Література

1. Григорків В.С., Білоскурський Р.Р. Деякі оптимізаційні моделі стохастичного еколого-економічного міжгалузевого балансу // Економічна кібернетика. – 2003. - №5-6 (23-24). – С. 18-24.
2. Основы теории оптимального управления / Под ред. В.Ф. Кротова. – М.: Знание, 1990. – 430 с.
3. Колемаев В.А. Математическая экономика. – М.: ЮНИТИ, 1998. – 240 с.

Одержано 16.05.2006 р.

УДК 519.876.5

І. Скрильник

Полтавський національний технічний університет імені Юрія Кондратюка

РОЗВ'ЯЗАННЯ NP-СКЛАДНИХ ЗАДАЧ ЯК ПОФАРБУВАННЯ ТЕОРЕТИКО-ГРАФОВИХ МОДЕЛЕЙ ЗА ДОПОМОГОЮ ГЕНЕТИЧНОГО АЛГОРИТМУ

У статті розглядається проблема пофарбування графів за допомогою генетичного алгоритму. Показано, що велика кількість NP-складних задач зводяться до проблеми пофарбування теоретико-графових моделей. На основі 0-1 програмування та лінійно-квадратичної оптимізації розроблено комбінований алгоритм, який дозволяє ефективно пофарбовувати графи зі $6 \cdot 10^{306}$ вершинами у мінімальну кількість кольорів.

I. Skrylnyk

THE NP-HARD PROBLEMS SOLVING AS A GRAPH COLORING TASK USING THE GENETIC ALGORITHM

In this paper the graph coloring problem using the genetic algorithm is investigated. It is shown that a great majority of NP-hard problems can be converged to graph coloring problem. The combined algorithm is elaborated, basing on the 0-1 programming and linear-quadratic optimization. The designed algorithm enables to color graphs up to $6 \cdot 10^{306}$ vertex in the minimal number of colors.

Умовні позначення

$\boldsymbol{V}$ — множина вершин графа;
$\boldsymbol{E}$ — множина ребер графа;
$\boldsymbol{G}$ — граф;
$\boldsymbol{C}$ — множина кольорів;
$\boldsymbol{c_i}$ — колір;
$\boldsymbol{v_i}$ — вершина графа;
$\boldsymbol{\chi(G)}$ — хроматичне число графа;
$\boldsymbol{\Re(C)}$ — пофарбування графа;
$\boldsymbol{P}$ — популяція індивідів;
$\boldsymbol{h_i}$ — ген;
$\boldsymbol{\sigma_I}$ — стандартне відхилення генів;
$\boldsymbol{p_c}$ — коефіцієнт схрещування.

Вступ

Проблема пофарбування теоретико-графових моделей посідає важливе місце у теорії графів та має особливе прикладне значення. Існує велика кількість задач, для

яких може бути побудована еквівалентна модель у вигляді графа, а пофарбування отриманого графа визначає у кінцевому випадку їх розв'язок. Серед цих задач найбільш розповсюдженими є: машинне генерування розкладів руху міського транспорту, складання розкладів проведення занять у навчальних закладах [1], планування процесорного часу для виконання комп'ютерних програм для операційних систем [2, 3], автоматичне розділення частотних смуг для радіопристроїв [4], розподілення ресурсів декількома напрямами для виконання встановлених робіт [5], розроблення алгоритмів сканування у системах захисту інформації [6]. Інший ряд задач також може бути ефективно розв'язаний за допомогою пофарбування теоретико-графових моделей, наприклад, у біометрії, теорії кодування, плануванні експерименту, розробленні запам'ятовуючих пристроїв, у радіоелектроніці [7, 8] тощо.

Наведені задачі відносять до NP-складних задач [9], тому вони вимагають спеціальних методів розв′язання. Час, необхідний для вирішення даного класу задач, може бути виражений деякою функцією від "довжини входу". Виділяють задачі, що можуть бути розв′язані за допомогою поліноміальних алгоритмів, а також задачі, які розв′язуються за субекспоненціальний, експоненціальний та факторіальний час. Зі збільшенням довжини входу, час, потрібний для розв'язання факторіальної за складністю задачі, завжди випередить час, затрачений на розв'язання експоненціальної за складністю задачі. У свою чергу, поліноміальні алгоритми вимагають менше часу, ніж факторіальні та експоненціальні задачі. Це твердження виконується, якщо довжина входу прямує до нескінченності, хоча при невеликій довжині входу ця ієрархія може порушуватись. Тому слушним є розроблення поліноміальних алгоритмів замість експоненціальних. Для деяких типів задач неможливо розробити поліноміальний алгоритм. Наприклад, відшукуючи перестановку, при якій досягається максимум функції загального вигляду, потрібно перебрати всі перестановки, тобто знайти їх факторіал. Але, вирішуючи задачу про розподілення *N* ресурсів на виконання *N* робіт, де затрати при розподіленні *i*-ого ресурсу на *j*-ту роботу рівні *A[i,j]*, можна переконатися, що вона є поліноміальною та розв'язується за поліноміальний час.

