі» розташовуються в тому порядку, в якому мурашка відвідував їх. Пізніше список використовується для визначення протяжності дороги між вузлами. Справжня мурашка під час переміщення по дорозі залишатиме за собою феромон. У алгоритмі агент залишає феромон на ребрах графа після завершення подорожі. Стартова точка, куди поміщається мурашка, залежить від обмежень, що накладаються умовами задачі, оскільки для кожної задачі спосіб розміщення мурашок є визначальним. Або всі вони поміщаються в одну точку, або в різні з повтореннями, або без повторень. На цьому ж етапі задається початковий рівень феромону. Він ініціалізується невеликим позитивним числом для того, щоб на початковому кроці ймовірності переходу в наступну вершину не були нульовими.

Рух мурашок. Рух мурашок ґрунтується на одному простому ймовірнісному рівнянні. Якщо мурашка ще не закінчила дорогу, тобто не відвідала всі вузли мережі, для визначення наступного ребра дороги використовується рівняння

$$P = \frac{\tau(r,u)^{\alpha}\eta(r,u)^{\beta}}{\sum_{k}\tau(r,u)^{\alpha}\eta(r,u)^{\beta}}.$$

Тут $\tau(r,u)$ - інтенсивність ферменту на ребрі між вузлами r та u, $\eta(r,u)$ - функція, яка представляє вимір зворотної відстані для грані, α - вага ферменту, а β - коефіцієнт евристики. Параметри α та β визначають відносну значущість двох параметрів, а також їх вплив на рівняння. Оскільки мурашка подорожує лише по вузлах, які ще не були відвідані (як вказано списком табу), ймовірність розраховується лише для ребер, які ведуть до ще не відвіданих вузлів. Змінна k представляє ці ребра.

Подорож мурашок. Пройдений мурашкою шлях відображується, коли вона відвідає всі вузли графа. Цикли заборонені, оскільки в алгоритм включений список табу. Після завершення довжина дороги може бути підрахована - вона дорівнює сумі довжин всіх ребер, по яких подорожувала мурашка. Рівняння показує кількість феромону, який був залишений на кожному ребрі шляху для мурашки k. Змінна Q є константою.

$$\Delta\tau_{ij}^{k}(t)=\frac{Q}{L^{k}(t)}.$$

Результат рівняння є засобом виміру шляху, - коротка дорога характеризується високою концентрацією феромону, а довша дорога - нижчою. Потім отриманий результат використовується в іншому рівнянні, аби збільшити кількість феромону уздовж кожного ребра пройденої мурашкою дороги.

$$\tau_{ij}(t)=\Delta\tau_{ij}(t)+(\tau_{ij}^{k}(t)\rho).$$

Важливо, що дане рівняння застосовується до всього шляху, при цьому кожне ребро позначається феромоном пропорційно довжині дороги. Тому слід діждатися, поки мурашка закінчить подорож і тільки потім відновити рівні феромону, інакше дійсна довжина дороги залишиться невідомою. Константа ρ - значення між 0 та 1.

Випар феромону. На початку шляху в кожного ребра є шанс бути обраним. Аби поступово видалити ребра, які входять в гірші шляхи графа, до всіх ребер застосовується процедура випару феромону. Використовуючи константу ρ з попереднього рівняння, ми отримуємо нове рівняння

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

Тому для випару феромону використовується зворотний коефіцієнт оновлення шляху.

Повторний запуск. Після того, як дорога мурашки завершена, ребра оновлені відповідно до довжини шляху і стався випар феромону на всіх ребрах, алгоритм запускається повторно. Список табу очищається, і довжина дороги обнуляється. Мурашкам дозволяється переміщатися по графові, засновуючи

вибір ребра на першому рівнянні. Цей процес може виконуватися для постійної кількості шляхів або до моменту, коли впродовж декількох запусків не було відмічено повторних змін. Потім визначається кращий шлях, який і є рішенням.

Розглянемо функціонування алгоритму на прикладі сценарію з двома мурашками. На рисунку 5.40 показаний приклад з двома ребрами між двома вузлами (V_0 та V_1). Кожне ребро ініціалізується і має однакові шанси на те, аби бути обраним.

