УДК 519.876.5

І. Скрильник

Полтавський національний технічний університет імені Юрія Кондратюка

ПОФАРБУВАННЯ ГРАФІВ ЗА ДОПОМОГОЮ ГЕНЕТИЧНОГО АЛГОРИТМУ

Резюме. *Розглянуто задачу пофарбування графів із накладеними обмеженнями. Показано, що її можна успішно розв'язати за допомогою генетичного алгоритму, розробленого на основі 0-1 програмування та квадратичної оптимізації. Особливу увагу автор приділяє кодуванню розв'язків та функції пристосованості. За результатами теоретичних досліджень створено програмний продукт Pattern, наведено приклад роботи цієї програми, розраховано складність алгоритму.*

Ключові слова: *граф, генетичний алгоритм, пофарбування, хроматичне число, функція належності, функція пристосованості.*

I. Skrylnyk

COLOURING GRAPH USING THE GENETIC ALGORITHM

The summary. *The problem of graph colouring with constraints is investigated. It has been shown that this problem can be successfully resolved using the genetic algorithm based on integer programming and quadratic optimization. The importance of coding in the chromosomes and the choice of fitness function is precisely examined in the article. The theoretical investigations resulted in special software development named Pattern. The example of calculus using Pattern software is presented. The complexity of algorithm is calculated.*

Key words: *graph, genetic algorithm, graph coloring problem, chromatic number, membership function, fitness function*

Умовні позначення:

$\boldsymbol{V}$ – *множина вершин графа;*

$\boldsymbol{E}$ – *множина ребер графа;*

$\boldsymbol{G}$ – *граф;*

$\boldsymbol{C}$ – *множина кольорів;*

$\boldsymbol{\mathcal{R}(C)}$ – *пофарбування графа;*

$\boldsymbol{P}$ – *популяція індивідів;*

$\boldsymbol{h_i}$ – *ген;*

$\boldsymbol{\sigma_I}$ – *стандартне відхилення генів;*

$\boldsymbol{p_c}$ – *коефіцієнт схрещування;*

$\boldsymbol{p_m}$ – *коефіцієнт мутації.*

Вступ. Група алгоритмів, в основі яких використано ідею еволюції біологічних систем, називають еволюційними алгоритмами. Існує кілька напрямків розвитку цієї теорії: генетичні алгоритми (Genetic Algorithms), еволюційні стратегії (Evolution Strategies), генетичне програмування (Genetic Programming), еволюційне програмування (Evolutionary Programming).

Історія еволюційних обчислень почалася з розроблень різних незалежних моделей. Основними з них були генетичні алгоритми та класифікаційні системи Холланда (J. Holland). У 1975 році він опублікував книгу "Адаптація у природних та штучних системах" ("Adaptation in Natural and Artifical Systems"). У 70-х роках Л.А. Растрігін запропонував алгоритми, у яких використано ідею біонічної поведінки особин. Ця ідея знайшла відображення у циклі робіт І.Л. Букатової з еволюційного моделювання, у роботах М.Л. Цетліна про оптимальну поведінку стохастичних автоматів, а також у роботах Ю.І. Неймарка. Останній запропонував здійснити пошук глобального екстремуму на основі множини незалежних автоматів, які моделюють

процеси розвитку та елімінації. Великий вклад у розвиток еволюційного програмування внесли Фогель (Fogel) та Уолт (Walsh) [1, 2].

Останнім часом широко застосовують генетичні алгоритми. Під терміном "генетичні алгоритми" приховано не одну модель, а достатньо широкий клас алгоритмів, часом мало схожих між собою. За останні роки реалізовано багато генетичних алгоритмів. Їх використовують для розв'язання таких конвексних та NP-складних задач, як: пошук глобального екстремуму багатопараметричної функції, апроксимації функції, знаходження найкоротшого шляху, розміщення, налаштування штучної нейронної мережі, ігрових стратегій, навчання машин. Фактично генетичні алгоритми максимізують багатопараметричні функції, тому їх область використання є широкою. Усі наведені вище задачі розв'язують саме шляхом формування функції, що залежить від певної кількості параметрів, а її глобальний екстремум відповідає розв'язку задачі [3]. Генетичні алгоритми також дають змогу фарбувати графи з великою кількістю вершин.

Мета роботи. Розглянути особливості застосування генетичного алгоритму до пофарбування графів.

