mathematics

MDPI

Article

Complexity Analysis and Stochastic Convergence of Some Well-known Evolutionary Operators for Solving Graph Coloring Problem

Raja Marappan * and Gopalakrishnan Sethumadhavan

Department of Computer Applications, School of Computing, SASTRA Deemed University, Tirumalaisamudram 613401, India; sgk@mca.sastra.edu

* Correspondence: raja_csmath@cse.sastra.edu

Received: 20 January 2020; Accepted: 23 February 2020; Published: 25 February 2020

Abstract: The graph coloring problem is an *NP*-hard combinatorial optimization problem and can be applied to various engineering applications. The chromatic number of a graph G is defined as the minimum number of colors required to color the vertex set $V(G)$ so that no two adjacent vertices are of the same color, and different approximations and evolutionary methods can find it. The present paper focused on the asymptotic analysis of some well-known and recent evolutionary operators for finding the chromatic number. The asymptotic analysis of different crossover and mutation operators helps in choosing the better evolutionary operator to minimize the problem search space and computational complexity. The choice of the right genetic operators facilitates an evolutionary algorithm to achieve faster convergence with lesser population size N through an adequate distribution of promising genes. The selection of an evolutionary operator plays an essential role in reducing the bounds for *minimum color* obtained so far for some of the benchmark graphs. This research also focuses on the necessary and sufficient conditions for the global convergence of evolutionary algorithms. The stochastic convergence of recent evolutionary operators for solving graph coloring is newly analyzed.

Keywords: approximation methods; chromatic number; combinatorial optimization; complexity analysis; evolutionary approach; genetic algorithm; graph coloring; *NP*-hard; stochastic convergence

1. Introduction

A simple graph $G = (V, E)$ consists of a set of vertices $V = \{v_1, v_2, v_3 \ldots v_n\}$ and a set of edges $E = \{e_1, e_2, e_3 \ldots e_m\}$ such that each e_i $(1 \le i \le m)$ is uniquely associated with an unordered pair of vertices (v_j, v_k) $(1 \le j, k \le n)$ and $j \ne k$ [1,2]. Its adjacency matrix is denoted by $A(G)$, an $n \times n$ symmetric matrix where $A(j, k) = 1$ $(1 \le j, k \le n)$ if $(v_j, v_k) \in$ E(G); and A$(j, k) = 0$ $(1 \le j, k \le n)$ otherwise [2].

$\chi(G)$, the chromatic number of G, is defined as the minimum number of colors required for $V(G)$ so that no two adjacent vertices are of the same color [2]. The graph coloring problem (GCP) finds the value for $\chi(G)$ and applies it to register allocation, channel assignment, image segmentation, resource utilization, and scheduling [3–10]. With the increasing values of n, the complexity of determining $\chi(G)$ also increases. GCP is an *NP*-hard combinatorial optimization problem. Hence, a fast evolutionary and approximation method is expected to reduce the problem search space by maximizing the number of successful runs [1,11–16]. Tabu search [17], backtracking [18], branch and bound [11], evolutionary algorithm [19–21], branch and cut [22], particle swarm optimization (PSO) [23–26], ant colony optimization (ACO) [27], local and cuckoo search [28,29] are some existing methods for finding $\chi(G)$. Some recent applications of GCP are selective graph coloring [30,31], signed graphs coloring [32,33], scheduling, and resource allocation [7–10,34–38]. In comparison to other methods, GA is useful for solving multi-objective optimization problems with vast search space. Hence, it is

expected to design new evolutionary operators to achieve faster stochastic convergence with minimal generations. Some recent evolutionary operators have achieved near-optimal solutions for some benchmark graphs [4,39–41]. Moreover, some recent genetic operators have increased the number of successful runs and reduced expected generations $\overline{g}$, expected crossovers $\overline{c}$, and the expected mutations $\overline{m}$. The notations $\overline{g}$, $\overline{c}$, and $\overline{m}$ indicate respectively the average number of generations, the average number of crossovers, and the average number of mutations performed to obtain the near-optimal solution for the specified number of total runs and generations during the execution of the program. This paper focused on the asymptotic analysis and stochastic convergence of some well-known and current evolutionary operators.