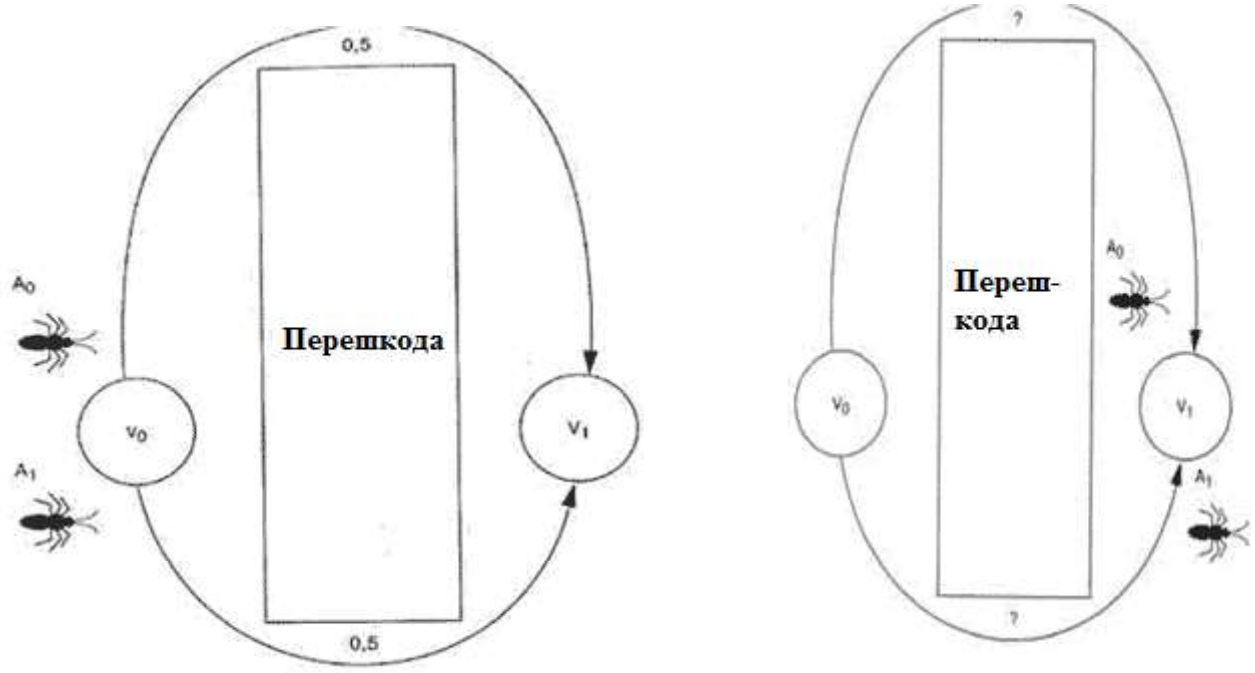


Рис. 5.40. Сценарій з двома мурашками.

Дві мурашки знаходяться у вузлі V_0 і позначаються як A_0 та A_1. Оскільки ймовірність вибору будь-якої дороги однакова, то в цьому циклі ігнорується рівняння вибору дороги. Дані для задачі:

- число пройдених кроків: для A_0 - 20, для A_1 - 10;
- рівень феромону (Q /пройдену відстань): для A_0 - 0.5, A_1 - 1.0;
- $\rho = 0.5,\ \alpha = 0.3,\ \beta = 1.0.$

Кожна мурашка обирає свій шлях (мурашка A_0 йде по верхній дорозі, а мурашка A_1 - по нижній). Мурашка A_0 зробив 20 кроків, а мурашка A_1, - лише 10. Згідно другого рівняння розраховуємо кількість феромону, яке має бути

«нанесене». Далі по третьому рівнянню розраховується кількість феромону, яка буде застосовано. Для мурашки A_0 результат складає $0.1 + (0.5 * 0.6) = 0.4$ Для мурашки A1 результат складає $0.1 + (0.1 * 0.6) = 0.7$. За допомогою четвертого рівняння визначається, яка частина феромону випарується і, відповідно, скільки залишиться. Результати (для кожного шляху відповідно) складають

$$0.4 * (1.0 - 0.6) = 0.16$$
$$0.7 * (1.0 - 0.6) = 0.28.$$

Ці значення представляють нову кількість феромону для кожного шляху (верхнього і нижнього, відповідно). Тепер перемістимо мурашку назад у вузол V_0 і скористаємося першим імовірнісним рівнянням вибору шляху, аби визначити, яку дорогу повинні обрати мурашки. Вірогідність того, що мурашка обере верхню дорогу (представлену кількістю феромону 0.16), складає

$$\frac{(0.16)^{3.0} * (0.5)^{1.0}}{(0.16)^{3.0} * (0.5)^{1.0}} + (0.28)^{3.0} * (1.0)^{1.0} = \frac{0.002048}{0.024} = P(0.085).$$