Означення 1. Назвемо *мовою* множину слів *x* над алфавітом *S*. Мова S_1 може бути зведена до мови S_2, якщо існує функція $f : S \to S$, котра може бути знайдена за поліноміальний час та виконується умова $f(x) \in S_2$ тоді і тільки тоді, коли $x \in S_1$.

Означення 2. Мова *S* називається *NP-складною*, якщо будь-яка мова із класу NP зводиться до нього. Мову *S* називають *NP-повною*, якщо вона є NP-складною та належить до класу NP.

Означення 3. Пофарбуванням графа $G = (V, E)$, утвореного набором вершин *V* та ребер *E*, називають таку *множину кольорів* $C : V \to \Re(C)$, для якої дві суміжні вершини $C(\upsilon_i) \neq C(\upsilon_j)$ для $i \neq j$ пофарбовані у різні кольори, а також $\exists i, j \,/\, C(\upsilon_i) \cap C(\upsilon_j) \neq \varnothing$, $V = \{\upsilon_1, \upsilon_2, \ldots \upsilon_n\}$.

Тобто, якщо заданий граф $G = (V, E)$, то можна виділити такий набір вершин *V*, пофарбований у колір $c_i \in C$, що виключає наявність двох суміжних вершин. Найменше число кольорів $\chi(G)$, що потрібне для пофарбування, називають *хроматичним числом графа G*. У роботах [12, 13, 14] показано, що задача пофарбування графа $G = (V, E)$ є NP-складною.

Ця проблема може бути розв'язана із застосуванням поліноміальних алгоритмів, якщо граф *G* належить до класу досконалих, *t*-досконалих, циклічних графів та їх доповнень або графів з непарними довгими циклами [10, 11]. Для точного пофарбування графів було запропоновано багато комбінаторних оптимізаційних методів. Широкого розповсюдження набула техніка пофарбування, запропонована Р. Караганом та П. М. Пардалосом [12]. Заслуговують уваги розроблені методи 0-1 програмування Л. Бабелем і Г. Тінгофером [13, 14], Е. Баласом, Х. Ксю і Чанг Сунг Йю [15, 16], К. Маніно і А. Сасано [17], а також Дж.Л. Немгаузером [18]. Відомі також

алгоритми пофарбування напівозначеного програмування із використанням розтинів графів, що описані в роботах [19, 20]. Однак, ефективність наведених методів знаходиться в межах пофарбування графів із 600 – 700 вершинами, оскільки всі вони мають експоненціальну складність. Тому зі збільшенням кількості вершин різко зростає час пошуку розв'язку.

На даний час розроблені евристичні субоптимальні алгоритми [11, 21, 22, 23], що дають змогу пофарбувати сильно зв'язані графи з великою кількістю вершин. До даного класу алгоритмів відносять також генетичні алгоритми. Перевагою останніх є можливість генерування субоптимального розв'язку при будь-яких накладених обмеженнях. Але їх недоліком є достатньо великий час пошуку. Крім цього, якість розв'язку залежить від обраного критерію оптимальності. Тому для підвищення ефективності даної евристичної техніки пошуку та досягнення оптимальності пропонується застосувати комбінований метод, заснований на 0-1 програмуванні та лінійно-квадратичній оптимізації [24].

Формулювання стратегії пошуку оптимального розв'язку

Останнім часом алгоритми оптимізації отримали широке розповсюдження у промисловості завдяки популяризації такого напряму, як комп'ютерна алгебра, яка має на меті впровадження теоретичних математичних розробок у промислові обчислювальні пристрої. З наукової точки зору, розроблення алгоритмів оптимізації є важливою проблемою, оскільки вони дають змогу знайти субоптимальний розв'язок NP-складних задач у тих галузях, де не існує аналітичних методів.

Покажемо, що задача пофарбування графа $G=(V,E)$ є задачею оптимізації, тобто відповідно до означення 3 необхідно присвоїти фарби із набору $C=\{c_1,c_2,\ldots c_m\}$ вершинам $V=\{\upsilon_1,\upsilon_2,\ldots\upsilon_n\}$, $m\le n$.

Лема 1. Для будь-якого графа G пофарбування $\Re(C)$ є оптимальним, якщо існує мінімум деякої функції $x^*=\min\limits_x f(x)$ та виконується обмеження $x^*\circ C=\Re(C)$.

Доведення. Розглянемо проблему пофарбування графа з точки зору 0-1 програмування. Нехай $x_{i,h},\in V,1\le h\le|C|$ бінарна змінна, для якої "1" означає, що i-та вершина пофарбована у h-ий колір із набору C, або "0" у протилежному разі. Оскільки кожна вершина графа G повинна бути пофарбована, отримуємо

$$\sum_{j=1}^{K} x_{i,j}=1,\qquad i=1,\ldots,n, \tag{1}$$

де K – кількість кольорів, у які має бути пофарбований граф G; n – кількість вершин графа G. Згідно з означенням 3, вимога про пофарбування двох суміжних вершин (i,j) у різні кольори на мові 0-1 програмування формулюється так:

$$x_{i,h}+x_{j,h}\le 1\qquad h=1,\ldots,K. \tag{2}$$

Отже, граф G може бути пофарбований у K кольорів, якщо існує розв'язок системи, складених із рівнянь (1) та (2). Оптимальність розв'язку випливає із наступного.