Some of the recent recombination operators are similar to mutation and are executed in place of recombination operation [2]. Some of the well-known and recent recombination operators operate on multiple parent gene sequences [2]. The order-based uniform crossover operator [42] uniformly selects genes from various parents. The penalty based crossover operator [43] generates offspring, which results in a minimum penalty. The one-point and two-point crossover operators generate random numbers and inherit genes from their parents. The graph adapted crossover generates genes based on conflicting and neighboring vertices. The union independent set (UIS) crossover [44] operator unifies the pairs of independent color sets from the selected parents. The greedy partition crossover (GPX) [45] operator divides $V(G)$ into k-sets and generates offspring. The multi-parent crossover [46] operator is the extension of GPX, which creates offspring from multiple parents. The well-informed partition crossover (WIPX) [47] operator selects the color classes from the randomly chosen parents based on a scoring function. The penalty-based color partitioning crossover (PCPX) [48] operator generates offspring based on the partition of $V(G)$ and its penalty. The degree based crossover (DBX) [48] operator applies the heuristics of the largest degree ordering (LDO) to order-based crossover or permutation one point crossover (POP) [40]. POP crossover performs an order-based permutation operation. The multi-point guided crossover (MPGX) [48] operator incorporates problem-specific knowledge. The merging crossover (MOX) [40] operator merges two-parent gene sequences and generates order-based offspring. The merging independent sets (MIS) [40] crossover groups the color sets of parents and generates offspring.

A recent crossover operator, the single parent conflict gene crossover (SPCGX), similar to mutation, is applied to the single parent to identify the conflicting genes and generate better offspring [49]. SPCGX is combined with the conflict gene removal (CGR) procedure to identify and remove some conflicting genes [50]. SPCGX is also connected with the advanced local guided search (ALGS) operator [2] to fine-tune the offspring further and also to reduce the computational complexity. The single parent conflict gene extended crossover (SPCGEX) operator is applied to the selected single parent for fixed iterations. Extended SPCGEX (ESPCGEX) extends SPCGEX with CGR for fixed iterations to produce better offspring. The multipoint SPCGX (MSPCGX) performs several crossovers for conflicting genes. Some of the recent genetic operators have achieved the better solution for some large DIMACS benchmark graphs in the reasonable expected generations [2].

The mutation operators operate on offspring generated by crossover operations. The random mutation operator [43] randomly swaps the color of each vertex with low mutation probability p_m. The polynomial mutation operator rounds off the floating-point color values. The swap mutation is applied to offspring generated by the DBX operator [51]. Problem-specific mutation [51] operators operate on the offspring generated by the MPGX operator. The order mutation (OM) [52] operator generates a random number r and performs r interchanges between vertices. The block mutation (BM) [52] performs the translation of blocks of successive vertices. The color spread mutation (CSM) [52] randomly selects the conflicting edges and moves the genes to new positions. The bad edge stretch mutation (BESM) [52] reduces the number of conflicting edges. The conflict gene mutation (CGM) [2] assigns the conflict-free integers to the offspring and performs fixed iterations. The CGM-CGR operator further reduces the number of conflicts in the offspring. The extended CGM (ECGM) operator fine-tunes the offspring by

extending CGM for finite iterations. The multipoint CGM (MCGM) performs multipoint mutations at the conflicting vertices to reduce the number of conflicts.

Section 2 explains the need for complexity analysis and stochastic convergence for evolutionary operators. Section 3 focuses on the general structure of the genetic algorithm for solving GCP. Sections 4 and 5 explain some of the well-known and recent recombination and mutation operators. The necessary and sufficient condition for the global convergence of evolutionary algorithms and the stochastic convergence of recent genetic operators are newly analyzed in Section 6. Conclusions and future research directions are delineated in Section 7.