Вірогідність того, що мурашка обере нижню дорогу (представлену кількістю феромону 0.28) складає

$$\frac{(0.28)^{3.0} * (1.0)^{1.0}}{(0.16)^{3.0} * (0.5)^{1.0}} + (0.28)^{3.0} * (1.0)^{1.0} = \frac{0.021952}{0.024} = P(0.915).$$

При зіставленні двох імовірностей обидві мурашки виберуть нижню дорогу, яка є оптимальною.

Слід зазначити, що для алгоритму мурашиної колонії необхідно вказати: закон виділення феромону, закон випару феромону, кількість агентів, місця розміщення. Всі ці характеристики вибираються з врахуванням особливості задачі на основі експериментальних досліджень.

За останній час було запропоновано декілька метаевристичних моделей мурашиної колонії. Серед них три найбільш успішні:

- Ant system (Dorigo, 1996);
- Ant colony system (ACS) (Dorigo & Gambardella, 1997);
- MAX-MIN Ant system (MMAS) (Stutzle & Hoos, 2000).

Розглянемо застосування мурашиних алгоритмів до рішення вже відомої задачі комівояжера. Дослідимо реалізацію чотирьох складових самоорганізації мурашок при оптимізації маршруту комівояжера. Багатократність взаємодії реалізується ітераційним пошуком маршруту комівояжера одночасно декількома мурашками. При цьому кожна мурашка розглядається як окремий, незалежний комівояжер, вирішуючий свою задачу. За одну ітерацію алгоритму кожна мурашка здійснює повний маршрут комівояжера. Позитивний зворотний зв'язок реалізується як імітація поведінки мурашок типа «залишення слідів - переміщення по слідах». Чим більше слідів залишено на тропі - ребрі графа в задачі комівояжера, тим більше мурашок пересуватиметься по ній. Для задачі комівояжера позитивний зворотний зв'язок реалізується наступним стохастичним правилом: ймовірність включення ребра графа в маршрут мурашки пропорційна кількості феромону на ньому. Вживання такого імовірнісного правила забезпечує реалізацію і іншої складовій самоорганізації - випадковості. Кількість феромону, що відкладається мурашкою, на ребрі графа обратно пропорційно до довжини маршруту. Чим коротше маршрут, тим більше феромону буде відкладено на відповідних ребрах графа і тим більше мурашок використовуватиме їх при синтезі своїх маршрутів. Відкладений на ребрах феромон виступає як підсилювач, він дозволяє хорошим маршрутам зберігатися в глобальній пам'яті мурашника. Ці маршрути можуть бути покращені на подальших ітераціях алгоритму.

Використання лише позитивного зворотного зв'язку приводить до передчасної збіжності рішень - до випадку, коли всі мурашки рухаються одним і тим же субоптимальним маршрутом. Для уникнення цього використовується негативний зворотний зв’язок - випар феромону.

Для кожної мурашки перехід з міста i в місто j залежить від трьох складових: пам'яті мурашки (tabu list), видимості і віртуального сліду феромону. Tabu list - це список відвіданих мурашкою міст, заходити в які ще раз не можна. Використовуючи цей список, мурашка гарантовано не попаде в одне і те ж місто двічі. Позначимо через, J_{ik} список міст, які ще необхідно

відвідати мурашці k, що знаходиться в місті i. Видимість - величина, зворотна відстані: $\eta_{ij} = 1/D_{ij}$, де D_{ij} - відстань між містами i та j. Видимість - це локальна статична інформація, що висловлює евристичне бажання відвідати місто j з міста i, - чим ближче місто, тим більше бажання відвідати його. Віртуальний слід феромону на ребрі (i, j) представляє підтверджене мурашиним досвідом бажання відвідати місто j з міста i. На відміну від видимості слід феромону є глобальнішою і динамічнішою інформацією - вона змінюється після кожної ітерації алгоритму, відображаючи придбаний мурашками досвід. Кількість віртуального феромону на ребрі (i, j) на ітерації t позначимо через $\tau_{ij}(t)$. Подальша робота алгоритму аналогічна наведеному вище.