Пофарбування графа з накладеними обмеженнями. Розглядали задачу пофарбування графа *G(V,E)* із накладеними обмеженнями, на основі якої створено квадратичну модель [4, 5]:

$$\begin{cases} f(x) = x \cdot Q \cdot x^T \to min; \\ \begin{cases} x_{i,h} + x_{j,h} \le 1 \quad \forall (i,j) \in A, \forall h; \\ \sum\limits_{h=1}^{h=|P|} x_{i,h} = 1 \qquad \forall i; \\ x_{i,h} \in \{0,1\}. \end{cases} \\ для\ x_{opt}^{\ *}\ \exists\ min\ f(x),\ P \circ x_{opt}^{\ *} = \Re \end{cases} \tag{1}$$

Отже, при накладених обмеженнях (1) необхідно знайти мінімум функції належності *f(x)*, при якому пофарбування $\Re(C)$ графа G буде оптимальним. Змінна x є бінарною змінною $\{0,1\}$, що визначає пофарбування вершин графа, Q – матриця суміжності графа G, у якій на головній діагоналі розташовані числа $diag\{a_1, a_2, \ldots, a_r\}$, такі, що $\det Q > 0$. У системі обмежень (1) перший вираз означає, що дві суміжні вершини (i, j) не можуть бути пофарбованими в один і той же колір. Другий вираз означає, що одна і та ж вершина не може бути пофарбована різними кольорами із набору кольорів *K*. Візьмемо, що кількість вершин дорівнює $n \times n$.

Універсальність генетичного алгоритму – у строго визначеному наборі процедур, що направляють вектор розв'язків в область глобального екстремуму. При цьому математичний базис кожної із процедур може бути модифікований. Тому прийнято говорити про класичний генетичний алгоритм та його різновиди – модифіковані генетичні алгоритми. Модифіковані генетичні алгоритми мають також другорядні завдання, окрім відшукання глобального екстремуму, такі, як скорочення часу пошуку розв'язку, стійкість розв'язку, забезпечення паралелізму обчислень, зменшення обсягу вхідної інформації. Вище сказане призвело до розвитку такого напрямку, як генетичне програмування. Таким чином, стратегія генетичного алгоритму залишається незмінною.

Кодування розв'язків та функція пристосованості. Ефективність генетичного алгоритму дуже залежить від методу кодування розв'язків. Множину всіх розв'язків називають популяцією *P*, що видозмінюється від одного покоління до іншого.

Відповідно кожна популяція складається із індивідів $P=\{x_1, x_2, \ldots, x_\ell\}$, які називають хромосомами, тобто закодованими розв'язками (генами), об'єднаними в ланцюг $x=(h_1, h_2, \ldots, h_r)\in R^r$. На практиці часто застосовують кодування у вигляді одиниць та нулів, отримуючи таким чином бінарні масиви.

Якщо задано граф $G=(V,E)$, то маємо $|V|$ змінних (кількість вершин). Вважаємо, що значення кожної змінної у бінарному вигляді залежить від кількості фарб. Наприклад, для трьох фарб можливі три значення змінних: $[0\,0\,1], [0\,1\,0], [1\,0\,0]$. Такі значення, як $[0\,0\,0]$ або $[1\,0\,1]$, або інші, що відрізняються від раніше названих, не мають існувати, бо вони суперечать формулюванню задачі пофарбування.

Для чотирьох фарб маємо чотири можливі значення змінних: $[0\,0\,0\,1], [0\,0\,1\,0], [0\,1\,0\,0], [1\,0\,0\,0]$. Для n фарб $[0\,0\ldots1], [0\ldots1\ldots0], \ldots, [1\ldots0\ldots0]$ (усього n можливих значень).

Пофарбувати граф – означає знайти такі значення змінних, для яких виконуються умови (1). Якщо граф має 9 вершин і має бути пофарбованим у 3 кольори, то буде дев'ять змінних, котрі прийматимуть три значення. Тоді розв'язок (хромосома) для задачі пофарбування виглядатиме як бінарний вектор $x_1=(100\,010\,001\,001\,100\,010\,010\,001\,100)$. Існують альтернативні розв'язки, а саме: $x_2=(100\,001\,010\,001\,100\,010\,010\,001\,100)$, $x_3=(001\,100\,010\,001\,100\,010\,010\,001\,100)$ і т.д. Усього може бути 9! варіантів розв'язків, серед них потрібно знайти глобальний екстремум.

