Arab J Sci Eng
DOI 10.1007/s13369-017-2686-9

RESEARCH ARTICLE - COMPUTER ENGINEERING AND COMPUTER SCIENCE

Solution to Graph Coloring Using Genetic and Tabu Search Procedures

Raja Marappan[1] · **Gopalakrishnan Sethumadhavan**[1]

Received: 16 December 2016 / Accepted: 8 June 2017

Abstract Some of the engineering applications warrant the solution of Graph Coloring Problem, an NP-hard combinatorial optimization problem. This paper focuses on designing three new evolutionary operators using Tabu searching which are expected to offset the problems in the existing well-known methods in minimal search space and generations, besides maximizing the percentage of successful runs through effectively distributing promising genes for achieving fast stochastic convergence with smaller population size N. In the first method, Single Parent Conflict Gene Extended Crossover and Conflict Gene Mutation with Advanced Local Guided Search operators are designed and employed. These operators have been processed with Conflict Gene Removal constraints in the second method to minimize the search space and to increase the percentage of successful runs of the genetic algorithm. Multipoint Single Parent Conflict Gene Crossover and Multipoint Conflict Gene Mutation with Advanced Local Guided Search operators are employed along with a Conflict Gene Removal constraint in the third method. It has been exhibited that these operators reduce the search space by minimizing $\bar{g}$ to obtain a better near optimal solution. The solution for most of the small and large benchmark graphs has been obtained through multipoint crossover and mutation with reduced $\bar{g}$ whose value lies in a specific interval represented using maximum and minimum degrees of G. Our methods reduce the bound for the minimum colors obtained so far for certain families of graphs using smaller population size.

Keywords Approximation methods · Chromatic number · Combinatorial optimization · Evolutionary approach · Genetic algorithm · Graph coloring · NP-hard

1 Introduction

A simple graph G = (V, E) consists of set of vertices V = $\{v_1, v_2, v_3, \ldots, v_n\}$ and set of edges E = $\{e_1, e_2, e_3, \ldots, e_m\}$ such that each $e_k (1 \leq k \leq m)$ is uniquely associated with an unordered pair of vertices (v_i, v_j) $(1 \leq i, j \leq n)$ and $i \neq j$ [1]. Its adjacency matrix is denoted by A(G), an $n \times n$ symmetric binary matrix where A$(i, j) = 1 (1 \leq i, j \leq n)$ if $(v_i, v_j) \epsilon$ E(G); and A$(i, j) = 0 (1 \leq i, j \leq n)$ otherwise. $\chi(G)$, the chromatic number of G is defined as the *minimum number of colors* required to color V(G) so that no two adjacent vertices have the same color. Finding the value for $\chi(G)$ is called the Graph Coloring Problem (GCP), and it is used to solve the problems such as bandwidth assignment, register allocation, scheduling, noise reduction in circuits, optimum resource utilization and image segmentation [2–9]. The complexity of finding the value of $\chi(G)$ increases with n. Hence it is necessary to design fast approximation method that minimizes the search space and in turn maximizes the percentage of successful runs. There is no polynomial time algorithm to solve GCP because it is an NP-hard combinatorial optimization problem [1,10–14]. Hence it is needed to design a fast and effective approximation algorithm to find $\chi(G)$. Some of the approximation methods used to solve GCP are branch and bound [10], backtracking [15], branch and cut [16], asexual evolutionary algorithm [17],

Electronic supplementary material The online version of this article (doi:10.1007/s13369-017-2686-9) contains supplementary material, which is available to authorized users.

✉ Gopalakrishnan Sethumadhavan
sgk@mca.sastra.edu

Raja Marappan
raja_csmath@cse.sastra.edu

[1] Department of Computer Applications, SASTRA University, Tirumalaisamudram 613401, India

Published online: 22 June 2017

Ant Colony Optimization (ACO) [18], Particle Swarm Optimization (PSO) [19,20], Cuckoo search [21], heuristic and local search [22,23]. Genetic algorithms are adaptive heuristic search algorithms based on evolutionary strategies of natural selection and genetics. Compared to several existing approximation methods, genetic algorithms are found to be useful in solving the optimization problems that have vast search space. Designing new genetic operators with integer encoding to traverse the search space with minimal generations toward achieving a near optimal solution is found to be necessary. Some of the existing approaches have obtained near optimal solutions for some of the benchmark graphs. The existing genetic algorithms perform crossover operation on multiple parents, and some of the gene sequences are updated during each generation [3,24–26]. However, this paper focuses on performing enhanced genetic operations on single parent which are expected to achieve fast stochastic convergence in comparison with some of other existing methods. The new genetic operators are designed to increase the percentage of successful runs and to reduce $\bar{g}$ significantly for the graphs of different densities. The average crossovers $\bar{c}$ and the average mutations $\bar{m}$ should also be minimized to reduce the computational complexity.

Single Parent Conflict Gene Crossover (SPCGX) is a crossover operator that operates on a parent gene sequence selected by the fitness proportionate selection [27]. SPCGX is performed only when the chosen random probability $p_{\text{cr}} >$ the crossover probability p_c. SPCGX identifies all the conflicting edges to generate a new gene. A modified genetic algorithm of SPCGX by incorporating the Conflict Gene Removal (CGR) procedure has been designed [28]. CGR, a procedure that identifies and removes some of the conflicting genes arose during SPCGX and mutation. CGR is used to generate a better gene, which will have less conflict. To reduce the computational complexity, the expected crossovers $\bar{c}$, mutations $\bar{m}$ and generations $\bar{g}$ are also to be minimized. Hence this paper presents three different genetic methods by extending SPCGX with a new Advanced Local Guided Search (ALGS) operator further to reduce the search space by modifying the search into a reduced search space. ALGS operator improves the selected gene sequence before applying the crossover operation. It also fine-tunes the crossover offspring before applying the mutation.

Method 1 gets implemented with Single Parent Conflict Gene Extended Crossover (SPCGEX) and Conflict Gene Mutation (CGM) with ALGS operators. SPCGEX, a crossover operator operates on a single parent for finite iterations. It randomly chooses the conflicting edges to generate offspring. CGM, a mutation operator sets a conflict free gene and performs finite iterations to fix better conflict free gene. Extended SPCGEX (ESPCGEX) and Extended CGM (ECGM) operators with ALGS are devised in method 2. ESPCGEX extends SPCGEX with CGR for finite iterations and updates the gene sequence. ECGM extends CGM for finite iterations to fine-tune the offspring. Method 3 is implemented with Multipoint SPCGX (MSPCGX) and Multipoint CGM (MCGM) with ALGS operators. MSPCGX and MCGM perform multipoint crossovers and multipoint mutations at the conflicting vertices, respectively.

Section 2 explains the need for new genetic operators over the existing approaches. Section 3 explains the three new genetic and ALGS operators to solve GCP. Section 4 focuses on the analytics and stochastic convergence. Experimental results and the comparative analysis of these methods based on the graph density have been presented in Sect. 5. Conclusions are drawn in Sect. 6.

2 The Need for Proposed Genetic Methods over the Existing Approaches

Since there will be n choices of color for each of the n vertices, a solution space contains a total of n^n candidate solutions [1,29]. This number obviously grows very rapidly with regard to n, meaning that the number of candidate solutions to be checked will quickly become too large for even the most powerful computer to tackle. The computational complexity of finding $\chi(G)$ of a graph G is directly proportional to its instances n and m. Even if we were to limit ourselves to candidate solutions using a maximum of $\chi(G) < n$ colors labeled $1, 2, 3, \ldots, \chi(G)$, this would still lead to $(\chi(G))^n$ assignments needing to be considered, a function that is still subject to a similar combinatorial explosion for any $\chi(G) > 1$. If we are given a candidate solution using $\chi(G)$ different colors with labels $1, 2, 3, \ldots, \chi(G)$, the number of different assignments that would actually specify the same solution would be $(\chi(G))!$, representing all possible assignments of the $\chi(G)$ labels to the color classes. The space of all possible assignments of colors to vertices is therefore far larger (exponentially so) than it needs to be.

Thus, it is important to devise the fast evolutionary approach to improve the performance, minimize the problem search space and in turn maximize the percentage of successful runs. Some of the existing approaches such as branch and cut, ACO, column generation, semi-definite programming, PSO, DSATUR, FINOCCHI and FROGSIM algorithms have obtained near optimal solutions for some of the benchmark graphs [10,15,18–20,23,30,31]. Good upper bound selection is required in the branch and cut method to find near optimal solution. ACO is required the recursive algorithm which warrants additional memory resources for stack execution. The column generation approach is required for some of the computational complex operations to find near optimal solution. Tabu search is applied for moving step by step toward the minimum value of a function. To avoid cycling and being trapped in local minima, a Tabu list of forbidden movements is updated during the iterations [32]. New order-based crossovers have obtained near optimal solution for *flat300_28_0.col* and *flat1000_76_0.col* graphs [25]. Some of the existing genetic algorithms perform crossover operation on multiple parents, and few gene sequences are updated during each generation [3,24–26].

GCP can be generalized to selective graph coloring. Algorithm and applications of selective graph coloring are discussed in [33,34]. Graph coloring is also applied for signed graphs [35,36]. Some of the recent applications of graph coloring are automated timetable scheduling, optimum resource utilization, doctors scheduling problem, job scheduling [6–9]. A maximum number of dominating color classes when G is colored using χ(G) colors are discussed in [37]. The polynomial space algorithm to compute the number of independent sets of G with maximum degree 3 is discussed in [38]. Defective coloring, a variant of GCP in which adjacent vertices are allowed to have the same color, as long as the induced monochromatic components have a certain structure is discussed in [39]. The decentralized algorithm with heuristics for graph coloring is presented in [40]. The performance comparison of graph coloring algorithms is presented in [41].

New genetic operators are expected to increase the percentage of successful runs, and also to minimize the expected number of genetic generations $\bar{g}$ to obtain the improved near optimal solution. To reduce the search space and to reach the fast stochastic convergence, the population size N of the genetic algorithm must be set smaller at a value ($N \leq 15$). The ALGS and Tabu search operations make a fast search to find the better near optimal solution in the vast solution space. This is achieved by effectively distributing the promising genes during the generation of subsequent better gene sequences with a motive to reduce the values of the fitness function monotonically. The domain specific knowledge also helps to choose a right genetic operator and also to improve the output performance measurements. The graph instances n, m and graph density are considered to evaluate the performance measurements. The genetic operators are designed to increase the percentage of successful runs and to significantly reduce $\bar{g}$ for large, intermediate and small graphs of various densities. The expected crossovers $\bar{c}$ and the expected mutations $\bar{m}$ should be minimized to reduce the computational complexity.

The following are the new design strategies employed to resolve the offsets in existing methods:

1. Selection of two better gene sequences to perform the enhanced crossover and mutation in each generation with the motive of achieving fast stochastic convergence.
2. Design of multipoint crossovers and mutations with ALGS operator to reduce the search space to obtain the better near optimal solution.

The efficacies of these methods are also tested on some of the benchmark graphs. The efficiency of these methods can be ascertained on comparing the values of $\bar{g}$.

3 The New Genetic and Tabu Search Procedures

3.1 Notations

The proposed genetic methods use the following notations:
Notation 1: Minimum and Maximum Degree of G

The number of edges incident on the vertex v_i $(1 \leq i \leq n)$ is the degree, $d(v_i)$ in G.

Let δ(G) $=$ $\min_{1\leq i\leq n}\{d(v_i)|v_i \in$ V(G)} and Δ(G) $=$ $\max_{1\leq i\leq n}\{d(v_i)|v_i \in$ V(G)} be the minimum and maximum degrees of G, respectively.

Notation 2: Graph Density

The density of a graph G is $\frac{2m}{n(n-1)}$ where n is the number of vertices and m is the number of edges in G.