Загальна кількість мурашок в колонії залишається постійною впродовж виконання алгоритму. Звичайне число мурашок призначають рівним кількості міст - кожна мурашка починає маршрут зі свого міста. Для поліпшення часових характеристик мурашиного алгоритму вводять так званих елітних мурашок. Елітна мурашка підсилює ребра найкращого маршруту, знайденого з початку роботи алгоритму. Кількість феромону, що відкладається на ребрах найкращого поточного маршруту T^+ приймається рівним Q/L^+, де L^+ - довжина маршруту T^+. Цей феромон спонукає мурашок до дослідження рішень, що містять декілька ребер найкращого на даний момент маршруту T^+. Якщо в мурашнику є e елітних мурах, то ребра маршруту T^+ отримуватимуть загальне посилення $\Delta\tau_e = eQ/L^+$.

Практична реалізація вказаного алгоритму може бути виконана різними програмними засобами. Зокрема, українським вченим С. Д. Штовбою запропонован варіант його застосування засобами пакету MATLAB. Для задачі з 29 населеними пунктами в Баварії «Bays - 29» алгоритм без елітних мурашок після 100 ітерацій знайшов оптимальний маршрут завдовжки 2020 лише в одному випадку з п'яти. Рішення можна поліпшити простим збільшенням кількості ітерацій до 1 - 2 тисяч. Довжини маршрутів мурашок на одній ітерації

відрізняються трохи, тому для прискорення знаходження оптимуму необхідно штучно підсилювати найкращі поточні рішення за допомогою елітних мурашок.

Еволюція маршрутів комівояжера, знайдених алгоритмом з трьома елітними мурашками, показана на рис. 5.41.

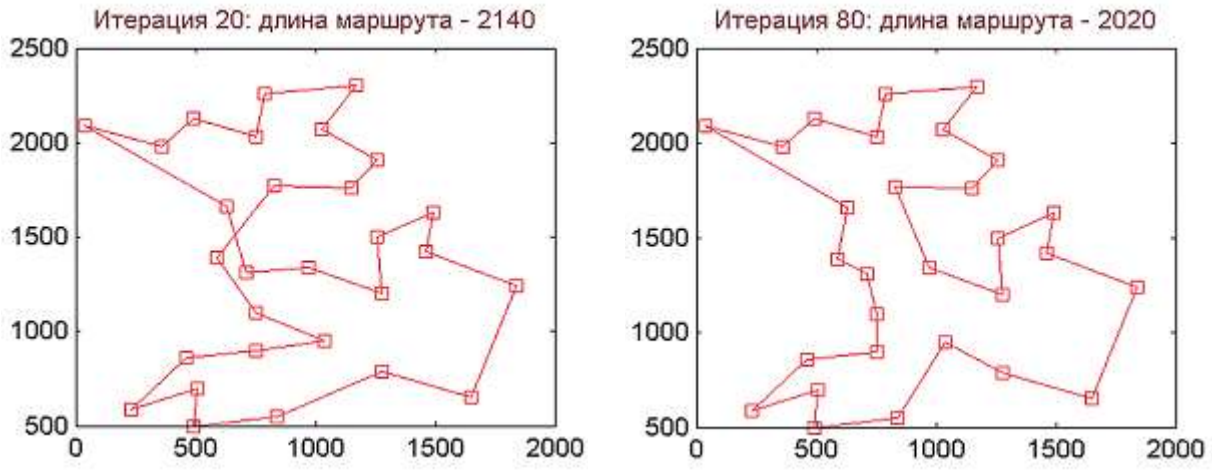


Рис.5.41. Еволюція маршрутів комівояжера.

В порівнянні з точними методами, наприклад динамічним програмуванням або методом гілок і границь, мурашиний алгоритм знаходить близькі до оптимуму рішення за значно менший час навіть для задач невеликої розмірності. Час оптимізації мурашиним алгоритмом є поліноміальною функцією від розмірності $F(t,n^2,m)$, тоді як для точних методів залежність експоненціальна.