Нехай на певному обмеженому просторі $\Re(C)$ задано квадратичну функцію $f(x)=xQx^T, x\in\{0,1\}$. Розглядаючи рівняння (1) та (2) як обмеження, отримуємо задачу лінійно-квадратичної оптимізації у наступному вигляді:

$$\begin{aligned} &min\ f(x) = xQx^T; \\ &x^* = min\ f(x); \\ &x_{i,h} + x_{j,h} \le 1; (i,j) = 1,\ldots,n; \\ &\sum_{h=1}^{K} x_{i,h} = 1; h = 1,\ldots,K. \end{aligned} \quad (3)$$

У даному випадку Q – деяка позитивна ненульова матриця. Метою розв'язання оптимізаційної задачі (3) є відшукання бінарного вектора x^*, такого що $x^* \circ C = \Re(C)$.■

Задача (3) може бути ефективно розв'язана при застосуванні евристичних методів. Одним із альтернативних підходів є застосування генетичних алгоритмів [25]. На практиці більшість оптимізаційних проблем мають декілька значимих локальних оптимумів, а простір для пошуку глобального екстремуму може бути занадто великим. Отже, точне знаходження глобального екстремуму неможливо відшукати за відведений машинний час. До того ж, поставлена оптимізаційна задача може мати декілька суперечливих критеріїв або динамічних складових, що впливають на місцезнаходження глобального екстремуму. Тому для швидкого розв'язання поставленої задачі детерміновані методи, такі, як алгоритм найшвидшого спуску, не можуть застосовуватися й не є ефективними, оскільки для них існує проблема застрягання на локальному екстремумі, замість відшукання глобального. З цієї точки зору, генетичні алгоритми представляють найбільший інтерес, оскільки дану техніку можна легко комбінувати із методами 0-1 програмування, лінійно-квадратичною оптимізацією, комбінаторикою. Крім цього, генетичні алгоритми дозволяють вести пошук глобального екстремуму одночасно по всьому простору невідомих параметрів або обстежувати декілька зон, де існує велика імовірність знаходження оптимуму. У процесі пошуку генетичний алгоритм дозволяє генерувати декілька субоптимальних розв'язків за одну ітерацію і добитися робастності алгоритму, на відміну від традиційних підходів, які дозволяють лише знайти один розв'язок протягом ітерації.

Поряд із перерахованими перевагами природно постають недоліки генетичних алгоритмів. Оскільки пошук ведеться одночасно по всьому простору, тому може мати місце проблема збільшення часу знаходження рішення та висуватися більш жорсткіші вимоги до обчислювальних пристроїв щодо об'єму оперативної пам'яті, центрального обчислювального модуля тощо. Отже, застосування генетичних алгоритмів в умовах реального часу може бути ускладненим.

У загальному випадку генетичний алгоритм представляє собою набір ітеративних, стохастичних процедур, що наслідують принципи концепції Ч. Дарвіна у теорії еволюції живих систем. Генетичний алгоритм відтворює процес еволюції у популяції індивідів, метою якого є знаходження найбільш близького до оптимального розв'язку. Множина усіх розв'язків називається популяцією P, що видозмінюється від одного покоління до іншого. Відповідно, кожна популяція складається із індивідів $P = \{x_1, x_2, \ldots, x_\ell\}$, які називаються хромосомами та представляють собою закодовані розв'язки (гени), об'єднані в ланцюг $x = (h_1, h_2, \ldots, h_r) \in R^r$. Отже, кожен індивід утворюється із двох компонентів (h_i, σ_i), а саме r-ої кількості генів h_i та r стандартних відхилень цих генів $\sigma_i = [\sigma_{j,1}, \sigma_{j,2}, \ldots, \sigma_{j,r}]^T$.

У циклі симуляції алгоритму важливу роль відіграють такі операції: схрещування, мутація, відбір. Загальна структура генетичного алгоритму наведена на рис. 1.

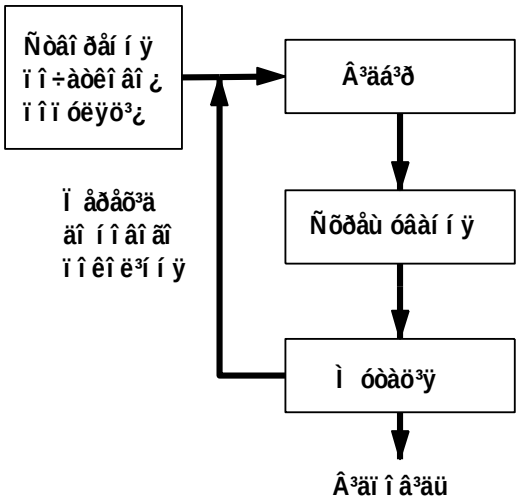

Рисунок 1- Узагальнена структура генетичного алгоритму.