Notation 3: Gene Sequence

The color assignments of V(G) are represented as $(g_1, g_2, \ldots, g_n)$ where each g_i $(1 \leq i \leq n)$ is a positive integer.

Notation 4: Crossover and Mutation Probability, Population Size

The crossover and mutation probabilities are defined as p_c and p_m. The random collection of gene sequences at generation g is called the population, and it is denoted as P_g. The population size N is the number of distinct gene sequences at generation g. That is, $N = |P_g|$.

Notation 5: Objective Function of χ(G)-coloring

f(G), the general fitness function of G is the number of distinct integers present in the gene sequence and is also positive. It is determined using the survival of the fittest strategy in each generation. The objective function of χ(G)-coloring is to minimize f(G) such that f(G) - χ(G) = 0.

Notation 6: Conflicting Genes/Edges and Gene Sequence

The genes g_a $(1 \leq a \leq n)$ and g_b $(1 \leq b \leq n)$ corresponding to the vertices v_a and v_b are conflicting in a given gene sequence $(g_1, g_2, \ldots, g_n)$ iff $g_a = g_b$ where $(v_a, v_b) \in$ E(G). Then $(g_1, g_2, \ldots, g_n)$ is a conflicting gene sequence. The vertices v_a and v_b are also termed as conflicting vertices of that edge.

Notation 7: More Conflicting Vertex

For a vertex v_i $(1 \leq i \leq n)$, *count* (v_i) is the number of conflicting edges incident to v_i. For a conflicting edge (v_i, v_j) $(1 \leq i, j \leq n)$, if $count(v_i) > count(v_j)$ then v_i is assumed to be the more conflicting vertex.

3.2 Design of ALGS Operator

The ALGS operator improves the selected gene sequences i and j before applying the crossover operation. The crossover offspring i' and j' are applied through ALGS operator for fixed iterations t before applying mutation. This is achieved with a Tabu search that performs guided search with the help of short and eventually long-term memories. The search requires a neighborhood function defined over the search space. Starting with the selected gene sequence, say i, the search procedure proceeds iteratively to visit a series of locally best gene sequences following the gene sequence i. A best neighbor is chosen to replace the current gene sequence during the iterations. The local optima problem can be avoided by introducing Tabu lists. Each visited neighborhood is recorded, and the visited neighborhoods are forbidden to revisit during t iterations. The improved gene sequences, crossover offspring i' and j' are also recorded, and they are forbidden to revisit during the subsequent generations. The ALGS operator extends the LS operator [42].

The ALGS operator is designed as follows: The algorithm uses a heuristic of identifying the most conflicting vertices and replacing them with a valid-color to make the search more efficient. The neighbors of the selected gene sequence i are obtained by replacing the most conflicting vertices with a valid-color in the range [1, *minimum color*] where *minimum color* is the minimum number of colors. Hence the neighbor of a vertex v in the gene sequence i is characterized by a valid move defined by the couple $<v$, *valid-color*$[v]>$ belongs to V(G) x {1, 2, 3, …, *minimum color*}. When such a valid move $<v$, *valid-color*$[v]>$ is performed, the couple $<v$, *valid-color*$(v)>$ is classified Tabu for the next t iterations where t = (*Random* $(n) + \alpha\times$ number of conflicting vertices in i); $\alpha > 0$ and *Random* (n) returns a random integer in [0, n-1]. Hence during this period *valid-color*(v) cannot be reassigned to v. V(G) x {1, 2, 3, …, *minimum color*} table implements the Tabu lists. This operator memorizes and returns the most recent configuration of a gene sequence i^* among the best configurations found. After each iteration, the current configuration i replaces i^* if $F(i) \le F(i^*)$. The last best configuration is returned to obtain a solution which is as far away from the starting solution in order to better preserve the diversity in the population P.

```
Operator  : Advanced Local Guided Search (i)
Input     : Gene sequence i
Result    : The best of gene sequence $i^*$ found
Begin
        $i_0$ = i;
        While not stop-condition ( ) do
         Identify all the conflicting vertices in $i_0$;
         For each conflicting vertex v do
                        Find the least possible valid color c in the range of used colors in $i_0$ and set it to
                        color[v]; if there is a conflict then set color[v] = maximum color used in $i_0$ + 1;
           Add (v, valid-color[v]) in Tabu-list; // do not assign valid-color[v] to v for t iterations
        End while
        Update the best gene sequence $i^*$;
End
```

3.3 Proposed Genetic Methods

The flowchart for the proposed genetic and Tabu search methods is illustrated in Fig. 1. The proposed genetic and Tabu search procedures are presented below:

Step 1: Population Initialization

Initialize the number of genetic generation $g = 0$. The population P_0 can also be initialized by generating values drawn randomly from [1,*minimum color*].

Step 2: Fitness Evaluation and Selection

The following procedure is applied to select two better and one worst gene sequences in each generation g [28]:

a. Compute the fitness evaluation function $F_g(a)$ for each gene sequence a $(1 \le a \le N)$ in P_g as the number of conflicting edges present in gene sequence a at generation g.
b. Compute the probability for each gene sequence a $(1 \le a \le N)$ in P_g as

$$p(a) = F_g(a) / \sum_{a=1}^{N} F_g(a)$$

c. Compute the expected number of observations for each gene sequence a $(1 \le a \le N)$ in P_g as $E(a) = Np(a)$.
d. Choose two better gene sequences i and j such that $E(i)$ and $E(j)$ are minimum and next minimum among all the expected number of observations. Choose the worst gene sequence w where $E(w)$ is maximum among all expected number of observations.

Step 3: Crossover Operation

The selected gene sequences $i = (i_1, i_2, i_3, \ldots, i_n)$ and $j = (j_1, j_2, j_3, \ldots, j_n)$ are applied with ALGS operator to generate the offspring, i' and j', a set of gene sequences using the proposed crossover operators with a chosen crossover

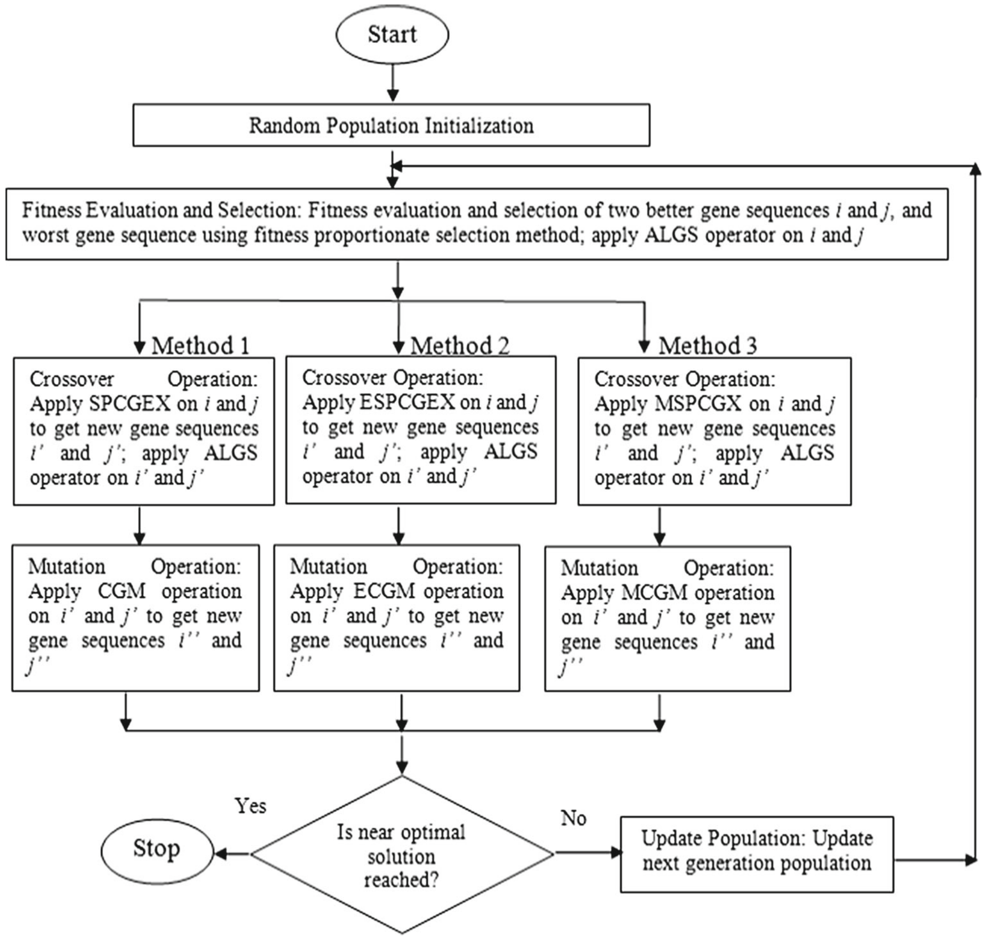
Start
Random Population Initialization
Fitness Evaluation and Selection: Fitness evaluation and selection of two better gene sequences i and j, and worst gene sequence using fitness proportionate selection method; apply ALGS operator on i and j
Method 1
Method 2
Method 3
Crossover Operation: Apply SPCGEX on i and j to get new gene sequences i' and j'; apply ALGS operator on i' and j'
Crossover Operation: Apply ESPCGEX on i and j to get new gene sequences i' and j'; apply ALGS operator on i' and j'
Crossover Operation: Apply MSPCGX on i and j to get new gene sequences i' and j'; apply ALGS operator on i' and j'
Mutation Operation: Apply CGM operation on i' and j' to get new gene sequences i'' and j''
Mutation Operation: Apply ECGM operation on i' and j' to get new gene sequences i'' and j''
Mutation Operation: Apply MCGM operation on i' and j' to get new gene sequences i'' and j''
Is near optimal solution reached?
Yes
Stop
No
Update Population: Update next generation population

Fig. 1 Flowchart of the proposed genetic algorithms to solve GCP

probability p_c and random probability p_{cr}. The crossover operations are performed when $p_{\mathrm{cr}} > p_c$. The various crossover operators are designed as follows: [Refer Supplementary Material No. 1]:
Method 1, SPCGEX:

a. Let (q, r) and (s, t) be two randomly chosen conflicting edges (i.e., $i_q = i_r$ and $j_s = j_t$), and let r and t be the most conflicting vertices in the gene sequences i and j respectively.
b. Generate offspring $(i_1, i_2, i_3, \ldots, i_r + 1, \ldots, i_n)$ and $(j_1, j_2, j_3, \ldots, j_t + 1, \ldots, j_n)$ such that $i_r + 1 \leq$ *minimum color* and $j_t + 1 \leq$ *minimum color*.
c. Repeat steps (a) and (b) for finite number of iterations and update the gene sequences i' and j' as $(i_{1'}, i_{2'}, i_{3'}, \ldots, i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'}, \ldots, j_{n'})$.

For example, in the graph shown in Fig. 2, the random color assignments of its vertices are indicated adjacent to them. For this instance, $i = (1, 1, 2, 1, 1, 2, 1, 1, 1, 1)$ is the selected gene sequence. For the conflicting edge (v_1, v_2), crossover is performed at its most conflicting vertex v_2. Hence the new gene at vertex v_2 becomes 2.
Method 2, ESPCGEX:

a. Apply the steps (a) and (b) of SPCGEX.
b. CGR: Check whether the new genes at r and t (i.e., i_r and j_t) corresponding to the edges (q, r) and (s, t) are conflicting. If so, apply SPCGEX at (q, r) and (s, t).
c. Repeat steps (a) and (b) for finite number of iterations and update the gene sequences i' and j' as $(i_{1'}, i_{2'}, i_{3'}, \ldots, i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'}, \ldots, j_{n'})$.