Мурашиний алгоритм оптимізації маршруту комівояжера після незначних модифікацій може використовуватися для вирішення різних комбінаторних задач: квадратичної задачі про призначення (Quadratic Assignment Problem), задачі про оптимізацію маршрутів вантажівок (Vehicle Routing Problem), задачі календарного планування (Job-Shop Schedule Planing), задачі розфарбовування графа (Graph Coloring Problem) та ін. Важливі результати були отримані для нестаціонарних систем із змінними в часі параметрами, наприклад, для розрахунків телекомунікаційних і комп'ютерних

мереж. Мурашиний алгоритм застосовувався для розробки оптимальної структури знімальних мереж GPS в рамках створення високоточних геодезичних і знімальних технологій.

Фахівці з американської дослідницької компанії Pacific Northwest National Laboratory знайшли новий підхід до вивчення безпеки комп'ютерних мереж. Для боротьби з вірусами, троянами і комп'ютерними черв'яками вони вирішили використовувати мурашині алгоритми. За допомогою спеціалізованої програми, дослідники намагаються знайти «мережеві аномалії». Їх програма використовує розподілені по комп'ютерних мережах сенсори, що безперервно збирають дані. Немов мурахи, які передають своїм родичам інформацію про їду або небезпеку за допомогою запахів, ці сенсори діляться зібраною інформацією один з одним. Таким чином, програма може визначити своєрідні мережеві аномалії, що сигналізують про можливу небезпеку, наприклад про масштабне зараження мережі. Сенсори бувають різної спрямованості - одні можуть збирати дані про надмірне завантаження центрального процесора комп'ютерів, а інші - перевіряти мережевий трафік. Також є «вартові» - спеціальні блоки програми, що аналізують інформацію, отриману від всіх сенсорів - мурашок. Хоча інноваційний антивірусний комплекс знаходиться на ранній стадії розробки, вже зараз він здатний виявляти деяких комп'ютерних черв'яків.

Перспективними напрямами поліпшення мурашиних алгоритмів є різні адаптації параметрів з використанням бази нечітких правил і їх гібридизація, наприклад, з генетичними алгоритмами. Як варіант, така гібридизація може полягати в обміні через певні проміжки часу поточними найкращими рішеннями.

Достоїнствами мурашиних алгоритмів є:

- в порівняння з GAs (Genetic Algorithms) мурав’їнні алгоритми спираються на пам'ять всієї колонії замість пам'яті лише про попереднє покоління і менше схильні до неоптимальних початкових рішень (із-за випадкового вибору шляху);

- можуть використовуватися в динамічних додатках (швидко адаптуються до змін);
- мають важливу практику застосовування до множини різних задач.

Недоліки таких алгоритмів вважають:

- ускладнений теоретичний аналіз в результаті послідовності випадкових (незалежних) рішень і зміни розподілу вірогідності при ітераціях;
- збіжність алгоритму гарантується, але час збіжності не визначений;
- зазвичай необхідне вживання додаткових методів таких, як локальний пошук;
- сильно залежать від параметрів настройки, які підбираються лише виходячи з експериментів.

Мурашині алгоритми засновані на імітації самоорганізації соціальних комах за основі використання динамічних механізмів, за допомогою яких система досягає глобальної мети в результаті локальної низькорівневої взаємодії елементів. Мурашині алгоритми забезпечують рішення також інших комбінаторних задач не гірше за загальні метаевристичні технології оптимізації і деякі проблемно-орієнтовані методи. Важливою властивістю мурашиних алгоритмів є неконвергентність: навіть після великого числа ітерацій одночасно досліджується множина варіантів рішення, внаслідок чого не відбувається тривалих часових задержок в локальних екстремумах. Все це дозволяє успішно застосовувати такі алгоритмі для вирішення складних задач інтелектуального аналізу даних.

Генетичне програмування (genetic programming, GP) - одна з найзручніших і універсальних методик вирішення задач, що встають перед аналітиками. Воно застосовується для вирішення широкого круга проблем: символьної регресії (symbolic regression), аналізу даних (data mining), оптимізації і дослідження поведінки потомства (emergent behavior), що з'являється, в біологічних співтовариствах.