Процедура створення нової популяції визначає початкову точку пошуку глобального екстремуму. Початкова популяція може бути створена декількома шляхами. На практиці найбільш уживаним методом утворення популяції є випадкове генерування хромосом за законом нормально розподіленої величини. При створенні початкової популяції намагаються покрити якомога більший об'єм простору пошуку глобального екстремуму, а також урізноманітнити генетичний матеріал у хромосомах. Іншим підходом визнано генерування деякої сітки пошуку оптимуму через рівномірні кроки. Знаючи імовірнісну область знаходження оптимуму, можна згенерувати набір субоптимальних розв'язків одним із описаних вище методом, але в такому випадку потрібно мати додаткові знання про знаходження оптимуму, які не завжди можуть бути доступними. Проте в останньому випадку час пошуку оптимуму може бути значно скорочений.

Відбір є особливо важливою процедурою в циклі генетичного алгоритму, оскільки він вилучає із популяції індивідів із низьким рівнем пристосованості, тобто залишаються лише ті роз'язки, які є найбільш близькими до оптимального. Оператор відбору визначає, яким чином оновлюється популяція при переході від одного покоління до іншого. В загальному випадку, процедура відбору може оновлювати лише певну частину популяції або повністю все покоління. Остання передбачає застосування стохастичного підходу, на відміну від часткового оновлення покоління, де використовується детермінований підхід. При оновленні всього покоління поточний субоптимальний розв'язок може втрачатися, що збільшує час пошуку оптимуму. Тому в детермінованих процедурах відбору, як правило, наступне покоління формується за рахунок генетичного матеріалу поточного субоптимального розв'язку. Важливим аспектом у розробленні процедури відбору є так званий "тиск" відбору, що керує індивідуальним показником пристосованості кожної хромосоми у популяції. Якщо величина "тиску" занадто велика, це може призвести до збігання алгоритму у невелику зону простору R^r та застрягання на локальному екстремумі. Якщо величина "тиску" є малою, то алгоритм має повільну збіжність.

Операція схрещування дає змогу отримувати нові рекомбіновані розв'язки. Схрещування може здійснюватися в цілому поколінні або між відібраними парами хромосом-батьків. В результаті схрещування отримуються хромосоми-нащадки, які мають генетичні ознаки своїх батьків. Традиційно використовуються такі оператори схрещування: дискретна рекомбінація, діагональна рекомбінація, геометрична рекомбінація. Формально оператори схрещування для двох відібраних хромосом-батьків F і M можна описати таким чином:

$$h_i' = \begin{Bmatrix} h_{F,i} \text{ АБО } h_{M,i} & \text{дискретна рекомбінація} \\ \dfrac{h_{F,i} + h_{M,i}}{2} & \text{діагональна рекомбінація} \end{Bmatrix}; \tag{4}$$

$$\sigma_i' = \begin{Bmatrix} \sigma_{F,i} \text{ АБО } \sigma_{M,i} & \text{дискретна рекомбінація} \\ \dfrac{\sigma_{F,i} + \sigma_{M,i}}{2} & \text{діагональна рекомбінація} \\ \sqrt{(\sigma_{F,i}\,\sigma_{M,i})} & \text{геометрична рекомбінація} \end{Bmatrix}. \tag{5}$$

Для цілого покоління, розмір якого становить N_p:

$$h_i' = \sum_{k=1}^{N_p} h_{k,i} / N_p ; \tag{6}$$

$$\sigma_i' = \sum_{k=1}^{N_p} \sigma_{k,i} / N_p . \tag{7}$$

Широко відомою є думка про те, що в загальному випадку після проходження певної кількості ітерацій, покоління буде прямувати або до глобального екстремуму, або до локального, незалежно від обраних процедур схрещування та відбору. Для уникнення застрягання алгоритму на локальному оптимумі, покращенні ефективності та швидкості роботи алгоритму застосовують процедуру мутації. Крім цього, її призначення полягає у постачанні нового "генетичного матеріалу" і забезпеченні пошуку глобального екстремуму по всьому простору. Найбільш уживаним способом внесення мутацій у покоління t є генерування деякого числа за законом нормально розподіленої величини $z_{i,j} \sim N(0,\sigma_{i,j})$ та випадкової заміни генів у хромосомі цим числом:

$$h_{i,j} \Leftarrow z_{i,j} . \tag{8}$$

При цьому величина стандартного відхилення $\sigma_{i,j}$ також оновлюється:

$$\sigma_{i,j}^{(t)} = \sigma_{i,j}^{(t-1)} exp\left[\frac{1}{\sqrt{2n}} N(0,1) + \frac{1}{\sqrt{2\sqrt{n}}} N(0,1) \right]. \tag{9}$$

Оцінювання пристосованості кожного індивіда у поколінні відбувається за допомогою функції відповідності. В нашому випадку будемо застосовувати квадратичну функцію виду

$$f(x) = x Q x^T . \tag{10}$$

Будемо вважати, що чим менше значення функції відповідності $f(x)$, тим краще пристосований індивід, тобто тим ближче знаходиться отриманий поточний розв'язок до оптимуму. В кінцевому випадку необхідно отримати хромосому x^* із найменшим значенням функції відповідності, для якої виконуються обмеження (3). Таким чином, відповідно до леми 1 знайдене пофарбування графа буде оптимальним.