For example, in the graph shown in Fig. 3, the random color assignments of its vertices are indicated adjacent to them. For this instance, $i = (1, 1, 2, 1, 2, 2, 2, 1, 1, 1)$ is the selected gene sequence. For the conflicting edge (v_2, v_1), crossover is performed at its conflicting vertex v_1. Hence the new gene at vertex v_1 becomes 2 and subsequently for the new conflicting edge (v_1, v_3), CGR sets new gene at vertex v_3 as 3.
Method 3, MSPCGX:

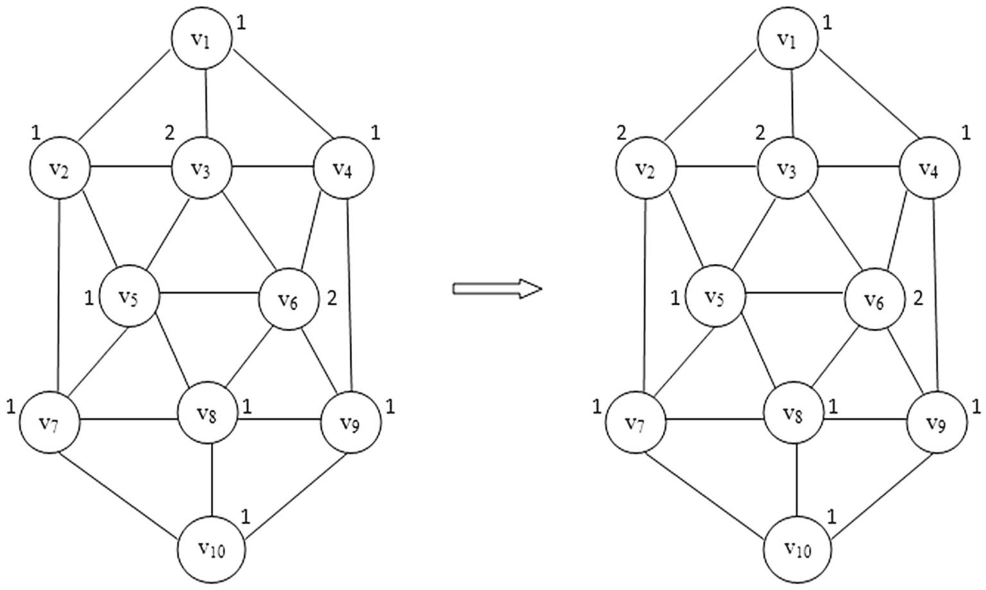

Fig. 2 SPCGEX operation

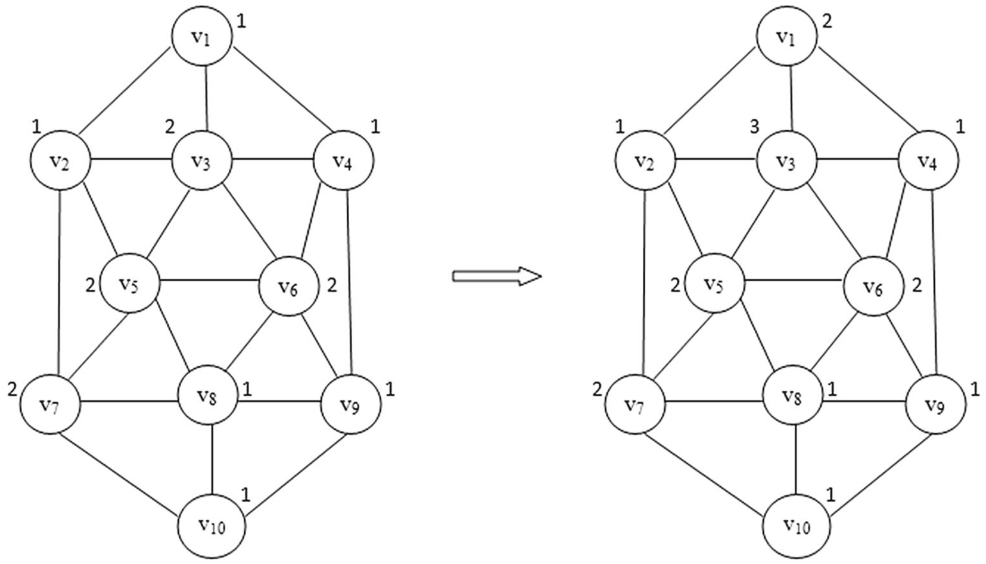

Fig. 3 ESPCGEX operation

a. Count the number of conflicting edges incident to each vertex in the gene sequences i and j.
b. Identify all the conflicting vertices in the gene sequences i and j.
c. Choose any two conflicting vertices r and t in gene sequences i and j.
d. Generate new gene sequences $(i_1, i_2, i_3, \ldots, i_r{+}1, \ldots, i_n)$ and $(j_1, j_2, j_3, \ldots, j_t{+}1, \ldots, j_n)$ such that $i_r + 1 \leq$ *minimum color* and $j_t + 1 \leq$ *minimum color*.
e. Check if the new genes at r and t (i.e., i_r and j_t) corresponding to the edges (q, r) and (s, t) are conflicting. If so, generate $i_r = i_r + 1$ and $j_t = j_t + 1$ such that $i_r + 1 \leq$ *minimum color* and $j_t + 1 \leq$ *minimum color*.
d. Repeat steps (d) and (e) for the remaining conflicting vertices and update the gene sequences i' and j' as $(i_{1'}, i_{2'}, i_{3'}, \ldots, i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'}, \ldots, j_{n'})$.

For example, in the graph shown in Fig. 4, the random color assignments of its vertices are indicated adjacent to them. For this instance, $i = (2, 2, 1, 2, 2, 1, 1, 2, 2, 1)$ is the selected gene sequence. For the conflicting vertices v_1, v_2 and v_3, the crossover generates new gene sequence as $i = (2, 3, 1, 2, 2, 3, 1, 2, 2, 1)$.

Step 4: Mutation Operation

The updated gene sequences i' and j' are applied with ALGS operator and are mutated with a chosen mutation

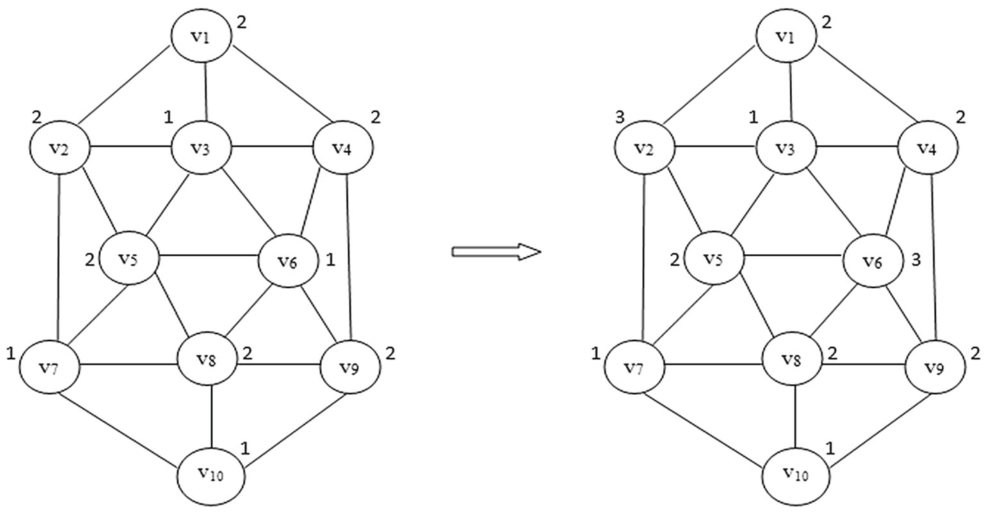

Fig. 4 MSPCGX operation

probability p_m to generate the offspring, i'' and j''. The proposed mutation operators are used to fine-tune the SPCGEX, ESPCGEX and MSPCGX generated offspring. The mutation operations are performed when the chosen random probability $p_{mr} > p_m$ and are defined as follows: [Refer Supplementary Material No. 1]:
Method 1, CGM:

a. Let (f', g') and (k', l') be two randomly chosen conflicting edges (i.e., $i'_{f'} = i'_{g'}$ and $j'_{k'} = j'_{l'}$)in the gene sequences i' and j'.
b. Generate new gene sequences $(i_{1'}, i_{2'}, i_{3'}, \ldots, i_{f'} - 1, \ldots, i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'}, \ldots, j_{k'} - 1, \ldots, j_{n'})$ such that $i_{f'} - 1 \geq 1$ and $j_{k'} - 1 \geq 1$.
c. Repeat steps (a) and (b) for finite number of iterations and update the gene sequences i'' and j'' as $(i_{1''}, i_{2''}, i_{3''}, \ldots, i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''}, \ldots, j_{n''})$.

For example, in the graph shown in Fig. 5, the random color assignments of its vertices are indicated adjacent to them. For this instance, $i' = (3, 1, 2, 1, 1, 2, 2, 1, 1, 1)$ is the selected gene sequence. For the conflicting vertex v_3 mutation updates its gene as 1.
Method 2, ECGM:

a. Apply steps (a) and (b) of CGM.
b. Check whether new genes at f' and k' (i.e., $i'_{f'}$ and $j'_{k'}$) corresponding to the edges (f', g') and (k', l') are conflicting. If so, apply CGM at f' and k'.
c. Repeat steps (a) and (b) for finite number of iterations and update the gene sequences i'' and j'' as $(i_{1''}, i_{2''}, i_{3''}, \ldots, i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''}, \ldots, j_{n''})$.

For example, in the graph shown in Fig. 6, the random color assignments of its vertices are indicated adjacent to them. For this instance, $i' = (3, 1, 2, 1, 3, 3, 2, 1, 1, 1)$ is the selected gene sequence. For the conflicting edge (v_5, v_6) mutation updates the gene at vertex v_5 to 1.
Method 3, MCGM:

a. Count the number of conflicting edges incident to each vertex in the gene sequence i' and j'.
b. Identify all the conflicting vertices in gene sequence i' and j'.
c. Choose any two conflicting vertices f' and k' in gene sequences i' and j'.
d. Generate new gene sequences $(i_{1'}, i_{2'}, i_{3'}, \ldots, i_{f'} - 1, \ldots, i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'}, \ldots, j_{k'} - 1, \ldots, j_{n'})$ such that $i_{f'} - 1 \geq 1$ and $j_{k'} - 1 \geq 1$.
e. Apply step (b) of ECGM.
f. Repeat steps (d) and (e) for the remaining conflicting genes and update the gene sequences i'' and j'' as $(i_{1''}, i_{2''}, i_{3''}, \ldots, i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''}, \ldots, j_{n''})$.

For example, in the graph shown in Fig. 7, the random color assignments of its vertices are indicated adjacent to them. For this instance, $i' = (3, 2, 3, 2, 3, 1, 1, 2, 2, 1)$ is the selected gene sequence. For the conflicting vertices v_1, v_3 and v_5 the mutation updates new gene as 1 at both vertices v_1 and v_5.
Step 5: Termination
Find $F_g(i'')$ and $F_g(j'')$ for the updated gene sequences i'' and j''. If either $F_g(i'')$ or $F_g(j'')$ is 0 then the solution is obtained and stop further generations. If not, $g = g + 1$ and update the next generation population P_g as follows:

a. If $F_g(i'') < F_{g-1}(w)$ then replace the worst gene w with i''.