Отже, кожна змінна x є вектором. Таким чином, розв'язком задачі пофарбування графів є вектор, елементи якого – вектори. Як відомо, вектор задається координатами, він існує у просторі змінних. Отже, розв'язком задачі пофарбування графів є вектор, що існує у просторі змінних (вершин), кожна з яких існує у своєму власному просторі (у нашому випадку – це бінарний простір, де норма $\|x\|_\infty=1$).

Нехай, наприклад, задано вектор $F(x,y)=[x(i,j,k),\ y(i,j,k)]$ з координатами x та y, що є функціями, заданими у просторі із загальним базисом $[i,j,k]$. Тоді вектор F можна визначити у цьому просторі як $F(i,j,k)=[i\,j\,k\quad i\,j\,k]$. Тим самим ми переходимо з 2-вимірного простору (x,y) у 3-вимірний (i,j,k). Повертаючись до прикладу про 3 кольори та 9 вершин, можна записати $[100\,010\,001\,001\,100\,010\,010\,001\,100]$. Маємо вектор розмірністю $1\times(9\times3)$, тобто 1×27. Фактично ми з простору вершин переходимо у простір фарб, де кожна фарба задана у деякому бінарному просторі. Розмір бінарного простору визначаємо набором фарб. Задаючи набір фарб, задаємо розмір простору, у наведеному вище прикладі це буде 27-вимірний простір. Алгоритм проектували для роботи з векторами фіксованої довжини.

Отже, потрібно знайти оптимальне пофарбування графа *G*, тобто так його пофарбувати, щоб виконувалися обмеження (1) і значення функції належності було мінімальним. Цю задачу можна переформулювати як знаходження максимуму деякої функції $f(x_1, x_2, \ldots x_n)$. Цю функцію називають функцією пристосованості (fitness function) і використовують для оцінювання пристосованості кожного індивіда у поколінні. Одним із необхідних кроків у реалізації генетичного алгоритму є вибір такої функції. Вона має набувати невід'ємних значень, а область визначення параметрів повинна бути обмеженою. У генетичних алгоритмах не використовують такі властивості функції, як неперервність, диференційованість та ін. Функцію мають на увазі як "пристрій" (*blackbox*), що на вхід отримує параметри, а на вихід виводить результат. Якщо, наприклад, потрібно знайти мінімум заданої функції, то достатньо перенести область її значень на додану область, а потім інвертувати. Таким чином,

максимум нової функції буде відповідати мінімуму вихідної функції. Чим більше значення fitness function (або менше значення функції належності), тим краще пристосований індивід, тобто тим ближче отриманий поточний розв'язок до оптимуму.

У даному випадку функція пристосованості має вигляд

$$\overline{f}\left(x_i\right) = \underset{k=1,N_p}{max}\, f\left(x_i\right) - f\left(x_k\right), \tag{2}$$

функцію належності обчислюємо $f\left(x\right) = \sum_{i}^{i=n} x_i \cdot Q \cdot x_i^T$.

Повертаючись до наведеного вище прикладу, що закодований розв'язок для графа дорівнює $x = (100\,010\,001\,001\,100\,010\,010\,001\,100)$ та кількість фарб дорівнює $n = 3$, визначимо значення функції належності. Для цього утворену хромосому *x* розіб'ємо на три бінарні змінні $x_1 = (100001010)$, $x_2 = (010100001)$, $x_3 = (001010100)$, тоді значення функції належності обчислюємо за формулою $f\left(x\right) = x_1 \cdot Q \cdot x_1^T + x_2 \cdot Q \cdot x_2^T + x_3 \cdot Q \cdot x_3^T$.

Програма Pattern. За результатами теоретичних досліджень створено універсальний програмний продукт Pattern, який розробляли у середовищі MatLab. Він дозволяє генерувати випадковим чином граф із вибраною кількістю вершин $n \times n$ та за допомогою генетичного алгоритму фарбувати його у довільно задану кількість кольорів.