Реалізація алгоритму

Задамо спосіб кодування розв'язків у хромосомах. Нехай маємо деяку хромосому *x*, утворену генами $(h_1, h_2, \ldots, h_r)$. Поставимо у відповідність кожному гену вершину графа $G = (V, E)$, $r = |V|$. Відповідно до рівняння (1) вважаємо, що вершина пофарбована у *i*-ий колір із набору фарб *K*, якщо її значення дорівнює "1". Тоді кожен окремий ген $h_{i,j}$ із урахуванням стандартного відхилення $\sigma_i = N(0,1)$ буде кодуватися наступним чином:

$$h_{i,j} = \left(Random\left(\sum_{j=1}^{K} \upsilon_j = 1 \right), \sigma_i \right), \tag{11}$$

де $Random(\bullet)$ позначає операцію генерування випадкової бінарної величини із множини значень $\{0,1\}$. Отже, для графа G із кількістю вершин r пофарбування буде визначатися значеннями генів у хромосомі x:

$$x = \{h_{i,j}\} \in \{0,1\}; \quad i = 1,\ldots,r;\ j = 1,\ldots,K. \tag{12}$$

Крок 1. Для створення початкової популяції задамо її розмір у кількості N_p індивідів

$$P = \{x_1, x_2, \ldots, x_{N_p}\} \in \{0,1\}. \tag{13}$$

Таким чином, для створення початкової популяції необхідно виконати операцію (12) N_p разів.

Крок 2. Відберемо із новоутвореного покоління хромосом-батьків у кількості $N_p \cdot p_c$ штук, які будуть брати участь у схрещуванні. Тут $0 < p_c \le 1$ є коефіцієнтом схрещування та показує, який процент хромосом у популяції може взяти участь у схрещуванні. Для цього для кожної хромосоми x_i із покоління P поставимо у відповідність деяке число $\delta_i, i = 1,\ldots,N_p$

$$\delta_i = \frac{\overline{f}(x_i)}{\sum_{k=1}^{N_p} \overline{f}(x_k)}, \quad \overline{f}(x_i) = \max_{k=1,N_p} f(x_i) - f(x_k); \tag{14}$$

та згенеруємо випадкову величину $z_i = Random([0,1])$. Хромосома x_i вважається відібраною, якщо $z_i \ge \delta_i$.

Функція відповідності $f(x)$ обчислюється відповідно до рівняння (10). При цьому матриця Q у рівнянні (10) може бути визначена як матриця суміжності графа G, у якої на головній діагоналі розташовані числа $diag\{a_1, a_2, \ldots, a_r\}$, такі що $det\, Q > 0$.

Крок 3. Операцію схрещування будемо проводити за допомогою дискретної рекомбінації із рівнянь (4, 5). Схрещування здійснюється шляхом обміну генів відносно n точок

$$h_{i,p}^{x_i'} = \begin{cases} h_{i,p}^{x_i}, & p \le z_i \\ h_{i,p}^{x_{i+1}}, & p > z_i \end{cases}, h_{i,p}^{x_{i+1}'} = \begin{cases} h_{i,p}^{x_{i+1}}, & p \le z_i \\ h_{i,p}^{x_i}, & p > z_i \end{cases}, \tag{15}$$
$$1 \le p \le n, 1 \le z_i < r, i = \overline{1,N_p};$$

де z_i випадкова величина, що приймає цілі значення на проміжку $[1,r)$. У результаті виконання операції (15) утворюється покоління розміром $N_p + N_p \cdot p_c$ індивідів.

Крок 4. Відповідно до структури алгоритму, зображеної на рис. 1, кожна хромосома нового покоління може бути піддана мутації. Кількість мутацій, що можуть бути внесені до покоління, визначаються коефіцієнтом мутації $0 < p_m \le 1$. Ген піддається мутації, якщо для нього виконується умова $\sigma_i < p_m$. Формально дана процедура відповідає рівнянню (8). Стандартне відхилення σ_i оновлюється за допомогою процедури (9).

Крок 5. Для отриманого мутованого покоління проводиться перевірка на пристосованість кожного індивіда. З цією метою для кожної хромосоми обчислюються значення функції відповідності (10) та відкидаються $N_p \cdot p_c$ хромосом, які мають найгірший показник.

Крок 6. Якщо у новоутвореному поколінні знаходиться хромосома, для якої справедливими є умови (3), то алгоритм припиняється, інакше пошук оптимуму продовжується із поверненням до кроку 2.

Симулювання алгоритму

Розглянемо чисельно роботу алгоритму на прикладі. Нехай потрібно знайти оптимальне пофарбування графа, зображеного на рис. 2. Кількість кольорів, у які потрібно пофарбувати даний граф, дорівнюють чотирьом $K = 4$. Позначимо кольори набору фарб *C* числами від 1 до 4, тобто $C = \{1,2,3,4\}$. Як видно із рис. 2, даний граф має шістнадцять вершин та є сильнозв'язаним. Визначимо кодування розв'язків у хромосомах. Для скорочення часу пошуку оптимального рішення умовно розіб'ємо граф на рис. 2 на чотири долі, які включають в себе чотири вершини та пофарбуємо ці долі почергово у кольори 1, 2, 3, 4 із набору фарб *C*.