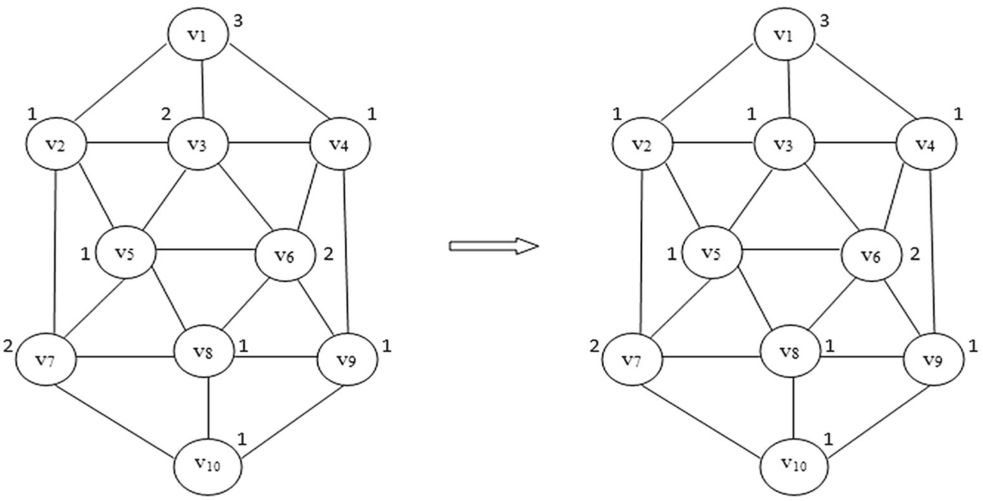

Fig. 5 CGM operation

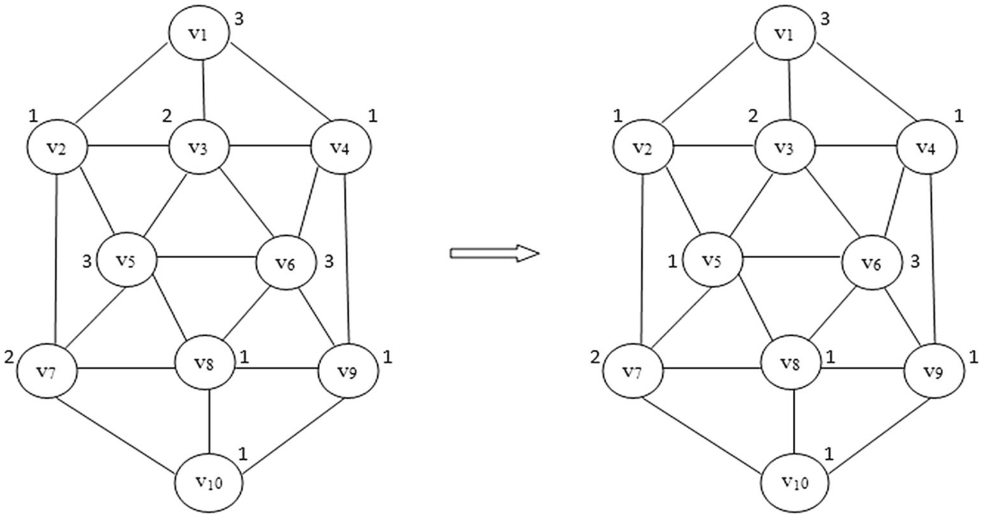

Fig. 6 ECGM operation

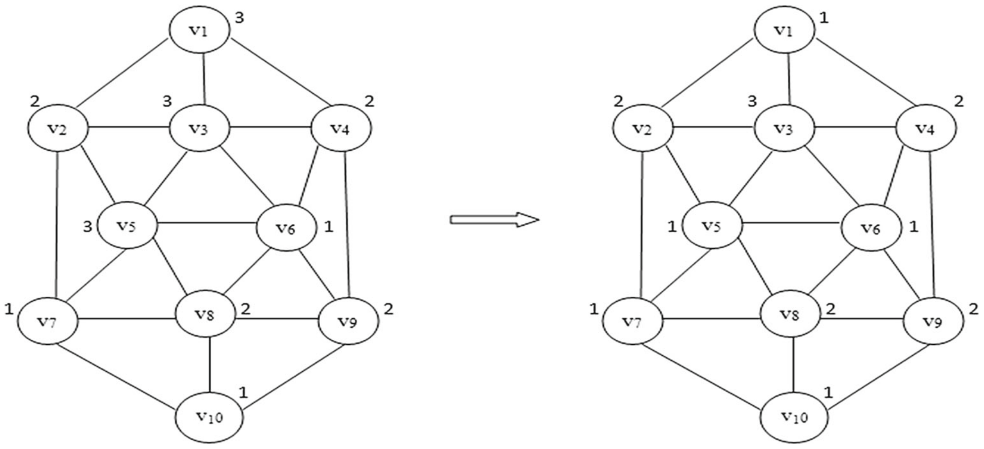

Fig. 7 MCGM operation

b. If $F_g(j'') < F_{g-1}(w)$ then replace the worst gene w with j'' and go to *Step 2*.

4 Analytics and Stochastic Convergence

Based on the values of n and m, the benchmark graphs are categorized into small, intermediate and large; and colorable and non-colorable cases are analyzed for these categories to achieve the fast stochastic convergence with smaller N. [Refer Supplementary Material No. 2, 4-6]:

4.1 Colorable Case

The crossover and mutation operations are performed based on the values of p_{cr} and p_{mr}. Let $i = (i_1, i_2, i_3, \ldots, i_n)$ be the gene sequence applied for crossover and mutation. The crossover operators always generate a new gene i_r such that $i_r + 1 \leq$ *minimum color* and the mutation operators set the new gene at $i'_{f'}$ in [1, f(G)]. The various devised genetic operators reduce the fitness function values $F_g(i)$ and $F_g(j)$ monotonically in the subsequent generations. However, the following cases are noticed during the execution of the devised methods:

4.1.1 Case 1

The values of $F_g(i)$ and $F_g(j)$ are monotonically decreasing in the subsequent generations and converging to best possible solution say at q^{th} generation such that $q \leq \Delta(G)/2^x$, $0 \leq x \leq \delta(G)$. That is, either $F_0(i) \geq F_1(i) \geq F_2(i) \geq \cdots \geq (F_q(i) = 0)$ or $F_0(j) \geq F_1(j) \geq F_2(j) \geq \cdots \geq (F_q(j) = 0)$. For the *myciel5.col* benchmark graph with 47 vertices with $F_0(i) = 10$ and $F_0(j) = 12$; $\Delta(G) = 23$; the values of the fitness function of i and j in the subsequent generations are obtained using MSPCGX & MCGM and are found to be 2, 1, 1, 1, 1, 1, 0 and 10, 2, 1, 1, 1, 1, 1, respectively.

4.1.2 Case 2

Initially $F_g(i)$ and $F_g(j)$ monotonically decrease for finite number of generations, say t. i.e., $F_0(i) \geq F_1(i) \geq F_2(i) \geq \cdots \geq F_t(i)$ and $F_0(j) \geq F_1(j) \geq F_2(j) \geq \cdots \geq F_t(j)$. Then based on the values of p_{cr} and p_{mr}, the fitness function values again increase for finite number of generations, say s. i.e., $F_t(i) \leq F_{t+1}(i) \leq \ldots \leq F_s(i)$ and $F_t(j) \leq F_{t+1}(j) \leq \ldots \leq F_s(j)$ such that $s \leq \Delta(G)/2^x$, $0 \leq x \leq \delta(G)$. Then solution has been reached after a finite number of generations. For the *jean.col* benchmark graph with 80 vertices $F_0(i) = 33$ and $F_0(j) = 34$; $\Delta(G) = 36$; the values of the fitness function of i and j in the subsequent generations are obtained using MSPCGX & MCGM and are found to be 33, 32, 32, 32, 34, 35, 36, 40, 2, 0 and 33, 32, 32, 33, 34, 39, 36, 40, 2, 2, respectively.

These cases are substantiating the fast stochastic convergence because the most promising genes are effectively distributed during the generation of subsequent better gene sequences with a motive to reduce the values of the fitness function monotonically.

4.2 Non-colorable Case

In the non-colorable case, after finite number of generations, say s, $F_g(i) \geq c$ and $F_g(j) \geq c$, for all $g = s, s{+}1, s+2$, etc., where c is a positive constant. For *queen6_6.col* ($\chi(G) = 7$) benchmark graph, a 6-coloring attempt with $F_0(i) = 13$ and $F_0(j) = 15$ brings $c = 16$; the values of the fitness function of i and j in the subsequent generations are obtained using MSPCGX & MCGM and are found to be: 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 17, 19, 27 and 16, 16, 16, 16, 16, 16, 16, 17, 17, 17, 17, 17, 19, 27, 37, respectively.

4.3 Stochastic Convergence

The genetic operators modify the population through a number of successive probabilistic transformations. Subsequent population is generated in a probabilistic way using the previous generation. This Markovian process is an appropriate model to analyze the probabilistic behavior of approximation algorithm [26]. The genetic algorithm is said to be stochastically convergent if the following conditions are satisfied [43]:

1. Reachability condition: For any two gene sequences i, $j \in$ finite search space S, then j is reachable from i. i.e., $j' = (j_{1'}, j_{2'}, j_{3'}, \ldots, j_{n'})$ and $j'' = (j_{1''}, j_{2''}, j_{3''}, \ldots, j_{n''})$ are the two gene sequences generated from the gene sequences $j = (j_1, j_2, j_3, \ldots, j_n)$ and $j' = (j_{1'}, j_{2'}, j_{3'}, \ldots, j_{n'})$, respectively.

   If $(0 < p\{j'' = (\mathit{crossover}(j) \text{ and } \mathit{mutation}(j'))\} < 1)$ then gene sequence j'' is said to be reachable from gene sequence j'.
2. Monotone: The population P_g is monotone.

Theorem *The proposed genetic algorithms converge stochastically.*

Proof To prove the reachability condition of gene sequences: SPCGEX generates a new gene at a conflicting vertex r as $j_r = j_r + 1$ if $j_r + 1 \leq$ *minimum color*. Now the new gene at r assumes either j_r or $j_r + 1$. ESPCGEX and MSPCGX further check the conflict at r. On the existence of conflict at r it generates j_r as $j_r = j_r + 1$ such that $j_r + 1 \leq$ *minimum color*. Hence the new gene takes any one of the three possible integers: j_r, $j_r + 1$, $j_r + 2$.

$$p\{j' = crossover(j)\} = c_1/3^k > 0 (1 \le j \le N, 1 \le j' \le N) \text{ where } c_1 \text{ and } k \text{ are positive integers.} \quad (1)$$

□

The CGM, ECGM and MCGM operations always update gene at a conflicting vertex in [1, f(G)].

$$p\{j'' = mutation(j')\} = c_2/f(G)^l > 0 \text{ where } c_2 \text{ and } l \text{ are positive integers.} \quad (2)$$

For the chosen p_c and p_m, now $p(j'')$ satisfies

$$p\{j'' = (crossover(j) \text{ and } mutation(j'))\} \ge p_c.p\{j' = crossover(j)\}.p_m.p\{j'' = mutation(j')\}$$

Using (1) and (2): $\mathrm{p}(\mathrm{j}'' = (crossover(\mathrm{j}) \text{ and } mutation(\mathrm{j}'))\} \ge \mathrm{p_c c_1/3^k p_m c_2/f(G)^l} > 0$

Hence j'' is reachable from j'.

To prove monotone condition: Fitness Evaluation and Selection selects two better gene sequences in each generation. The probability of selecting better gene sequences using this selection is always greater than 0. Moreover, the worst gene sequence is replaced with a better gene sequence for updating the population. Hence P_g is monotone for the positive integers of g, and the proposed genetic algorithms converge stochastically.