У даному алгоритмі початкова популяція формується методом випадкового генерування хромосом за законом нормально розподіленої величини. Фіксується розмір популяції, який не змінюється протягом роботи всього алгоритму. Кожний індивід утворюється із двох компонентів $\left(h_i, \sigma_i\right)$, а саме *r*-ої кількості генів h_i та *r* стандартних відхилень цих генів $\sigma_i = \left[\sigma_{j,1}, \sigma_{j,2}, \ldots, \sigma_{j,r}\right]^T$. У програмі кожний окремий ген $h_{i,j}$ з урахуванням стандартного відхилення $\sigma_i = N(0,1)$ кодується таким чином:

$$h_{i,j} = \left(Random\left(\sum_{j=1}^{K} \upsilon_j = 1 \right), \sigma_i \right), \tag{3}$$

де $Random(\bullet)$ позначає операцію генерування випадкової бінарної величини із множини значень $\{0,1\}$.

Розроблений алгоритм, як і класичний генетичний алгоритм, складається з чотирьох стадій: генерація початкової популяції (*initial generation*), відбір (*selection*) хромосом для схрещування (*recombination*), власне операція схрещування шляхом кросовера (*crossover*) та мутація нового покоління. Перелічені стадії призводять до формування нового покоління (*next generation*).

У класичному генетичному алгоритмі ймовірність кожної особини передати свій генетичний матеріал у наступну популяцію пропорційна її пристосованості, тобто працює пропорційний відбір (*proportional selection*). Застосовано спосіб відбору *remainder stochastic sampling*, тобто для кожної особини обчислюється відношення її пристосованості до середньої пристосованості популяції:

$$\delta_i = \frac{\overline{f}\left(x_i\right)}{\sum_{k=1}^{N_p} \overline{f}\left(x_k\right)}. \tag{4}$$

При цьому генерується ще випадкова величина $z_i = Random([0,1])$, хромосому x_i вважаємо відібраною, якщо $z_i \geq \delta_i$.

Після відбору особини проміжної популяції випадковим чином розбиваються на пари. Кожна з них із ймовірністю p_c схрещується (до неї застосовують оператор

кросовера), у результаті чого отримуємо два нащадки. Вони записуються у проміжне покоління. Якщо ж парі не випало схрещуватися, то у нове покоління записують самі особини цих пар. Проміжне покоління (*intermidiate generation*) складається із початкових генерованих особин та рекомбінованих нащадків. При цьому кількість хромосом у проміжному поколінні збільшується у $(1+p_c)$ раз.

Операцію схрещування у даному генетичному алгоритмі проводимо за допомогою дискретної рекомбінації: для батьківських хромосом випадковим чином вибираємо точку поділу, і вони обмінюються відсіченими частинами. Отримані два рядки є нащадками:

$$h_{i,p}^{x_i'} = \begin{cases} h_{i,p}^{x_i}, & p \le z_i \\ h_{i,p}^{x_{i+1}}, & p > z_i \end{cases}, h_{i,p}^{x_{i+1}'} = \begin{cases} h_{i,p}^{x_{i+1}}, & p \le z_i \\ h_{i,p}^{x_i}, & p > z_i \end{cases}, \quad (5)$$

$$1 \le p \le n, 1 \le z_i < r, i = \overline{1, N_p},$$

де z_i – випадкова величина, що приймає цілі значення на проміжку $[1, r)$.

До отриманого у результаті схрещення нового покоління застосовують оператор мутації. Кількість мутацій, що можна внести до покоління, визначають коефіцієнтом мутації $0 < p_m \le 1$. Ген піддається мутації, якщо для нього виконано умову $\sigma_i < p_m$. Найуживанішим способом внесення мутацій у покоління t є генерування деякого числа за законом нормально розподіленої величини $z_{i,j} \sim N\left(0, \sigma_{i,j}\right)$ та випадкової заміни генів у хромосомі таким числом $h_{i,j} \Leftarrow z_{i,j}$. При цьому величина стандартного відхилення $\sigma_{i,j}$ також оновлюється:

$$\sigma_{i,j}^{(t)} = \sigma_{i,j}^{(t-1)} exp\left[\frac{1}{\sqrt{2n}} N(0,1) + \frac{1}{\sqrt{2\sqrt{n}}} N(0,1)\right]. \quad (6)$$

Процес відбору, схрещування та мутації призводить до формування нового покоління. Далі всі дії повторюються.