Ідею генетичного програмування (ГП) вперше запропонував Дж. Коза в 1992 році, спираючись на концепцію генетичних алгоритмів. Генетичне

програмування - це розширення генетичної моделі навчання в область програмного забезпечення. Його об'єктом на відміну від генетичних алгоритмів є не символьні рядки фіксованої довжини, що кодують можливі рішення проблеми, а власне програми, виконуючи які і отримують різні варіанти рішення задачі. У генетичному програмуванні програми представляються у вигляді дерева граматичного розбору, а не у вигляді рядків коду, тому в ГП всі операції виконуються не над рядками, а над *деревами*. При цьому використовуються такі ж оператори, як і в генетичному алгоритмі: селекція, схрещування і мутація [31, 66].

У ГП хромосомами є *програми*. Програми представлені як дерева з *функціональними* (проміжними) і *термінальними* (кінцевими) елементами. Термінальними елементами є константи, дії і функції без аргументів, функціональними - функції, що використовують аргументи. Для приклада можна розглянути функцію $y = 2 + (4/7)x$, представлену на рис. 5.42. Термінальні елементи $T = \{2, x, 4, 7\}$, функціональні $F = \{+, *, /\}$.

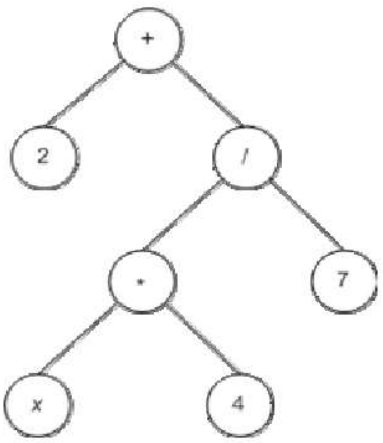


Рис. 5.42. Деревовидне представлення функції.

Для того, щоб застосувати ГП до якої-небудь проблеми, необхідно визначити:

- множину термінальних елементів;
- множину функціональних елементів;
- міру пристосованості;

- параметри, контролюючі еволюцію;
- критерій останову еволюції.

Генетичні програми не пишуться програмістами, а створюються в результаті наступного ітераційного покрокового процесу.

1. Генерується початкова популяція програм, що є випадковими композиціями функцій (арифметичних, логічних та ін. операцій) і терміналів (змінних і констант), узятих з множини функціональних і термінальних елементів, що відносяться до вирішуваної проблеми. Якщо ми збираємося вивести програму, яка управляє транспортною логістикою, то до набору терміналів увійдуть наступні змінні - відстань до об'єкта, швидкість і вантажопідйомність, тип транспортних засобів і так далі. Набор функцій в цьому випадку включатиме різні математичні операції, як прості (множення, ділення, віднімання, складання), так і складніші.

2. Кожна програма виконується і їй привласнюється значення пристосованості, відповідне тому, наскільки добре вона вирішує задачу.

3. Створюється нова популяція комп'ютерних програм за рахунок:

- копіювання в неї найкращих програм старої популяції;
- створення нових програм за допомогою мутацій (випадкової зміни функцій і терміналів або навіть цілих піддерев);
- створення нових програм за допомогою схрещування тих, що існують. Операція схрещування реалізується за рахунок обміну піддерев двох хромосом, вибраних випадково (відповідно до їх пристосованості).

4. Всі програми знову виконуються і цикл повторюється до тих пір, поки не буде отриманий необхідний результат.

Слід зазначити, що алгоритм роботи ГП такий же, як і ГА: селекція, схрещування і мутація. Проте оскільки ГП оперує над деревами, а не над рядками, то оператори схрещування і мутації мають відмінності.

Оператор схрещування працює наступним образом: вибираються випадкові частини батьківських дерев, і ці частини міняються місцями (рис. 5.43).