5 Experiments on Some Benchmark Graphs and Their Comparisons

The experiments are conducted on some of the benchmark graphs using Intel Xeon processor with 256 GB DDR3 in Windows 8 Professional OS under JDK 1.8 environment. Experiments are conducted on the different categories of benchmark graphs for 1000 runs each with 5000000 generations to obtain *minimum color*. The SPCGEX & CGM, ESPCGEX & ECGM and MSPCGX & MCGM operators have been tested on some of the benchmark graphs with different crossover and mutation probability. Crossover at each gene within a single gene sequence i depends on the dissimilarity of genes of the adjacent vertices and the reduced value of $\mathrm{F}_g(i)$. Thus, too low p_c reduces the chance of minimizing $\mathrm{F}_g(i)$ to the offspring in one run. Whereas, a high value of p_c can choose each individual point within the gene sequence to perform crossover operation. Thus, p_c and p_m have been set as 0.8 and 0.3 respectively.

The graph instances n, m and graph density are considered and categorized to assess the performance measurements of the proposed genetic procedures. Results are tabulated in Tables 1, 2, 3 and 4. Outcome of these experiments reveals the following: [Refer Supplementary Material No. 2, 4-6, 8]:

1. The percentage of successful runs gradually increases in the methods 1, 2 and 3 for *queen5_5.col* graph whose density is 1.1. Moreover, $\bar{g}$ is also significantly reduced as shown in Fig. 8. MSPCGX & MCGM operators have achieved the stochastic convergence to reach the optimal solution even for the high-density graph *miles1500.col*.
2. The density of *queen6_6.col*, *queen7_7.col* and *queen8_8.col* is 0.9, 0.8 and 0.7, respectively. It has been found that for these graphs $\chi(G)$ increases with decreasing densities. Moreover, methods 2 and 3 reduce $\bar{g}$ compared to method 1 (Fig. 8). For *miles1000.col* graph whose density is 0.8, method 3 stochastically converges to the optimal solution. The stochastic convergence of the genetic operators to reach an optimal solution is proportional to the number of edges m and $\chi(G)$.
3. Graphs whose density lie in [0.1, 0.6]: No significant differences are observed in the high performance of successful runs and $\bar{g}$ for small *Mycielski* graphs. However for the intermediate graphs, these methods yield low percentage of successful runs. It has also been observed for small graphs such as *huck.col*, *david.col* whose density is 0.1, method 2 outperforms methods 1 and 3 in average number of generations (Figs. 8, 9).
4. Graphs whose density lies in [0.01, 0.09]: $\chi(G)$ increases when density decreases except for *miles250.col* and *homer.col* graphs. No significant differences are observed in methods 1 and 2. Method 3 increases the percentage of successful runs and reduces $\bar{g}$ for graphs with $\chi(G) \ge 10$ such as *jean.col*, *inithx.i.1.col*, *inithx.i.2.col*, *inithx.i.3.col*. Low percentage of successful runs is obtained for *miles250.col*. For the low-density (0.01) graph *homer.col*, method 3 seems to be suitable in terms of successful runs. However in terms of $\bar{g}$, method 1 is feasible (Figs. 8, 9).
5. $\bar{g}$ increases with n and m for most of the planar and book graphs.
6. Method 2 minimizes $\bar{g}$ and increases the percentage of successful runs for most of the graphs compared to method 1.
7. In method 3, $\bar{g} \in [2, \Delta(G)/2^x]$ such that $0 \le x \le \delta(G)$, for small and large graphs except planar, book graphs such as *david.col*, *homer.col* and large triangle free graph *myciel7.col*. For example, in the case of *queen6_6.col* graph ($n = 36$, $\delta(G) = 15$, $\Delta(G) = 19$), $\bar{g}$ lies in [2, $\Delta(G)/2^x$] with $x = 1$. The value of x reaches 0 for smaller n, $n \le 25$. For $n > 25$, x increases with n and $\bar{g}$. For large size graphs such as *inithx.i.2.col* and *inithx.i.3.col* it has been found that x also increases.
8. The genetic operators have achieved the stochastic convergence for the graphs having large edges such as *queen*

Table 1 Computation of near optimal solution on small graphs ($n < 100$)

Graph No	Graph (G)				Minimum color obtained in			
	Graph type	Instances	χ(G)	Density	Existing method	Method 1	Method 2	Method 3
1	queen5_5.col	$n = 25; m = 320$	5	1.07	5	5	5	5
2	queen6_6.col	$n = 36; m = 580$	7	0.92	7	7	7	7
3	queen7_7.col	$n = 49; m = 952$	7	0.81	7	7	7	7
4	queen8_8.col	$n = 64; m = 1456$	9	0.72	9	9	9	9
5	myciel5.col	$n = 47; m = 236$	6	0.22	6	6	6	6
6	myciel6.col	$n = 95; m = 755$	7	0.17	7	7	7	7
7	myciel4.col	$n = 23; m = 71$	5	0.28	5	5	5	5
8	myciel3.col	$n = 11; m = 20$	4	0.36	4	4	4	4
9	huck.col	$n = 74; m = 301$	11	0.11	11	11	11	11
10	jean.col	$n = 80; m = 254$	10	0.08	10	10	10	10
11	david.col	$n = 87; m = 406$	11	0.11	11	11	11	11
12	queen8_12.col	$n = 96; m = 2736$	12	0.60	12	15	12	12
13	queen9_9.col	$n = 81; m = 2112$	10	0.65	10	11	10	10
14	1-Insertions_4.col	$n = 67; m = 232$	4	0.10	4	4	4	4
15	2-Insertions_3.col	$n = 37; m = 72$	4	0.11	4	4	4	4
16	3-Insertions_3.col	$n = 56; m = 110$	4	0.07	4	4	4	4
17	4-Insertions_3.col	$n = 79; m = 156$	3	0.05	3	3	3	3

Table 2 Computation of near optimal solution on intermediate graphs ($100 \leq n < 500$)

Graph No	Graph (G)				Minimum color obtained in			
	Graph type	Instances	χ(G)	Density	Existing method	Method 1	Method 2	Method 3
18	myciel7.col	$n = 191; m = 2360$	8	0.13	8	8	8	8
19	games120.col	$n = 120; m = 638$	9	0.09	9	9	9	9
20	miles250.col	$n = 128; m = 387$	8	0.05	8	8	8	8
21	anna.col	$n = 138; m = 493$	11	0.05	11	11	11	11
22	queen10_10.col	$n = 100; m = 2940$	11	0.59	11	14	11	11
23	queen12_12.col	$n = 144; m = 5192$	12	0.50	12	18	12	12
24	queen14_14.col	$n = 196; m = 8372$	14	0.44	14	20	15	14
25	queen15_15.col	$n = 225; m = 10360$	15	0.41	15	20	15	15
26	queen16_16.col	$n = 256; m = 12640$	16	0.39	16	22	17	16
27	queen11_11.col	$n = 121; m = 3960$	11	0.55	11	18	11	11
28	queen13_13.col	$n = 169; m = 6656$	13	0.47	13	19	14	13
29	miles500.col	$n = 128; m = 1170$	20	0.14	20	22	20	20
30	miles750.col	$n = 128; m = 4226$	31	0.52	31	32	31	31
31	miles1000.col	$n = 128; m = 6432$	42	0.79	42	48	43	42
32	miles1500.col	$n = 128; m = 10396$	73	1.28	73	78	75	73
33	zeroin.i.1.col	$n = 211; m = 4100$	49	0.19	49	51	49	49
34	zeroin.i.2.col	$n = 211; m = 3541$	30	0.16	30	30	30	30
35	zeroin.i.3.col	$n = 206; m = 3540$	30	0.17	30	30	30	30
36	mulsol.i.1.col	$n = 197; m = 3925$	49	0.20	49	50	49	49
37	mulsol.i.2.col	$n = 188; m = 3885$	31	0.22	31	32	31	31
38	mulsol.i.3.col	$n = 184; m = 3916$	31	0.23	31	33	31	31
39	mulsol.i.4.col	$n = 185; m = 3946$	31	0.23	31	32	31	31
40	mulsol.i.5.col	$n = 186; m = 3973$	31	0.23	31	32	31	31

Table 2 continued

Graph No	Graph (G)				Minimum color obtained in			
	Graph type	Instances	χ(G)	Density	Existing method	Method 1	Method 2	Method 3
41	le450_5a.col	$n = 450; m = 5714$	5	0.06	5	12	9	5
42	le450_5b.col	$n = 450; m = 5734$	5	0.06	5	12	9	5
43	le450_5c.col	$n = 450; m = 9803$	5	0.10	5	13	10	5
44	le450_5d.col	$n = 450; m = 9757$	5	0.10	5	13	11	5
45	le450_15b.col	$n = 450; m = 8169$	15	0.08	15	24	20	15
46	le450_15c.col	$n = 450; m = 16680$	15	0.16	15	25	22	15
47	le450_15d.col	$n = 450; m = 16750$	15	0.17	15	25	21	15
48	le450_25a.col	$n = 450; m = 8260$	25	0.08	25	38	30	25
49	le450_25b.col	$n = 450; m = 8263$	25	0.08	25	38	29	25
50	le450_25c.col	$n = 450; m = 17343$	25	0.17	25	27	27	25
51	le450_25d.col	$n = 450; m = 17425$	25	0.17	25	27	26	25
52	school1.col	$n = 385; m = 19095$	14	0.26	14	31	25	14
53	school1_nsh.col	$n = 352; m = 14612$	14	0.24	14	30	23	14
54	fpsol2.i.1.col	$n = 496; m = 11654$	65	0.09	65	70	68	65
55	fpsol2.i.2.col	$n = 451; m = 8691$	30	0.09	30	35	30	30
56	fpsol2.i.3.col	$n = 425; m = 8688$	30	0.10	30	34	31	30
57	2-Insertions_4.col	$n = 149; m = 541$	4	0.05	4	4	4	4
58	DSJC125.1.col	$n = 125; m = 736$	5	0.09	5	5	5	5
59	DSJC125.5.col	$n = 125; m = 3891$	–	0.50	17	17	17	17
60	R125.1.col	$n = 125; m = 209$	–	0.03	5	5	5	5
61	R125.1c.col	$n = 125; m = 7501$	–	0.97	46	46	46	46
62	R125.5.col	$n = 125; m = 3838$	–	0.50	36	36	36	36
63	R250.1.col	$n = 250; m = 867$	–	0.03	8	8	8	8
64	R250.1c.col	$n = 250; m = 30227$	–	0.97	64	64	64	64
65	flat300_28_0.col	$n = 300; m = 21695$	28	0.48	28	28	28	28

Table 3 Computation of near optimal solution on large graphs ($n \geq 500$)

Graph No	Graph (G)				Minimum color obtained in			
	Graph Type	Instances	χ(G)	Density	Existing method	Method 1	Method 2	Method 3
66	latin_square_10.col	$n = 900; m = 307350$	–	0.76	122	122	105	98
67	homer.col	$n = 561; m = 1629$	13	0.01	13	13	13	13
68	inithx.i.1.col	$n = 864; m = 18707$	54	0.05	54	54	54	54
69	inithx.i.2.col	$n = 645; m = 13979$	31	0.07	31	31	31	31
70	inithx.i.3.col	$n = 621; m = 13969$	31	0.07	31	31	31	31
71	DSJC500.5.col	$n = 500; m = 62624$	–	0.50	48	48	48	48
72	DSJR500.1.col	$n = 500; m = 3555$	–	0.03	12	12	12	12
73	DSJR500.5.col	$n = 500; m = 58862$	122	0.47	122	122	122	122
74	DSJC1000.5.col	$n = 1000; m = 249826$	–	0.50	83	83	83	83
75	R1000.1.col	$n = 1000; m = 14378$	–	0.03	20	20	20	20
76	flat1000_50_0.col	$n = 1000; m = 245000$	50	0.49	50	50	50	50
77	flat1000_60_0.col	$n = 1000; m = 245830$	60	0.49	60	60	60	60
78	flat1000_76_0.col	$n = 1000; m = 246708$	76	0.49	82	82	82	76