Такий процес еволюції може тривати до нескінченності. Критерієм зупинки може слугувати задана кількість поколінь або сходження (*convergence*) популяції. Сходженням називають такий стан популяції, коли усі рядки популяції майже однакові та містяться в області деякого екстремума. У такій ситуації кросовер не змінює популяцію, а особини, що вийшли з цієї області унаслідок мутації, здатні вимирати, бо мають меншу пристосованість. Таким чином, сходження популяції означає, що знайдено кращий розв'язок або близький до нього.

Складність генетичного алгоритму. Розглянемо часову складність даного алгоритму (псевдокод розробленого генетичного алгоритму для пофарбування графів наведено нижче). Вхідними параметрами є: g – кодування у генах, p – кількість генів у хромосомі, q – кількість індивідів у поколінні, n – кількість ітерацій.

Оскільки для графів кодування у генах пов'язане з кількістю генів у хромосомах, то при кількісному підрахункові операцій добуток $p \times q$ замінюється на $p^{3/2}$, тобто $p \cdot q = p^{3/2}$.

generation $\leftarrow$ *GenerationInitialize(q);*	$C_1 \cdot 1$	$\times f_1(p,g)$
for i $\leftarrow \overline{1,n}$	n	$\times 1$
selected $\leftarrow$ *Selection(generation);*	$C_2 \cdot 1$	$\times f_2(p,g)$
generation $\leftarrow$ *Glossover(selected,* p_c*);*	$C_3 \cdot 1$	$\times f_3(p,g)$
generation $\leftarrow$ *Mutation(generation,* p_m*);*	$C_4 \cdot 1$	$\times f_4(f_5(p,g))$

fitness← FitnessFunction(generation); $C_5 \cdot 1 \quad \times \left[f_6\left(f_7(p,g) \right) + f_8(p,g) \right]$

generation← NextGeneration(generation, fitness); $C_6 \cdot 1 \quad \times f_9(p,g)$

end for

Нехай часова складність виконання кожної операції дорівнюватиме C_j, де j – лічильник операції у тілі циклу. Тоді кожна операція у тілі циклу програми вимагатиме $C_j \cdot n$ розрахунків. Складність виконання кожної операції залежить від вхідних параметрів p та q, що виражається через функціональну залежність $f_j(p,q)$. Крім того, деякі операції містять вкладені функції. Час виконання певних операцій, таких, як схрещення та мутації, може регулюватися користувачем. Для цього задають коефіцієнт схрещування у хромосомах $0 < p_c \le 1$ та коефіцієнт мутацій $0 < p_m \le 1$.

Відповідно обчислювальна складність наведеного алгоритму виражається такою залежністю:

$$\begin{aligned} &T(p,q,n) = n \times \sum_{j=1}^{|F|} C_j \cdot f_j(p,q), \\ &F = \{ f_1, f_2, f_3, f_4, f_5, f_6, f_7, f_8, f_9, 1 \}. \end{aligned} \tag{7}$$

При детальному аналізі кожної операції тіла циклу функції множини F мають вигляд:

$$f_1(p,q) = c_{11} + c_{12}q + c_{13}q \cdot p^{\frac{3}{2}};$$

$$f_2(p,q) = c_{21} + c_{22}q \cdot p_c;$$

$$f_3(p,q) = c_{31} + c_{33}q + c_{13}q \cdot p_c;$$

$$f_4(p,q) = c_{41} + c_{42}(q + q \cdot p_c) \cdot \left[c_{42} + c_{44}p^{\frac{3}{2}} + c_{45}p^{\frac{3}{2}} \cdot p_m \right];$$

$$f_6(p,q) = c_{61} + c_{62}(q + q \cdot p_c) \cdot \left[c_{63}p^2 \left(p^{\frac{3}{2}} - p - 1 \right) + c_{64} \left(p^{\frac{3}{2}} - p - 1 \right) + c_{65} \right];$$

$$f_8(p,q) = c_{81} + c_{82}q + c_{83}p^2;$$

$$f_9(p,q) = c_{91} + c_{92}q.$$

Виконуючи алгебраїчні перетворення кожної функції, отримаємо редукований набір

$$f_1'(p,q) = k_{11}(1 + p_c) \cdot q + k_{12} \cdot q^2;$$

$$f_2'(p,q) = k_{12} \cdot p^{\frac{3}{2}} \cdot q;$$

$$f_3'(p,q) = (1 + p_c) \left[k_{31}p^{\frac{5}{2}}q + k_{31}(1 + p_m) \cdot p^{\frac{3}{2}} \cdot q - k_{33}p^3q - k_{34}p^2q \right];$$