Такий підхід гарантує використання всіх фарб із набору *C*. Отримане пофарбування графа є субоптимальним. Відповідно до (11) будь-який ген у хромосомах може мати наступні значення:

$$h_{i,j} = \begin{cases} (1\ 0\ 0\ 0), \sigma_i \\ (0\ 1\ 0\ 0), \sigma_i \\ (0\ 0\ 1\ 0), \sigma_i \\ (0\ 0\ 0\ 1), \sigma_i \end{cases} \quad (16)$$

Оскільки ми маємо граф із шістнадцятьма вершинами, де кожна вершина кодується послідовністю із чотирьох знаків $\{0,1\}$, то хромосома буде складатися із $r = 16$ генів.

Задамо розмір популяції в кількості $N_p = 10$ індивідів та згідно з кроком 1 алгоритму, використовуючи рівняння (12, 13), згенеруємо початкову популяцію.

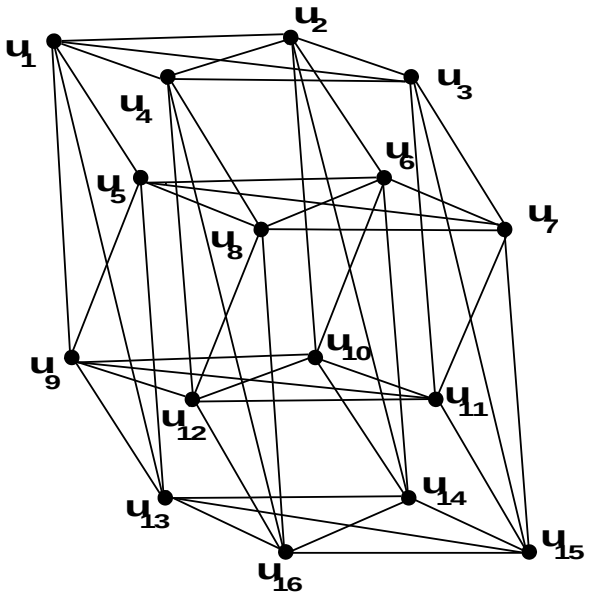

Рисунок 2 - Приклад графа для тестування алгоритму.

Для реалізації операції відбору та схрещування задамо коефіцієнт схрещування $p_c = 0,5$, тобто 50 % хромосом із утвореної популяції зможуть рекомбінувати. Обчислимо для новоутвореного покоління числа δ_i. Для цього спочатку потрібно знайти значення функції відповідності $f(x)$. Матриця *Q*, утворена матрицею суміжності графа та числами $diag\{a_1, a_2, \ldots, a_r\}$ на головній діагоналі, має наступний вигляд:

Таблиця 1 - Матриця Q для графа, зображеного на рис. 2

	υ_1	υ_2	υ_3	υ_4	υ_5	υ_6	υ_7	υ_8	υ_9	υ_{10}	υ_{11}	υ_{12}	υ_{13}	υ_{14}	υ_{15}	υ_{16}
υ_1	4	1	1	1	1	0	0	0	1	0	0	0	1	0	0	0
υ_2	1	4	1	1	0	1	0	0	0	1	0	0	0	1	0	0
υ_3	1	1	4	1	0	0	1	0	0	0	1	0	0	0	1	0
υ_4	1	1	1	4	0	0	0	1	0	0	0	1	0	0	0	1
υ_5	1	0	0	0	4	1	1	1	1	0	0	0	1	0	0	0
υ_6	0	1	0	0	1	4	1	1	0	1	0	0	0	1	0	0
υ_7	0	0	1	0	1	1	4	1	0	0	1	0	0	0	1	0
υ_8	0	0	0	1	1	1	1	4	0	0	0	1	0	0	0	1
υ_9	1	0	0	0	1	0	0	0	4	1	1	1	1	0	0	0
υ_{10}	0	1	0	0	0	1	0	0	1	4	1	1	0	1	0	0
υ_{11}	0	0	1	0	0	0	1	0	1	1	4	1	0	0	1	0
υ_{12}	0	0	0	1	0	0	0	1	1	1	1	4	0	0	0	1
υ_{13}	1	0	0	0	1	0	0	0	1	0	0	0	4	1	1	1
υ_{14}	0	1	0	0	0	1	0	0	0	1	0	0	1	4	1	1
υ_{15}	0	0	1	0	0	0	1	0	0	0	1	0	1	1	4	1
υ_{16}	0	0	0	1	0	0	0	1	0	0	0	1	1	1	1	4

Визначник $det\, Q = 278{,}69 \cdot 10^6$. Значення функцій відповідності та δ_i для хромосом наведені у таблиці 2 нижче.

Таблиця 2 - Значення $f(x_i)$ та δ_i для новоутвореного покоління

Хромосома	$f(x_i)$	δ_i	**Хромосома**	$f(x_i)$	δ_i
x_1	73	0,145	x_6	73	0,145
x_2	75	0,109	x_7	76	0,091
x_3	81	0	x_8	75	0,109
x_4	76	0,091	x_9	73	0,145
x_5	73	0,145	x_{10}	80	0,018

Із таблиці 2 видно, що хромосоми x_3, x_4, x_7, x_8, x_{10} не будуть відібрані для схрещування або імовірність їх відбору є невеликою, оскільки вони мають найбільші значення функції відповідності.