Table 4 Comparison of *Minimum Color* Obtained with Other Methods [19,20,30,31,40–42]

Graph	$\chi(G)$	Near optimal color obtained using							
		TS	HCA	RLF	HDPSO	MTPSO	Our methods		
							1	2	3
le450_25c.col	25	26	26	27	26	31	27	27	25
le450_25d.col	25	26	26	28	26	32	27	26	25
flat300_28.col	28	32	31	37	31	35	28	28	28
flat1000_76_0.col	76	87	83	106	85	87	82	82	76
latin_square_10.col	–	130	127	122	132	134	122	105	98

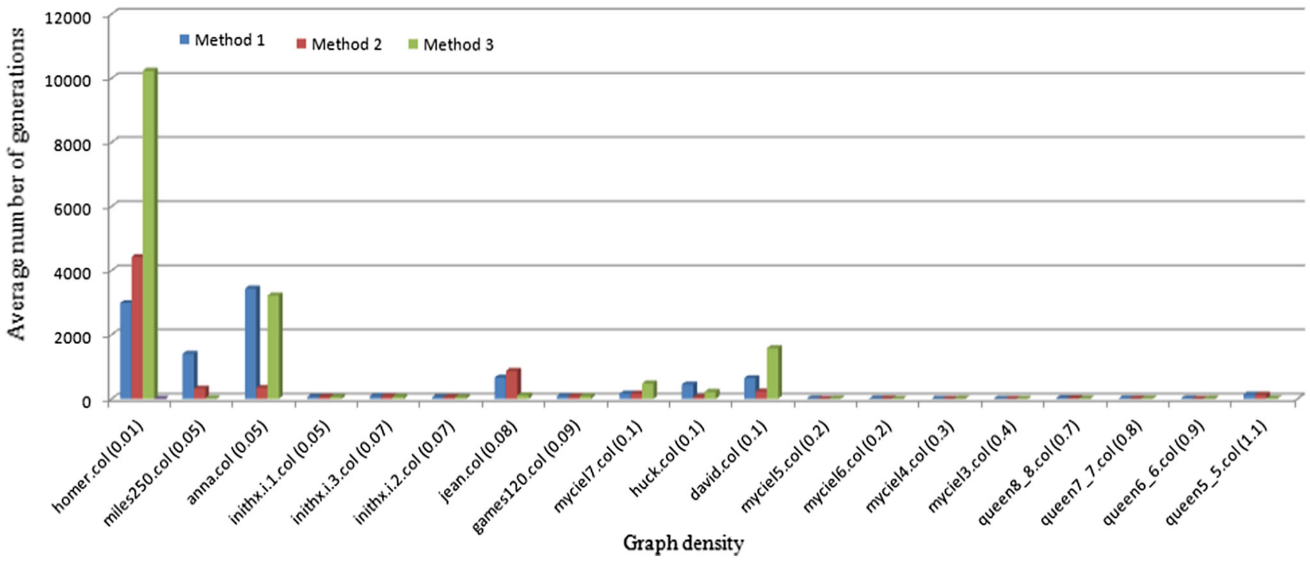

Fig. 8 Graph density versus average number of generations ($\bar{g}$) obtained

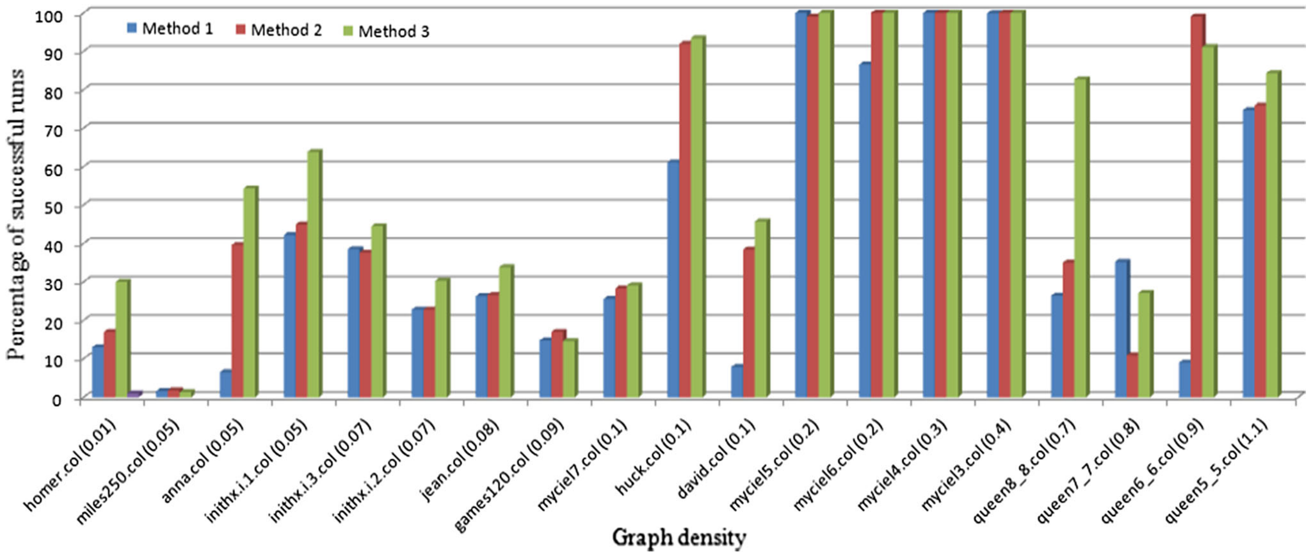

Fig. 9 Graph density versus percentage of successful runs obtained

and register allocation graphs in the reduced computational complexity. For the register allocation graphs, the solution has been reached in minimal values of $\bar{g}$ which are less than 100. Compared to methods 1 and 2, method 3 also stochastically converges in the minimal $\bar{g}$ even for Miles graphs which are similar to geometric graphs.

9. For most of the graphs $\bar{c} < \bar{m}$ for $\chi(G\pm i)$ colors for all $i = 0, 1, 2$.
10. Many distinct optimal gene sequences (x*) are also obtained.

The proposed evolutionary methods are compared with some of the existing well-known approaches to offset the problems such as improving the near optimal performance, to minimize the problem search space and in turn to maximize the percentage of successful runs, to reduce the search space and to reach the fast stochastic convergence with smaller N. The major inferences to target the justification of these new evolutionary operators are experimentally achieved and are listed below [Refer Supplementary Material No. 2, 4-8]:

1. The proposed methods stochastically converge even for the smaller population size N. It has been reported that the average number of generations $\bar{g}$ taken to obtain the solution for *myciel3.col* graph as 57.8 for $N = 20$ after performing preprocessing such as sorting on the independent sets, reordering largest sets first, relabeling the color [26]. However, the solution for the same graph has been achieved with minimal average number of generations ($\bar{g} < 6$) even for lesser population size ($N = 5$) in any of the methods presented in this paper without performing any preprocessing operations. But the devised methods do not need any preprocessing operations unlike in [25,26]. It has been found for all large size graphs solution has been achieved with very less population size ($N = 15$) unlike in [44]. The devised methods stochastically converge for *inithx.i.1.col* and *myciel6.col* graphs. However, it has been reported and got near equal convergence using PCPX and DBX operators [44].
2. MSPCGX & MCGM stochastically converge and run 82.7% successfully for *queen8_8.col* graph. It has been reported that the percentage of successful runs obtained for *queen7_7.col* and *queen8_8.col* graphs using random mutation is 1% and 2%, respectively, besides excessive memory utility during the execution of backtracking [45].
3. Only on the selection of good upper bound the branch and cut method yields solution [15]; otherwise, it provides lower bound on $\chi(G)$. ACO requires the execution of a recursive algorithm for initial coloring [18]. One of the semi-definite programming based on recursive heuristic presented in [23] obtains near optimal solution for *queen8_8.col* graph as *minimum color* = 10. But our genetic methods give the solution for this graph as $\chi(G) = 9$.
4. The column generation approach presented in [10] applies implicit optimization at each node of the branch and bound search tree. The method requires complex computational operations such as finding the independent sets using recursion, generating columns to improve the linear program at each node of the branch and bound tree, applying the branching methodology using depth-first search strategy and restricted programming technique, enforcing integrality if the solution to the linear relaxation contains fractional values. Moreover, it has been reported by the author that the solution for *Mycielski* graphs has not been obtained. But our operators obtain the solution for these graphs as well.
5. ESPCGEX & ECGM minimize the value of $\bar{g}$ for the planar graph with $n = 28$ as compared to the MTPSO and MPSO algorithms proposed in [19] and [20].
6. Near optimal solution for queen graphs such as *queen6_6.col*, *queen7_7.col* and *queen8_8.col* is obtained [30]. DSATUR algorithm obtained the near optimal solution for queen graphs *queen11_11.col*, *queen12_12.col*, *queen13_13.col*, *queen14_14.col*, *queen15_15.col* and *queen16_16.col*; but our method 3 obtains the solution for these graphs. Method 3 also minimizes near optimal performance of *le450_15b.col*, *le450_15c.col*, *le450_15d.col* compared to DSATUR algorithm.
7. The FINOCCHI & FROGSIM algorithms obtain the near optimal coloring for *queen13_13.col* (χ(G) = 13) graph as 18 and 17 respectively [30]. But our method 3 gives the optimal solution. FINOCCHI & FROGSIM obtained the near optimal color for *miles750.col* (χ(G) = 31) as 32, but our method 3 yields the optimal solution. The FROGSIM algorithms obtained the near optimal solution for *le450_5a.col*, *le450_5b.col*, *le450_5c.col*, *le450_5d.col*, *le450_15b.col*, *le450_15c.col*, *le450_15d.col*, *le450_25a.col*, *le450_25b.col*, *le450_25c.col*, *le450_25d.col* as 11, 11, 8, 10, 20, 28, 28, 27, 26, 34, 34, respectively [30]; but our method 3 obtains optimal solution to these graphs.
8. Our methods obtain the solution for some of the random graphs such as 1-*Insertions_4.col*, *2-Insertions_3.col*, *3-Insertions_3.col*, *4-Insertions_3.col*, *2-Insertions_4.col*, *DSJC125.1.col*, *flat300_28_0.col*, *DSJR500.5.col*, *flat1000_50_0.col*,*flat1000_60_0.col*,*flat1000_76_0.col*.
9. Our methods are compared with some of the existing well-known methods, and the results are tabulated in Table 4. Compared with some of the existing methods, method 3 stochastically converges to *flat1000_76_0.col* and also minimizes the minimum color of *latin_square_10.col* to 98. Our methods obtain better near optimal solution compared to FF, LDO, WP, IDO, DSATUR and RLF algorithms presented in [41].