2. The Need for Complexity Analysis and Stochastic Convergence of Evolutionary Operators

GCP is very challenging for genetic algorithms because of its vast solution space. Hence, designing and choosing the right genetic operators in population-based methods are important for the following reasons [53–56]:

1. Solution space of GCP consists of n^n candidate solutions, and n^n rapidly grows with n. The computational complexity of finding $\chi(G)$ is proportional to m and n. If the solutions are represented using $\chi(G) < n$, then the solution space contains $\chi(G)^n$ assignments. Therefore, the total number of different assignments in the solution space is $\chi(G)!$
2. The computational complexity of GCP is reduced only when the values of $\bar{c}$, $\overline{m}$, and $\overline{g}$ are minimized.
3. The evolutionary operators are expected to improve the performance of the genetic algorithm by reducing the search space as well as by increasing the successful runs.
4. The evolutionary operators should improve the near-optimal solution obtained from the existing methods [11,22,25–27,57–59].
5. The evolutionary operators should avoid additional memory resources and complex computational operations.
6. The performance of operators depends on how many parent gene sequences are considered for crossover and mutation. It also depends on the update of gene sequences [4,39–41].
7. A smaller population size ($N \leq 15$) is expected to reduce the complexity and to achieve a sufficient convergence rate. The crossover and mutation operations can be embedded with the right search operators to reduce the value of $\overline{g}$ further.
8. The evolutionary operators should effectively distribute the promising genes from one generation to the next generation in order to reduce the fitness function values.
9. The graph characteristics, graph instances such as n, m, and graph density, can be considered for choosing the right genetic operators [60].
10. The evolutionary operators should quickly converge with the reduced computational complexity.

3. The General Structure of Genetic Algorithm for Solving GCP

Notations

1. Degree, $\delta(G)$ and $\Delta(G)$ The number of edges incident on v_i ($1 \leq i \leq n$) is the degree, $d(v_i)$ in G. Let $\Delta(G) = \max_{1 \leq i \leq n}\{d(v_i)|v_i \in V(G)\}$ and $\delta(G) = \min_{1 \leq i \leq n}\{d(v_i)|v_i \in V(G)\}$ be the maximum and minimum degrees of G respectively.
2. Encoding of Gene Sequence The color values of $V(G)$ are encoded as (g_1, g_2, g_3 ... g_n), where g_j ($1 \leq j \leq n$) is a non-zero integer.
3. Density The density of G is $\frac{2m}{n(n-1)}$.
4. Population Size, Crossover, Mutation Probability p_c is the crossover probability, and the mutation probability is denoted as p_m. P_g is the population, which represents the collection of gene sequences for generation g. $N = |P_g|$ defines the population size for g.

5. Fitness Function $f(G)$, the general fitness function of G, is the number of distinct genes in the gene sequence [2]. The main objective of GCP is to minimize $f(G)$, hence, $\chi(G) - f(G) = 0$.
6. Conflicting Genes When $g_a = g_b$ and $(v_a, v_b) \in E(G)$, the genes g_a $(1 \leq a \leq n)$ and g_b $(1 \leq b \leq n)$ corresponding to vertices v_a and v_b conflict with the gene sequence $(g_1, g_2, g_3 \ldots g_n)$.

The general structure of the genetic algorithm for solving GCP is presented below.

Step 1: Initialization

First, initialize $g = 0$. The initialization of P_g randomly or heuristically generates values in [1, *minimum color*].

Step 2: Evaluation of Fitness Function and Selection of Individuals

The fitness of individuals in the population is evaluated, and one or more parent gene sequence(s) are selected for the generation of the subsequent gene sequences. Different selection methods can be applied to select gene sequences for crossover operation.

Step 3: Recombination or Crossover Operation

The recombination operation is performed on the selected gene sequence(s) with probability p_c (high value of p_c, for example, $p_c = 0.8$) to generate offspring.

Step 4: Performing Mutation

The crossover gene sequences or offspring are mutated with the chosen mutation probability p_m (low value of p_m, for example, $p_m = 0.2$) to generate new offspring.

Step 5: Termination

Find F_g (offspring) for the updated gene sequence. If F_g (offspring) = 0, the generation of offspring stops; otherwise, P_g is updated ($g = g + 1$). If F_g (offspring) < F_{g-1} (worst), P_g is updated by replacing the worst gene with offspring. Then transfer the control to Step 2 for the next generation.

The pseudo-code of the general genetic algorithm is presented below.