Кількість мутацій у популяції визначаються коефіцієнтом мутацій, який приймемо таким, що рівний $p_m = 0{,}05$, тобто лише 5 % генів у кожній хромосомі можуть бути мутовані. Мутації в популяції відбуваються відповідно до кроку 5 алгоритму. Значення функцій відповідності для покоління $N_p + N_p \cdot p_c$ після рекомбінації та мутації наведені у таблиці 3.

Таблиця 3 - Значення функції відповідності для покоління $N_p + N_p \cdot p_c$

Хромосома	$f(x_i)$	**Хромосома**	$f(x_i)$	**Хромосома**	$f(x_i)$
x_1	76	x_6	73	x_{11}	75
x_2	80	x_7	73	x_{12}	75
x_3	81	x_8	70	x_{13}	80
x_4	76	x_9	70	x_{14}	81
x_5	73	x_{10}	75	x_{15}	70

У цілому алгоритм тестувався протягом 100 поколінь. Значення функції належності для найкращої хромосоми i-ого покоління, зображено на рис. 3.

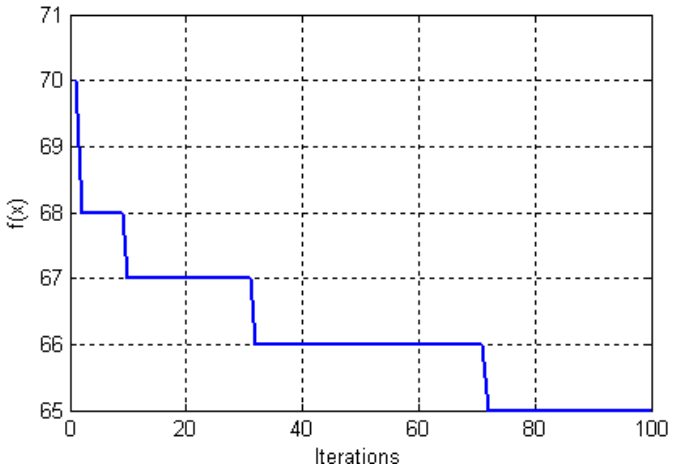

Рисунок 3 - Значення функції належності для найкращої хромосоми.

Як видно із рис. 3, найкращий розв'язок був знайдений на 72 ітерації алгоритму. При цьому x^* дорівнює

$$x^* = \left|\begin{array}{cccc|cccc|cccc|cccc} 0&0&1&0 & 0&0&0&1 & 1&0&0&0 & 0&1&0&0 \\ 0&0&0&1 & 1&0&0&0 & 0&1&0&0 & 0&0&1&0 \\ 0&1&0&0 & 0&0&1&0 & 0&0&0&1 & 1&0&0&0 \\ 1&0&0&0 & 0&1&0&0 & 0&0&1&0 & 0&0&0&1 \end{array}\right|.$$

Отже, пофарбування графа у вигляді матриці можна визначити наступним чином:

$$x^* \cdot C = \Re(C);$$

$$x^* \cdot \begin{bmatrix} 1 & 2 & 3 & 4 \end{bmatrix} = \begin{bmatrix} 4 & 3 & 1 & 2 \\ 2 & 4 & 3 & 1 \\ 1 & 2 & 4 & 3 \\ 3 & 1 & 2 & 4 \end{bmatrix}.$$

Оптимально пофарбований граф зображений на рис. 3.

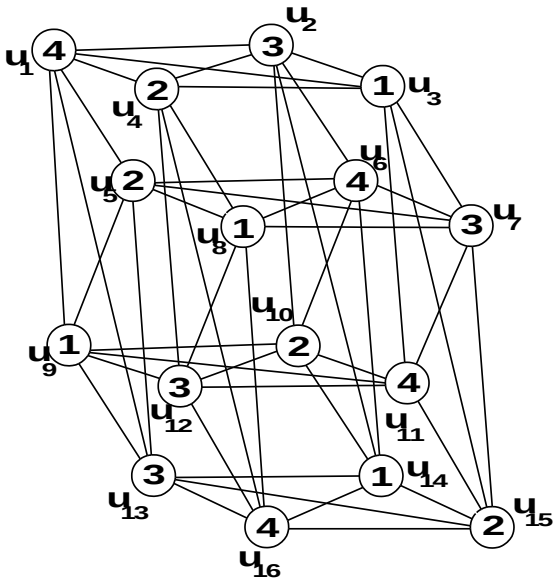

Рисунок 4 - Граф, оптимально пофарбований за допомогою генетичного алгоритму.