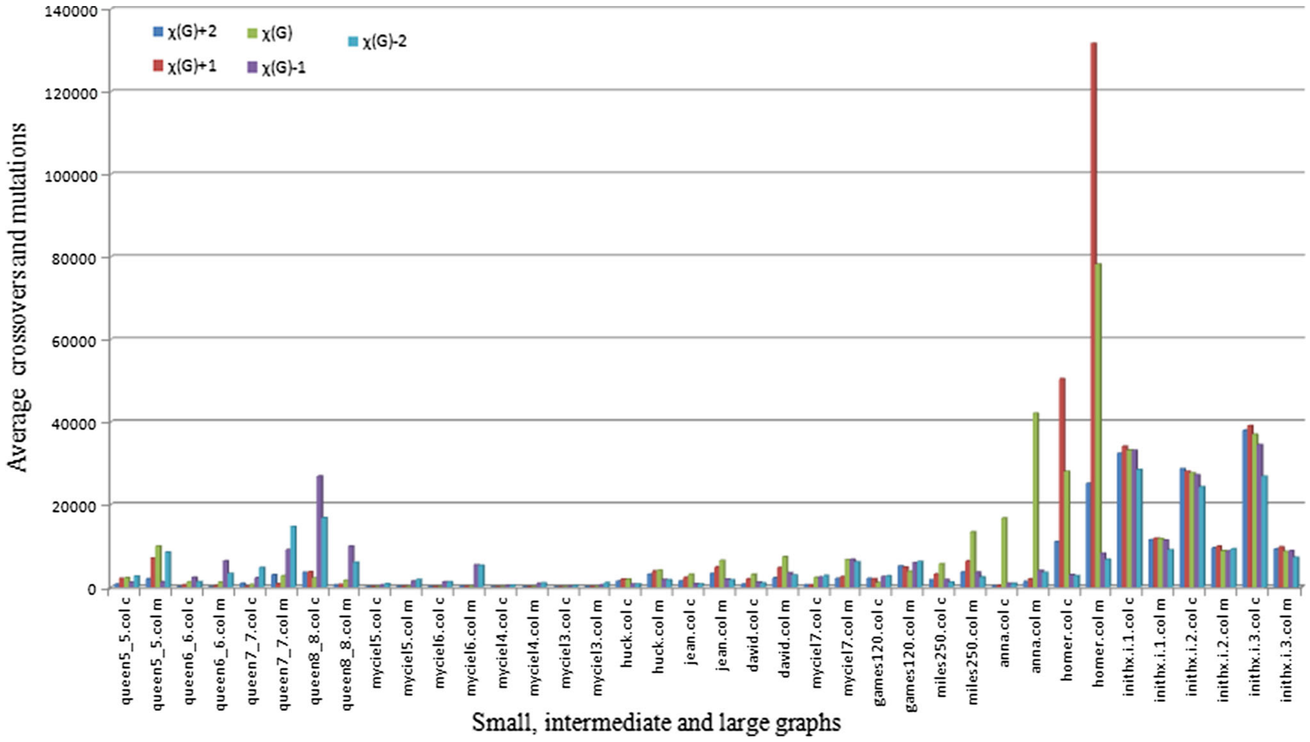

Fig. 10 $\bar{c}$ & $\bar{m}$ performed around χ(G), for various small, intermediate and large graphs using method 1

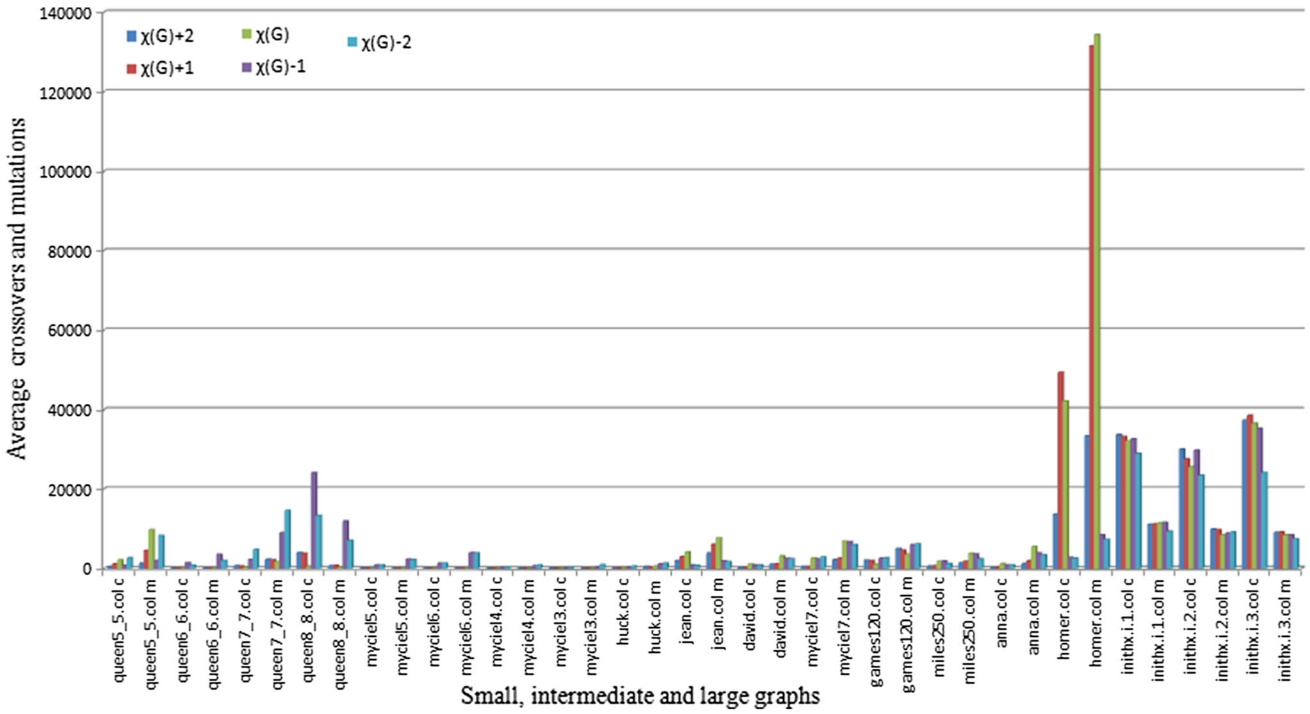

Fig. 11 $\bar{c}$ & $\bar{m}$ performed around χ(G), for various small, intermediate and large graphs using method 2

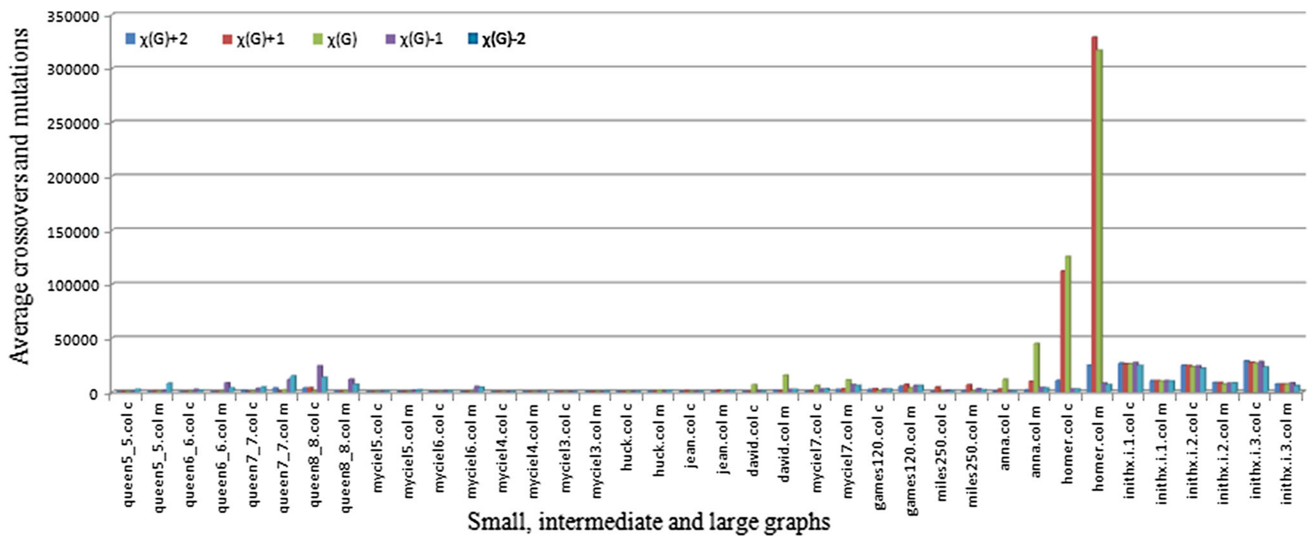

Fig. 12 $\bar{c}$ & $\bar{m}$ performed around χ(G), for various small, intermediate and large graphs using method 3

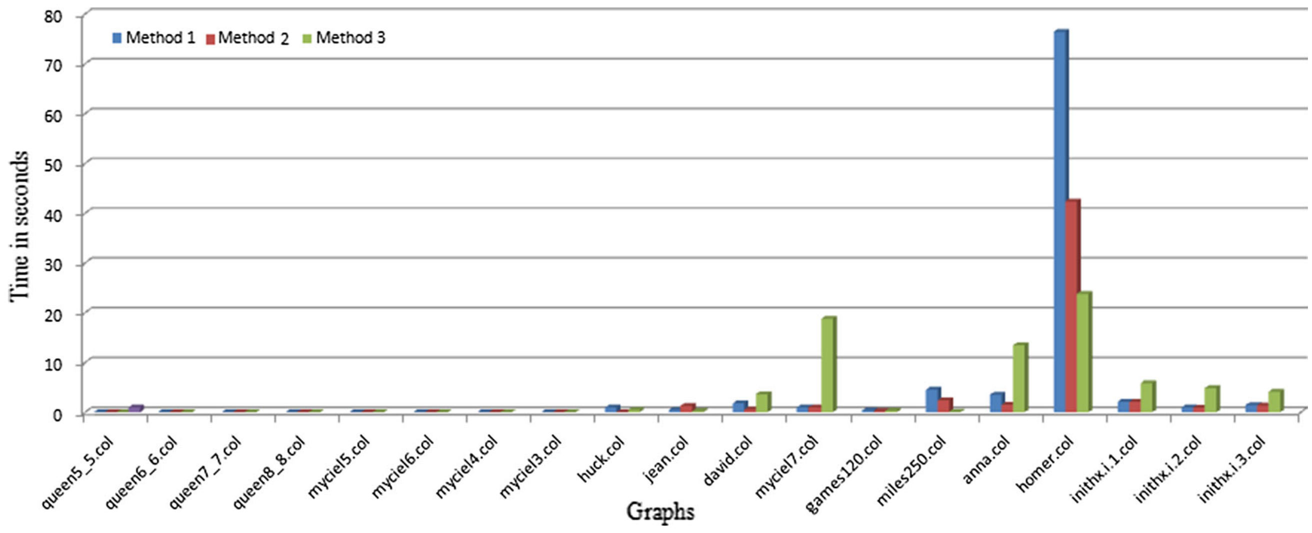

Fig. 13 The average computing time of χ(G), for various benchmark graphs using the devised methods

10. FROGSIM and SDGC algorithms [40] are obtained near optimal solution to *flat300_28_0*, *flat1000_50_0*, *flat1000_60_0*, *le450_25c.col*, *le450_25d.col* graphs. But our methods obtain the solution to *flat300_28_0*, *flat1000_50_0*, *flat1000_60_0* graphs. MSPCGX & MCGM operators obtain the solution to *le450_25c.col*, *le450_25d.col* graphs.

Methods 2 and 3 reduce the *minimum color* to χ(G) compared to method 1 for most of the graphs. $\bar{g}$ and the percentage of successful runs obtained for various benchmark graphs are shown in Figs. 8 and 9 (Refer Supplementary Material No. 8). The density of the benchmark graphs is indicated in parenthesis adjacent to the respective graphs. It has been found from Fig. 8 that higher $\bar{g}$ is obtained for low-density graphs and lower $\bar{g}$ is obtained when density increases. Figure 9 concludes that low percentage of successful runs is obtained for most of the low-density graphs; when density increases the percentage of successful runs is increased for most of the graphs.

The $\bar{c}$ & $\bar{m}$ performed around χ(G) for various categories of graphs using the devised methods are shown in figures from Figs. 10, 11 and 12 [Refer Supplementary Material No. 8]. Figures 10, 11 and 12 conclude that less $\bar{c}$ & $\bar{m}$ are performed for small and intermediate graphs; $\bar{c}$ & $\bar{m}$ are found to be increasing for the large graphs. The average computing time of χ(G) for various categories of graphs is compared and presented in Fig. 13. Figure 13 indicates that the computing time is higher for the low-density graphs and is lower for high-density graphs.