$$f_4'(p,q) = 1.$$

Звідси випливає, що обчислювальна складність алгоритму дорівнює

$$T(p,q,n) = n \times \sum_{j=1}^{|F'|} C_j' \cdot f_j'(p,q). \tag{8}$$

Візьмемо, що оцінка складності виконання функції дорівнює

$$n \cdot f_j'(p,q) = \Theta\left(n \cdot \varphi_j(p,q) \right). \tag{9}$$

Тоді загальна складність алгоритму дорівнюватиме

$$T(p,q,n)=max_j\Theta\left(n\cdot\varphi_j(p,q)\right). \quad (10)$$

Отже, якщо знехтувати меншими порядками, теоретична складність розробленого алгоритму дорівнює

$$T(p,q,n)=\Theta\left(p^{\frac{5}{2}}\cdot q\cdot n\right). \quad (11)$$

Генетичний алгоритм має достатньо високий ступінь складності, який визначається $p^{\frac{5}{2}}$. Причиною цього є обчислювальна складність функції $f_6(p,q)$, що виконує перевірку цільової функції, тим самим обчислюючи пристосованість кожної хромосоми.

У зв'язку з тим, що цільову функцію задано як $f(x)=\sum_{i=1}^{|V|}x_i^T\cdot Q\cdot x_i$, потрібно виконати принаймні p^2 елементарні алгебраїчні операції, аби обчислити добуток матриць та $\left(p^{\frac{3}{2}}-p-1\right)$ операцій, щоб декодувати хромосому на елементарні вектори x_i.

Слід зазначити, що алгоритм реалізує механізм випадкового пошуку, тому час знаходження субоптимального розв'язку є також деякою випадковою функцією, що залежить від кількості ітерацій n^*. Тут n^* – кількість ітерацій, після закінчення яких було знайдено субоптимальний розв'язок, що задовольняє критерій оптимальності цільової функції. Практичне застосування алгоритму показує, що величина n^* не описується законом нормального розподілення. На рисунку 1 наочно зображено розподілення кількості ітерацій, після закінчення яких було знайдено найкращий розв'язок.

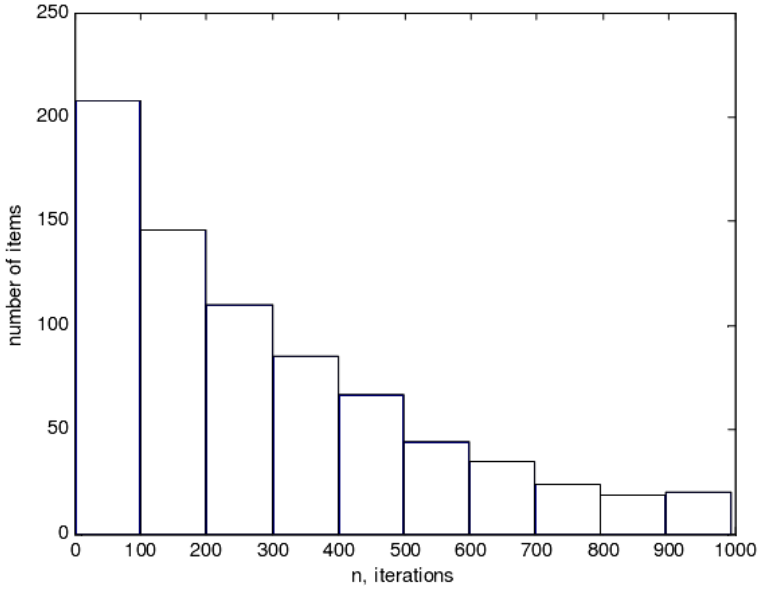

Рисунок 1. Гістограма розподілення кількості ітерацій

Дане розподілення є експоненціальним. Тоді середня кількість ітерацій, необхідна для пошуку найкращого субоптимального розв'язку, буде визначатися середнім експоненціальним розподіленням виду $P\left(n^*\right)=a\cdot e^{-bn^*}$. Середня складність виконання

алгоритму дорівнює $\Theta\left(p^{\frac{5}{2}} \cdot q \cdot \bar{n}\right)$, де $\bar{n}$ – середнє експоненціальне розподілення $P\left(n^*\right)$.