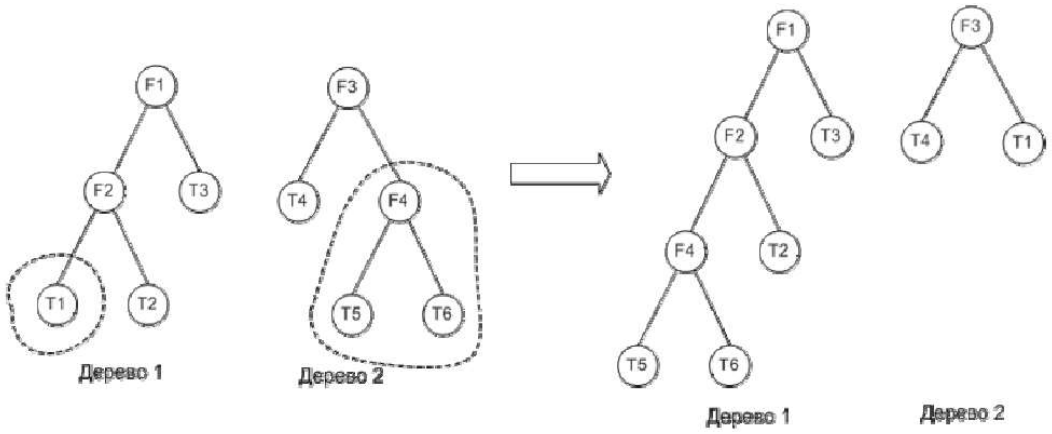


Рис. 5.43. Схрещування двох дерев.

В якості особливості необхідно відзначити, що в ГП розмір хромосоми міняється. Аби запобігти надмірному розростанню дерева, вводять максимальну кількість функціональних елементів в дереві або максимальну глибину дерева. Проте при операції схрещування можлива ситуація, коли при схрещуванні двох дерев вийде одне з дерев, що перевершує заданий ліміт. В цьому випадку замість конфліктного дерева копіюється батьківське дерево (рис. 5.44).

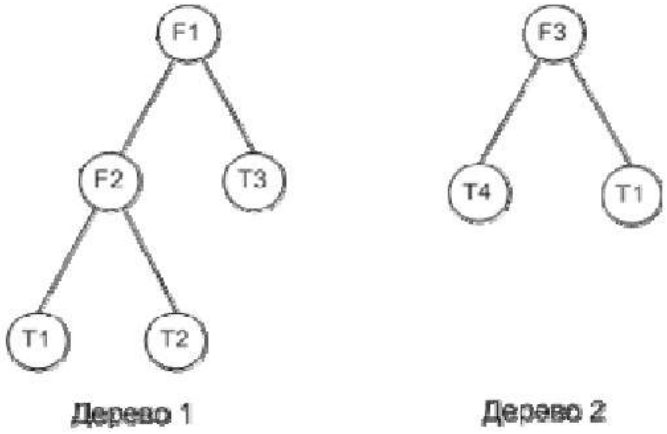


Рис. 5.44. Розв’язання конфліктної ситуації попереднього оператора схрещування при максимальній глибині дерева, рівній трьом.

Також слід зазначити, що часто при вживанні оператора схрещування з'являються *інтрони* - ділянки коди, що нічого не роблять (наприклад, обчислюють вирази вигляду $x = x * 1$). На перший погляд, час, який ГП витрачає на збільшення і розвиток інтронів, проходить даремно. З іншого боку,

інтрони допомагають зберігати від руйнування хороші блоки для майбутніх поколінь, що підвищує шанси на здобуття ще ефективніших особин.

Мутація. Оператор мутації випадково видаляє частину дерева і замінює її новим деревом (рис. 5.45).

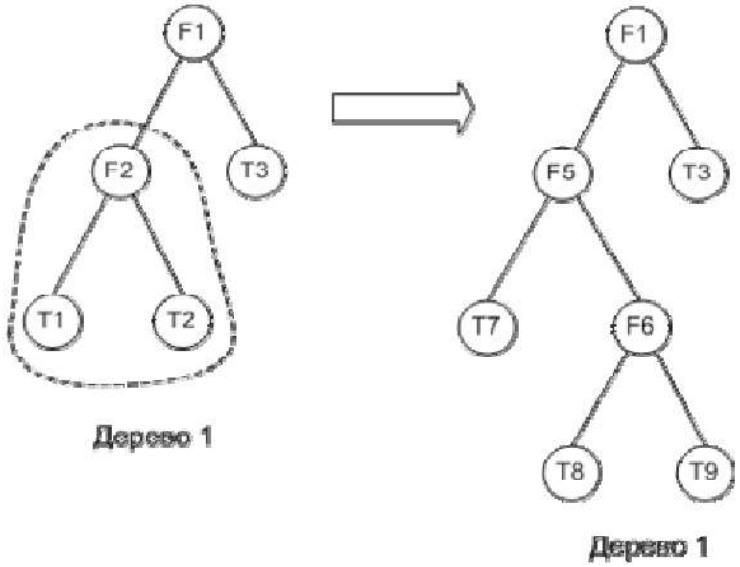


Рис. 5.45. Дія оператора мутації.