Висновок

Таким чином, розроблений алгоритм дозволяє роз'язати NP-складні задачі пофарбування зв'язаних графів із великою кількістю вершин. Отримане пофарбування графа є оптимальним з точки зору лінійно-квадратичної оптимізації. Використовуючи

комбінований підхід, а саме 0-1 програмування та квадратичну функцію, час пошуку оптимального розв'язку значно скорочено. Теоретично розроблений алгоритм дозволяє знаходити оптимальне пофарбування графів із нескінченною кількістю вершин V_k. Але практично сучасні можливості обчислювальних пристроїв дозволяють зберігати числа розміром 64 Байт, що становить $1{,}8 \cdot 10^{307}$, тобто при мінімальній кількості кольорів, що дорівнює три, даний алгоритм зможе пофарбувати граф із $6 \cdot 10^{306}$ вершинами.

Література

1. De Wera D. An introduction to timetabling // European Journal of Operations Research. – 1985. - № 19. – P. 151 – 162.
2. Fred C. Chow, John L. Hennesy. Register allocation by priority-based coloring. In Proceedings of the ACM SYGPLAN 84 Symposium on Compiler Construction. - New York: ACM, 1984. – P. 222 – 223.
3. Fred C. Chow, John L. Hennesy. The priority-based coloring approach to register allocation // ACM Transactions on Programming Languages and Systems. – 1990. – № 12(4). – P. 501 – 536.
4. Gamst A. Some lower bounds for a class of frequency assignment problems // IEEE Transactions of Vehicular Echnology. – 1986. - № 35(1). – P. 8 – 14.
5. Arkin M., Silverberg B. Scheduling jobs with fixed start and end times // Discrete Applied Mathematics. – 1987. – № 18.
6. Mikhail J. Atallan. Algorithms and Theory of Computation Handbook. U.S.A, New York: CRC Press, 1999.
7. Grotschel M., Junger M., Reinelt G. An application of combinatorial optimization to statistical physics and Circuit layout design // Operations Research. – 1988. - № 36(3). – P. 493 – 513.
8. Agarwal S., Belongie S. On the non-optimality of four color coding of image partitions // IEEE Proceedings of Int. Conf. Image Processing. – 2002.
9. Алексеев В.А., Носов В.А. NP-полные задачи и их полиномиальные варианты. Обзор. Обозрение промышленной и прикладной математики. - 1997. - т. 4, вып. 2. - С.165 – 193.
10. Michael R. Garey, David S. Johnson. Computers and Intractability: A Guide to NP-completeness, San Francisco: W.H. Freeman, CA. – 1979.
11. Craig A. Morgenstern, Harry D. Shapiro. Coloration neighborhood structures for general graph coloring. In Proceedings of the First Annual ACM-SIAM Symposium on Discrete Algorithms, San Francisco, Jan. 1990. Society for Industrial and Applied Mathematics, Philadelphia. – 1990.
12. Carrahan R., Pardalos P. M. An exact algorithm for the maximum clique problem // Operations Research Letters. – 1990. - № 9. – P. 375 – 382.
13. Babel L. Finding maximum cliques in arbitrary and special graphs // Computing. – 1991. - № 46. – P. 321 – 341.
14. Babel L., Tinhofer G. A branch and bound algorithm for the maximum clique problem // Journal of Global Optimization. – 1994. - № 4.
15. Balas E., J. Xue. Minimum weighted coloring of triangulated graphs, with application to maximum weight vertex packing and clique finding in arbitrary graphs // SIAM Journal on Computing. – 1991. - № 20(2). – P. 209 – 221.
16. Balas E., Chang Sung Yu. Finding a maximum clique in arbitrary graph // SIAM Journal on Computing. – 1986. - № 15(4). – P. 1054 – 1068.
17. Mannino C., Sassano A. An exact algorithm for the maximum cardinality stable set problem // Networks pages, ftp://dimacs.rutgers.edu/pub/challenge/graph. - 1993.
18. Nemhauser G. L., Sigismondi G. L. A strong cutting plane. Branch and bound algorithm for node packing // Journal of Operational Research Society. – 1992. - № 43(5).
19. de Klerk E., Pasechnik D. V. On approximate graph colouring and MAX-*k*-cut algorithms based on the ϑ-function // Journal of Combinatorial Optimization. – 2004. - № 8(2004).
20. Kochenberger G.A., Glover F., Alidaee B., Rego C., An Uncostrained quadratic binary programming approach to the vertex coloring problem // Annals of Operations Research. – 2005. - № 139.
21. David S. Johnson. Approximation algorithm for combinatorial problem // Journal of Computer and System Sciences. – 1974. - № 9. – P. 256 – 278.
22. David S. Johnson. Worst case behavior of graph coloring algorithms // In Proceedings of 5th Southeastern Conference on Combinatorics, Graph Theory and Computing, Utilitas Mathematica, Winnipeg, Canada. – 1974. – P. 513 – 527.
23. Pittel B. On the probable behavior of some algorithm for finding the stability number of graph // Mathematical Proceedings of Cambridge Philosophical Society. – 1982. - № 92. – P. 511 – 526.
24. Boros E., Hammer P. Pseudo-Boolean optimization // Discrete Applied Mathematics. – 2002. – № 123 (1-3).
25. Mertz P., Freisleben B. Genetic algorithms for binary quadratic programming // In Proceedings of the 1999 International Genetic and Evolutionary Computational Conference (GECCO'99), Morgan Kaufmann. – 1999.

Одержано 16.04.2007 р.