Приклад пофарбування графів за допомогою генетичного алгоритму. Розглянутий генетичний алгоритм фарбує граф з кількістю вершин $n \times n$. Яку кількість фарб потрібно задавати на вході цього алгоритму?

Задача пофарбування графів за допомогою генетичного алгоритму має на меті знайти відповіді на запитання: чи існує екстремум функції у заданому *n*-вимірному просторі (як зазначалося вище, розмір простору задається кількістю фарб). Хроматичне число графа *χ(G)*, у свою чергу, показує розмір простору, де функція все ще має екстремум, оскільки вже за межами цього простору її екстремуми не існують. У той же час задача про знаходження хроматичного числа має на меті пошук відповіді на запитання: скільки вимірів має мати простір, щоб у функції був хоча б один екстремум? Це є дві задачі, що мають різну мету. За допомогою цього генетичного алгоритму не можна знайти хроматичне число, тому що він спроектований для роботи з векторами фіксованої довжини.

Для визначення кількості фарб, яку подають на вхід, використовують метод "примірювання" простору. Застосовують критерій для оцінювання хроматичного числа заданого графа, а потім він фарбується у цьому просторі. Кількість фарб, яку беруть для початкового пофарбування такого графа, дорівнює $|C| \geq n$. Якщо алгоритм не може пофарбувати граф у задану кількість кольорів, це означає, що вибраний простір не підходить для існування глобального екстремуму. У цьому випадку підраховують кількість ребер графа, суміжні вершини яких мають однаковий колір, оскільки це суперечить умові (1). Нехай, наприклад, ця кількість дорівнює *k*. Розмір наступного простору, у якому буде фарбуватися новий граф з кількістю вершин $n \times n$, збільшується на визначену кількість таких ребер, його розмірність дорівнюватиме $n+k$. Ми можемо отримати одразу розв'язок за "перше примірювання", а можемо зробити кілька таких "примірок". Методом перебирання можна уточнювати хроматичне число.

Приклад. Пофарбувати граф *G*, що має 25 вершин ($V = 25$, тобто матриця 5×5). Розв'язували задачу за допомогою програмного продукту Pattern. За цією програмою фарбували генеровані випадковим чином графи з кількістю вершин $V = 25$ спочатку у $n = 5$ кольорів. Відповідний генерований випадковим чином граф зображено на рисунку 2.

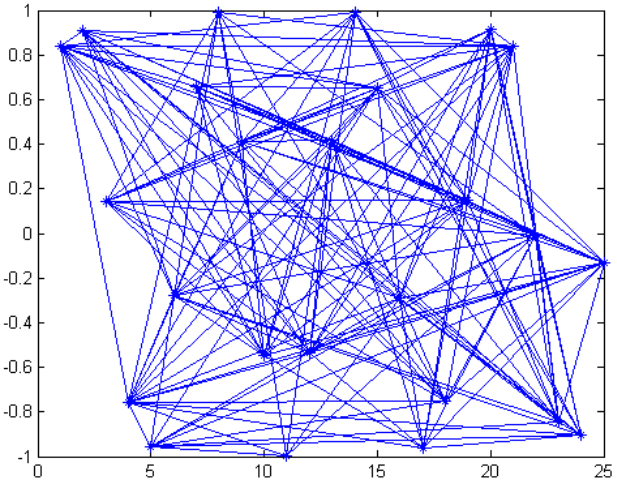

Рисунок 2. Граф G_{11}

Задача із заданими вхідними параметрами $V = 25,\ C = 5$ розв'язку не має, оскільки вершини ребер (6–13), (11–14), (8–19), (11–14), (14–23), (18–22) графа G_{11} однакового кольору ($k = 6$).

Наступний простір, у якому шукали екстремум функції, мав розмірність $n + k = 5 + 6 = 11$. У цьому просторі ми отримали пофарбування новоутвореного графа G_{12}. Методом перебирання уточнили хроматичне число. Найменша кількість кольорів, у які можна пофарбувати граф G, що має 25 вершин, дорівнює 9. Розрахунки проводили на персональній ЕОМ, що має процесор Pentium III та 256 Мб оперативної пам'яті, а також на ЕОМ з процесором AMD Turion X2 та з оперативною пам'яттю 1Гб . Час роботи програми та отримані результати наведено у таблиці 1.