Зупинимося на деяких напрямах застосування генетичного програмування.

Класифікуючі системи (classifier systems) - дуже цікавий напрям, що вивчає питання створення самонавчальних машин. Один з його творців Дж. Холланд запропонував для цих цілей використовувати когнітивну систему, здатну класифікувати стан довкілля і відповідно реагувати на цей стан. Що необхідне для побудови такої системи? Вочевидь (1) - середовище; (2) - рецептори, які повідомлятимуть систему про те, що відбувається; (3) - ефектори, які дозволять нашій системі маніпулювати середовищем; і (4) - власне систему, «чорний ящик», який має (2) і (3) і «живе» усередині (1). Такі системи часто називають «аніматами» (animal + robot = animat). Основна ідея класифікуючих систем - почавши з нульового знання, використовуючи випадково згенеровану класифікуючу популяцію, дозволити системі створити свою програму за допомогою індукції. Вона приводить вхідний потік до

деякого шаблону, який дозволяє анімату класифікувати свій поточний стан/контекст і реагувати відповідним чином [62].

Системи зі складною поведінкою. Програмні і апаратні системи, а також їх окремі елементи, часто мають складну поведінку (наприклад, пристрої управління і мережеві протоколи). Останнім часом для опису об’єктів із складною поведінкою в програмуванні пропонується використовувати автоматний підхід. Еволюційним моделям обчислень у вигляді кінцевих автоматів було присвячено багато досліджень. При цьому більшість авторів займалися оптимізацією автоматів - розпізнавателей (parsers) і перетворювачів (transducers). Відповідно до автоматного підходу об'єкт представляється у вигляді автоматизованого об'єкту - сукупності управляючого автомата і об'єкту управління. При цьому вся «логіка поведінки» виявляється зосередженою в автоматі, що управляє. Побудова автомата, що управляє, зазвичай вимагає від розробника набагато більше зусиль і породжує більше помилок, чим реалізація об'єкту управління. Для деяких задач евристична побудова автомата або неможлива, або досить складна. Крім того, навіть для простих задач автомат, побудований вручну, часто є неоптимальним. Цю проблему можна вирішувати методами генетичного програмування. При цьому автомати генеруються автоматично, що істотно спрощує побудову автоматних програм і дозволяє підвищити рівень автоматизації їх проектування.

До перспективних напрями розвитку ГП слід віднести роботи по так званих автоматично визначаємих функціях (ADF), ідея яких полягає в підвищенні ефективності ГП за рахунок модульної побудови програм, що складаються з головної програми і ADF-модулів, які генеруються в ході моделювання еволюції. До початку еволюції визначається структура програми, число ADF-модулів і параметри кожної ADF. Всі вершини даної структури мають свій номер, список аргументів включає окремі локальні змінні, які визначаються при виклику ADF. Задання функції головної програми завершує установку загальної програми. Налаштування ADF (їх число, аргументи і тому

подібне) залежить від вирішуваної задачі, наявних обчислювальних ресурсів і попереднього досвіду.

Іншим перспективним напрямом є реалізація ГП на трансп'ютерних обчислювальних системах.

Тест

1. Застосування еволюційних методів в інтелектуальному аналізі даних передбачає, що …

а) моделювання складного об'єкту замінюється моделюванням його генетичних параметрів;

б) моделювання складного об'єкту замінюється моделюванням його популяції;

в) моделювання складного об'єкту замінюється моделюванням його біологічних характеристик;

г) моделювання складного об'єкту замінюється моделюванням його еволюції.

2. Під поняттям «еволюційні обчислення» Ви розумієте:

а) сукупність алгоритмів пошуку, оптимізації або навчання, заснованих на формалізованих принципах штучного інтелекту;

б) сукупність алгоритмів пошуку, оптимізації або навчання, заснованих на формалізованих принципах природного біологічного процесу;

в) сукупність алгоритмів пошуку, оптимізації або навчання, заснованих на формалізованих принципах природного еволюційного процесу.

3. Якщо для аналізу даних Ви вирішили застосувати еволюційний алгоритм на основі методу групового обліку аргументів, то розумієте, що його сутністю є …

а) розробка сімейства імітаційних алгоритмів для моделювання мультипараметричних даних;