6 Conclusions

The GCP is applied in some real-world problems such as bandwidth assignment, register allocation, scheduling, image segmentation and noise reduction in circuits. Genetic algorithms are adaptive heuristic search algorithms based on evolutionary strategies of natural selection and genetics. Compared to several existing methods, genetic algorithms are applied to solve optimization problems in the vast search space in obtaining near optimal solution.

Attempts are made to devise new genetic operators to solve GCP with reduced computation by avoiding the pitfalls in the well-known methods. In our earlier works we devised the operators SPCGX and CGR. In this paper, these operators are extended to exhibit three new genetic and Tabu search procedures that apply SPCGEX & CGM, ESPCGEX & ECGM and MSPCGX & MCGM. Experiments were conducted on small, intermediate and large benchmark graphs. The proposed ALGS operator is efficient in finding good solutions in a particular region of the search space. ALGS with Tabu search offers a good compromise between computational complexity and the solution quality. Two better gene sequences are considered using fitness proportionate selection in order to minimize $\bar{g}$. An attempt has been made to devise multipoint crossover and mutation operators to reduce the search space. The existing genetic algorithms perform crossover operation on multiple parents which requires more computation. This paper performs enhanced genetic operations on single parent which are expected to achieve fast stochastic convergence in finding the better near optimal solution compared to existing methods. The results obtained in proposed methods are found to be promising for different categories of graphs including random graphs. Our methods reduce the bound for *minimum colors* obtained so far for certain families of graphs. Compared with some of the existing methods, MSPCGX & MCGM stochastically converge to *flat1000_76_0.col* and also minimize the *minimum color* of *latin_square_10.col* to 98. It has been noticed that the devised methods work efficiently even for minimal population size ($N \leq 15$). The devised operators play a major role in achieving monotonically decreasing the values of fitness function and hence minimize $\bar{g}$ toward reaching the fast stochastic convergence. For most of the small and large benchmark graphs tested in this work, MSPCGX & MCGM satisfy $2 \leq \bar{g} \leq \Delta(G)/2^x$ such that $0 \leq x \leq \delta(G)$. The graph density also affects the output performance measurements of the devised genetic procedures. Experimental results conclude that higher $\bar{g}$ is obtained for low-density graphs and lower $\bar{g}$ is obtained when density increases. The stochastic convergence of the proposed genetic operators to reach an optimal solution is proportional to the graph instances and its density. Less average crossovers $\bar{c}$ and the average mutations $\bar{m}$ are performed for small and intermediate graphs, while $\bar{c}$ and $\bar{m}$ are found to be increasing for the large graphs.

The following research directions are opened up by the present work:

- SPCGEX & CGM, ESPCGEX & ECGM and MSPCGX & MCGM with ALGS operators can be combined with meta-heuristics further for search space reduction in solving GCP and will also be applied in solving channel allocation problem.
- It would be interesting to analyze the behavior of proposed operators with the selection of more than two gene sequences and applying the proposed genetic operators in parallel.
- The proposed operators can be extended to swarm intelligence further to reduce the expected generations.
- These proposed genetic operators can be refined by applying suitable statistical measurements to minimize the computational effort and to analyze if the characterization of input instances can have an impact on general optimization problem solving methods. Refined statistical genetic operators will also be applied in image segmentation applications.

Acknowledgements The authors would like to acknowledge the support rendered by the Management of SASTRA University for providing the financial support and BRNS, India for providing the Workstation, which was procured for the project No. 2013/36/40-BRNS/2305 for execution. The authors would like to thank Gary Lewandowski and Michael Trick for uploading the graph repository in World Wide Web Consortium.

Compliance with ethical standards

Conflict of interest The authors declare that there is no conflict of interests regarding the publication of this paper.

References

1. Balakrishnan, R.; Ranganathan, K.: A Textbook of Graph Theory, 1st edn. Springer, New York (2000)
2. Garey, M.R.; Johnson, D.S.: Computers and Intractability: A Guide to the Theory of NP-Completeness. W. H. Freeman and Company, New York (1979)
3. Maitra, T.; Pal, A.J.; Bhattacharyya, D.; Kim, T.: Noise reduction in VLSI circuits using modified GA based graph coloring. Int. J. Control Autom. **3**(2), 37–44 (2010)
4. Yoshino, J.; Ohtomo, I.: Study on efficient channel assignment method using the genetic algorithm for mobile communication systems. Soft Comput. **9**(2), 143–148 (2005)
5. Chen, X.; Zu, Y.; Xu, J.; Wang, Z.; Yao, B.: Vertex-distinguishing E-total colorings of graphs. Arab. J. Sci. Eng. **36**(8), 1485–1500 (2011)
6. Abdelfattah, M.; Shawish, A.: Automated academic schedule builder for University's faculties. In: Proceedings of the World Congress on Engineering (2016)
7. Saharan, S.; Kumar, R.: Graph coloring based optimized algorithm for resource utilization in examination scheduling. Appl. Math. Inf, Sci (2016)

8. Tawfiq, F.M.O.; Al-qahtani, K.K.S.: Graph coloring applied to medical doctors schedule. In: The 10th International Conference on Advanced Engineering Computing and Applications in Sciences (2016)
9. Thevenin, S.; Zufferey, N.; Potvin, J.-Y.: Graph multi-coloring for a job scheduling application. CIRRELT (2016)
10. Mehrotra, A.; Trick, M.A.: A column generation approach for graph coloring. Informs J. Comput. **8**, 344–354 (1995)
11. Galinier, P.; Hertz, A.: A survey of local search methods for graph coloring. Comput. Oper. Res. **33**(9), 2547–2562 (2006)
12. Johnson, D.S.; Aragon, C.R.; McGeoch, L.A.; Schevon, C.: An experimental evaluation; Part II, Graph coloring and number partitioning. Oper. Res. **39**(3), 378–406 (1991)
13. Johnson, D.S.; Trick, M.A.: Cliques, coloring, and satisfiability. Am. Math. Soc. **26**, 1–8 (1993)
14. Mizuno, K.; Nishihara, S.: Constructive generation of very hard 3-colorability instances. Discr. Appl. Math. **156**, 218–229 (2008)
15. Méndez-Díaz, I.; Zabala, P.: A Branch and Cut algorithm for graph coloring. Discr. Appl. Math. **154**, 826–847 (2006)
16. Monasson, R.: On the analysis of backtrack procedures for the coloring of random graphs. Lect. Notes Phys. **650**, 235–254 (2004)
17. Eiben, A.E.; Van Der Hauw, J.K.; Van Hemert, J.I.: Graph coloring with adaptive evolutionary algorithms. J. Heuristics **4**, 25–46 (1998)
18. Bui, T.N.; Nguyen, T.H.; Patel, C.M.; Phan, K.-A.T.: An ant-based algorithm for coloring graphs. Discr. Appl. Math. **156**, 190–200 (2008)
19. Hsu, L.-Y.; Horng, S.-J.; Fan, P.; Khan, M.K.; Wang, Y.-R.; Run, R.-S.; Lai, J.-L.; Chen, R.-J.: MTPSO algorithm for solving planar graph coloring problem. Expert Syst. Appl. **38**, 5525–5531 (2011)
20. Cui, G.; Qin, L.; Liu, S.; Wang, Y.; Zhang, X.; Cao, X.: Modified PSO algorithm for solving planar graph coloring problem. Progr. Nat. Sci. **18**, 353–357 (2008)
21. Zhou, Y.; Zheng, H.; Luo, Q.; Wu, J.: An improved cuckoo search algorithm for solving planar graph coloring problem. Appl. Math. Inf. Sci. **7**(2), 785–792 (2013)
22. Prestwich, S.: Generalised graph colouring by a hybrid of local search and constraint programming. Discr. Appl. Math. **156**, 148–158 (2008)
23. Dukanovic, I.; Rendl, F.: A semidefinite programming-based heuristic for graph coloring. Discr. Appl. Math. **156**, 180–189 (2008)
24. Fleurent, C.; Ferland, J.A.: Genetic and hybrid algorithms for graph coloring. Ann. Oper. Res. **63**(3), 437–461 (1996)
25. Mumford, C.L.: New order based crossovers for the graph coloring problem. Parallel Probl. Solving Nat. **4193**, 880–889 (2006)
26. Han, L.; Han, Z.: A novel bi-objective genetic algorithm for the graph coloring problem. In: 2nd International Conference on Computer Modeling and Simulation (2010)
27. Marappan, R.; Sethumadhavan, G.: A new genetic algorithm for graph coloring. In: 5th International Conference on Computational Intelligence, Modelling and Simulation, pp. 49–54 (2013)
28. Sethumadhavan, G.; Marappan, R.: A genetic algorithm for graph coloring using single parent conflict gene crossover and mutation with conflict gene removal procedure. In: IEEE International Conference on Computational Intelligence and Computing Research, pp. 350–355 (2013)
29. Lewis, R.M.R.: A Guide to Graph Coloring. Algorithms and Applications. Springer, Berlin (2016)
30. Hernández, H.; Blum, C.: FrogSim: distributed graph coloring in wireless ad hoc networks. Telecommun. Syst. **55**(2), 211–223 (2014)
31. San Segundo, P.: A new DSATUR-based algorithm for exact vertex coloring. Comput. Oper. Res. **39**(7), 1724–1733 (2012)
32. Hertz, A.; de Werra, D.: Using tabu search techniques for graph coloring. Computing **39**(4), 345–351 (1987)
33. Demange, M.; Ekim, T.; Ries, B.; Tanasescu, C.: On some applications of the selective graph coloring problem. Eur. J. Oper. Res. **240**, 307–314 (2014)
34. Demange, M.; Ekim, T.; Ries, B.: On the minimum and maximum selective graph coloring problems in some graph classes. Discr. Appl, Math (2015)
35. Takeshita, L.: Coloring signed graphs, pp. 1–7 (2016). https://math.mit.edu/~apost/courses/18.204-2016/18.204_Lynn_Takeshita_final_paper.pdf
36. Macajov, E.; Raspaud, A.; Skoviera, M.: The chromatic number of a signed graph. Cornell University Library, pp. 1–9 (2016). arXiv:1412.6349
37. Zhou, B.: On the maximum number of dominating classes in graph coloring. Open J. Discr. Math **6**, 70 (2016)
38. Gaspers, S.; Lee, E.J.: Faster graph coloring in polynomial space. Cornell University Library, pp. 1–18 (2016). arXiv:1607.06201
39. Angelini, P.; Bekos, M.A.; De Luca, F.; Didimo, W.; Kaufmann, M.; Kobourov, S.; Montecchiani, F.; Raftopoulou, C.N.; Roselli, V.; Symvonis, A.: Vertex-coloring with defects. J. Graph Algorithms Appl. **21**(3), 313–340 (2017)
40. Galán, S.F.: Simple decentralized graph coloring. Comput. Optim. Appl. **66**(1), 163–185 (2017)
41. Aslan, M.; Baykan, N.A.: A performance comparison of graph coloring algorithms. Int. J. Intell. Syst. Appl, Eng (2016)
42. Galinier, P.; Hao, J.-K.: Hybrid evolutionary algorithms for graph coloring. J. Comb. Optim. **3**(4), 379–397 (1999)
43. Rudolph, G.: Finite markov chain results in evolutionary computation: a tour d'Horizon. Fundam Inform **35**(1–4), 67–89 (1998)
44. Saha, S.; Kumar, R.; Baboo, G.: Characterization of graph properties for improved Pareto fronts using heuristics and EA for bi-objective graph coloring problem. Appl. Soft Comput. **13**(5), 2812–2822 (2013)
45. Szép, T.; Mann, Z.Á.: Graph coloring: The more colors, the better? In: 11th International Symposium on Computational Intelligence and Informatics (CINTI) (2010)