Таблиця 1. Результати пофарбування графів за допомогою генетичного алгоритму

Вхідні параметри	Випадково згенерований граф	Pentium III		AMD Turion X2	
		Час, с	Результат	Час, с	Результат
$V = 25, C = 5$	G_{11}	576.35	Задача не має розв'язку	61,07	Задача не має розв'язку
$V = 25, C = 11$	G_{12}	597.63	Розв'язано	62,64	Розв'язано
$V = 25, C = 10$	G_{13}	596.65	Розв'язано	62,36	Розв'язано
$V = 25, C = 9$	G_{14}	592.71	Розв'язано	61,95	Розв'язано
$V = 25, C = 8$	G_{15}	590.45	Задача не має розв'язку	61,56	Задача не має розв'язку

Для тестування алгоритму проводили серію обчислень для пофарбування графів за допомогою програми Pattern з різною кількістю вершин, зокрема для графів, що мають $V = 49$, $V = 64$.

Висновки. Генетичний алгоритм має такі основні переваги над іншими.

Алгоритм добре працює у багатовимірних просторах, де застосовано нелінійну оптимізацію. Він дозволяє шукати розв'язок одночасно в усіх точках простору.

Техніку генетичного алгоритму можна легко комбінувати з методами 0-1 програмування, квадратичною оптимізацією, комбінаторикою.

Генетичний алгоритм уже передбачає кодування розв'язку мовою 0-1 програмування, тому декодування розв'язку із хромосоми не впливає на точність розрахунків.

Алгоритм здатен маніпулювати одночасно багатьма параметрами, цю особливість використовують у прикладних програмах.

За допомогою генетичного алгоритму достатньо швидко можна знайти прийнятний "хороший" розв'язок задачі, якщо простір пошуку, який потрібно дослідити, – великий, або функція пристосованості з шумами, або якщо задача не потребує строго знаходження глобального оптимуму.

При достатньо складному рельєфі функції пристосованості генетичний алгоритм має менше шансів збігтися до локального оптимуму та стійко функціонує на багатофункціональному ландшафті.

Генетичний алгоритм дозволяє генерувати кілька субоптимальних розв'язків за одну ітерацію і добитися стійкості алгоритму, на відміну від традиційних підходів, які дозволяють знайти лише один розв'язок протягом ітерації.

Генетичний алгоритм має і свої недоліки.

Методологію управління збіжністю ГА досі не вироблено. Необхідна збалансована збіжність ГА: швидка збіжність може призвести до сходження до неоптимального розв'язку, а повільна збіжність часто призводить до втрати знайденої найкращої особини.

Оскільки пошук ведеться одночасно по всьому простору, то може мати місце проблема збільшення часу знаходження розв'язку та можуть висуватися жорсткіші вимоги до обчислювальних пристроїв щодо обсягу оперативної пам'яті, центрального обчислювального модуля тощо.

Література

1. David S. Johnson. Worst case behavior of graph coloring algorithms / S. Johnson David // In Proceedings of 5th Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, Winnipeg, Canada. – 1974. – P. 513 – 527.
2. Генетические алгоритмы, искусственные нейронные сети и проблемы виртуальной реальности / [Вороновский Г. К., Махотило К. В., Петрашов С. М., Сергеев С. А.] – Х: Основа, 1997. – С. 107 – 112.
3. Mertz P. Genetic algorithms for binary quadratic programming / *P. Mertz, B. Freisleben* // In Proceedings of the 1999 International Genetic and Evolutionary Computational Conference (GECCO'99), Morgan Kaufmann. – 1999.
4. Скрильник І.І. Розв'язання NP-складних задач як пофарбування теоретико-графових моделей за допомогою генетичного алгоритму / І.І. Скрильник // Вісник Тернопільського державного університету: (фізико-математичні науки): наукові статті. – Тернопіль: ТДТУ, 2007. – № 2. – С. 166 – 176.
5. Скрильник І.І. Застосування генетичного алгоритму для пофарбування n-дольних графів / І.І. Скрильник // Стратегические вопросы мировой науки – 2007: Междунар. науч.-практ. конфер., 15-28 февр. 2007 г.: тезисы докл.: (Математика. Физика. Современные информационные технологии). – Днепропетровск: Наука и образование, 2007. – Т. 8. – С. 17 – 20.