УДК 519.161

К.т.н., доцент Зорін Ю.М., студент Санжаревський Я.Ю.

**Національний технічний університету України
«Київський політехнічний інститут»**

ГЕНЕТИЧНИЙ АЛГОРИТМ РОЗФАРБУВАННЯ ГРАФА В *k* КОЛЬОРІВ

Вступ

Задача розфарбування графа в *k* кольорів є комбінаторною задачею, яку за часом виконання відносять до класу NP-складних задач.

Нехай є граф

$$G = (V, B) \qquad (1)$$

де V – вершини, B – ребра, причому

$$V = \{v_i \mid \forall i \in N \Rightarrow v_i \in [N \cup \{0\}]\},$$

$$B = \{< v_i, v_j > | [(i, j \in N) \wedge (i \neq j)] \Rightarrow v_i, v_j \in V\} \qquad (2)$$

З метою розфарбування графа доповнимо його новою характеристикою

$$G' : G' = (G, C) \equiv (V, B, C) \qquad (3)$$

де $C -$ *множина кольорів*. Задамо функцію розфарбування вершин графа:

$$F_c : N \cup \{0\} \rightarrow N \cup \{0\}, F_c(v \in V) = c \in C \qquad (4)$$

Виходячи з того, що кількість кольорів *k*, введемо обмеження на функцію:

$$card(im(F_c)) = K \qquad (5)$$

Тобто будь-яке відображення з точки зору оператора, що задовольняє операторну умову (5) буде точним (тобто в *k* і лише в *k* кольорів) розв'язком поставленої задачі.

Постановка задачі

Метою роботи є розробка генетичного алгоритму (ГА) розфарбування графа. Зокрема, розроблено кроссовер, що дозволяє ефективне використання канонічної моделі ГА для розв'язку цієї задачі. В свою чергу, запропонований кроссовер потребує кодування геному, яке істотно зменшує час його виконання.

Кодування геному й обчислення пристосованості

Кодування геному включає подання гена й геному як такого. Таке подання дозволяє оптимізувати час виконання кроссовера, що обраний для розв'язку поставленої задачі. Метою кодування є подання геному як набору множин вершин, що пофарбовані в один і лише один колір кожна, причому таких, що серед них немає сусідніх вершин, що пофарбовані в один колір.

З формальної точки зору граф розуміємо як (1), де відповідно його компоненти мають структуру (2).

Тепер задамо додаткове подання графа (3) як графа G з розфарбованими вершинами. Це означає, що визначено функцію розфарбовування (4) на множині його вершин. Таке подання графа дає повну інформацію про граф з точки зору задачі, що розглядається. З іншого боку, щоб збільшити швидкість виконання кроссовера, треба подавати граф у дещо інший спосіб.

Скористуємося наступним поданням розфарбованого графа: $G' = \hat{\bigcup}_{c \in C} V_c = \hat{\bigcup}_{c \in C} \bigcup_{v_C \in V} \{v_c\}$, де перший оператор об'єднання "збирає" множини в набір, утворюючи множину всіх множин.

Таке подання не може задовольнити основну умову в тому сенсі, що в кожній з множин не може бути двох інцидентних вершин. Таким чином, необхідно виключити такі вершини з кожної множини. Власне, це буде подання неповного графа. Це буде його підмножина, на якій задача розфарбування вже вирішена. Будемо розуміти шукане подання як обмеження існуючого:

$$G' \supset \overline{G'} = \hat{\bigcup}_{c \in C} \overline{V_c} = \hat{\bigcup}_{c \in C} \bigcup_{v_C \in \overline{V}} \{v_c\}, \qquad (6)$$

де $\overline{V} = V \setminus \{v_i, v_j \in V \mid \forall i, j \in [N \cup \{0\}] \wedge < v_i, v_j > \in B \Rightarrow F_c(v_i) = F_c(v_j)\}$

Подання кодування геному після завершення формалізації матиме наступний вигляд. Кожен ген містить номер кольору, в який він розфабрований та позицію у векторі вершин, що пофарбовані у цей колір. Якщо колір гена є *n* і існує *m* інцидентних вершин, що пофарбовані у цей же колір, то позиція у векторі цього кольору для всіх цих вершин буде дорівнювати -1. Безпосередньо геном містить в собі подання графа (6) та вектор конфліктів для кожного гена (вершини графа). Тобто геном має наступну структуру:

class Genome {

<boolean array[nodes]> conflicts;/* Цей масив для кожної вершини визначає чи має вона інциденту пофарбовану у той же колір */ <gene array[nodes]> genes;/* Цей масив включає в себе усі гени*/ <<integer array[PowerOfColoredSet] > array[colors]> representation; /* Для кожного з кольорів визначається множина вершин без ребер */

Найбільш природнім критерієм пристосованості геному була б кількість конфліктів, тобто кількість суміжних вершин, пофарбованих одним кольором. Але зважаючи на те, що у канонічному ГА більш пристосований геном повинен мати більшу міру пристосованості, у якості фітнесс-функції будемо брати величину, зворотну до кількості конфліктів.

Генетичні оператори

У роботі запропоновано спеціалізований тип кроссовера – MPI-кроссовер (Maximal Power of Intersection) [1]. Завдяки запропонованому способу подання графа з'являється можливість зменшити час його виконання. Псевдокод виконання кроссовера має вигляд:

```
MPICrossover(in: Genome father1, father2; out: Genome child1, child2)
     MPIChild(father1, father2, child1);
     MPIChild(father2, father1, child2);

MPIChild(in: Genome father1, father2; out: Genome child){
/* Для кожної незалежної розфарбованої підмножини з father1 */
     foreach (StableColoredSet1 in father1->StableColoredSets){
          int indexMaxIntersection = 0;
/* Для початку будемо вважати,що максимальна потужність
   перетину з підмножиною father2,що розфарбована в колір 0 */
     int max = Power(StableColoredSet1 intersect
                        father2->StableColoredSets[0]);
          Union _union = StableColoredSet1 union
                                father2->StableColoredSets[0];
          Nodes NodeColor; /* Це буде набір кольорів
                              для кожної з вершин child */
 /* Для кожної незалежної розфарбованої підмножини з father2*/
          foreach (StableColoredSet2 in father2->StableColoredSets){/*
Перевірка чи не більше потужність перетину з поточною підмножиною
father2 чим вважаємий нами максимум */
         if (Power(StableColoredSet1 intersect StableColoredSet2)> max
            max = Power(StableColoredSet1 intersect
            StableColoredSet2);
            indexMaxIntersection = father2->StableColoredSets->
                                      index(StableColored2);
           _union = StableColoredSet1 union StableColoredSet2;
   /* Розфарбовуємо вершини породженого геному, такі що
   належать кращему об'єднанню множин у колір цього об'єднання */
```

```
            foreach (Set in _union){
                  color(NodeColors[index(Set)]) = color(_union);
/* Розфарбовуваємо вершини, що не були розфарбовані
          у кольори відповідних вершин з parent1 */
      foreach (Node in NodeColors){
            if (color(Node) is undefined)
               color(Node) = color(parent1->Nodes[index(Node)]);
```

Таким чином складність алгоритму становитиме *O (K * K * N)*, що є цілком прийнятним.

У запропонованій реалізації ГА застосовано традиційний оператор мутації [2,3]. З огляду способу кодування геному, що використовується, слід зазначити, що мутація дещо ускладнюється, оскільки треба модифікувати подання після випадкової зміни кольору у випадково обраної вершини графу. Тобто змінюємо випадковим чином подання (3), після чого змінюємо згідно з ним подання (6).

Результати

Результати тестування ГА складних графів (500-1000 вершин) з набору тестів DIMACS [4] показали, що використання такого способу подання графа й спеціалізованого кроссовера істотно поліпшують результати розв'язування задачі за допомогою канонічної моделі ГА. При тестуванні використовувались три схеми селекції: стохастична селекція, селекція за допомогою колеса рулетки, турнірна селекція. Були також визначені оптимальні параметри ГА: розмір популяції – 130-160, вірогідність мутації – 12-14%, кроссовера – 90-92% .

Література

1. *Raphael D.*, *Jin-Kao H.* Parallel Problem Solving from Nature. – Springer Berlin / Heidelberg, 1998. – P. 845.
2. *Goldberg D.* Genetic Algorithms in Search, Optimization and Machine Learning – Addison Wesley, 1989. – P. 478.
3. *Mitchell M.* An Introduction to Genetic Algorithms. – MIT Press, 1999. – P. 158.
4. DIMACS Graphs: BenchmarkInstances : http://info.univ-angers/pub/ porumbel/graphs/index.html