// Pseudo code of the Genetic Algorithm

```
1: Initialization of individuals;
2: While termination conditions are not satisfied, do
3:    Evaluate the fitness function and select better individuals;
4:    Perform the recombination operation;
5:    Perform the mutation operation;
6:    Apply elitism operation;
7: Print the near-optimal results;
```

4. Asymptotic Analysis of Some Existing Well-known and Recent Crossover Operators

Some of the existing well-known and recent crossover or recombination operators for solving GCP are presented in this section. The asymptotic complexity of these operators is also analyzed in Table 1. The simple control constructs if and if-else statements that can be implemented in a constant complexity, $\Theta(c)$. The control constructs *for*, *while*, *do-while*, and *repeat-until* can be implemented in a linear complexity, $\Theta(n)$. The quadratic complexity is represented in $\Theta(n^2)$. For example, color assignments of $V(G)$ can be verified in $\Theta(n^2)$ complexity.

1. Order-based Uniform Crossover

The order-based uniform crossover operator uniformly selects the genes from multiple parents. The procedure can be described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Generate the offspring by selecting the first $n/2$ length of genes from p_1 and the next $n/2$ length of genes from p_2.
c. Randomly select any two vertices, v_{i} and v_{j} in the offspring, and scramble all vertices between v_{i} and v_{j}.

Analysis: Steps (a), (b) and (c) take $\Theta(n)$ complexity. Thus the order-based uniform crossover takes $\Theta(n)$ complexity.

2. Penalty Based Crossover

The penalty based crossover operator generates the offspring, which results in a minimum penalty. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Generate the offspring by determining the color for each vertex in every parent by assigning colors, which results in a minimum penalty.

Analysis: Step (a) takes $\Theta(n)$ complexity. Step (b) takes $\Theta(n^2\ \chi(G))$ complexity. The computational complexity of penalty based crossover is $\Theta(n^2\ \chi(G))$.

3. One-point Crossover

The one-point crossover operator generates random numbers and inherits genes from their parents. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Choose a random point r in between $[1, n]$.
c. Generate the offspring based on r by inheriting genes from both parents.

Analysis: Step (a) takes $\Theta(n)$ complexity. Step (b) can be implemented in constant running time. Step (c) requires $\Theta(n)$ complexity. Thus the one-point crossover takes $\Theta(n)$ complexity.

4. Two-point Crossover

The two-point crossover operator generates random numbers and inherits genes from their parents. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Choose two random points, r and s in between $[1, n]$.
c. Generate the offspring based on r and s by inheriting genes from both parents.

Analysis: Step (a) takes $\Theta(n)$ complexity. Step (b) can be implemented in constant running time. Step (c) requires $\Theta(n)$ complexity. Thus the two-point crossover takes $\Theta(n)$ complexity.

5. Graph Adapted Crossover

The graph adapted crossover generates genes based on the conflicting and neighboring vertices. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Choose a random vertex v in $V(G)$.
c. Color the vertices with respect to the parents that have no conflicts for vertex v.
d. If p_1 and p_2 have conflicts with vertex v then assign the least color to its neighboring vertices.

Analysis: Step (a) takes $\Theta(n)$ complexity. Step (b) requires constant running time. Step (c) requires $\Theta(n^2)$ complexity. Step (d) requires $\Theta(n^2)$ complexity. The complexity of graph adapted crossover is $\Theta(n^2)$.

6. UIS Crossover

The UIS crossover operator unifies a pair of independent color sets from the selected parents. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Find the largest conflict-free subset of vertices having a color c in p_1.
c. Find the largest conflict-free subset in p_2 such that a common number of vertices in these subsets are maximal.
d. Repeat the steps (b) and (c) for every color.
e. Generate the offspring o_1 and o_2 by performing set union operation.

Analysis: Steps (a), (b) and (c) require $\Theta(n)$ complexity. Step (d) requires $\Theta(n\ \chi(G))$ complexity. Step (e) requires $\Theta(n)$ complexity. Thus the complexity of UIS crossover is $\Theta(n\ \chi(G))$.

7. Multi-Parent Crossover

Multi-parent crossover is an extension of GPX, which creates offspring from multiple parents. The procedure is described in the following steps.

a. Select the parent color classes from multiple parents.
b. Select the largest color class c in the population.
c. Add the color class c to the offspring.
d. Remove all vertices in c from all parent color classes.
e. Repeat the operation until the required number of color classes has reached.
f. Add the remaining vertices randomly to the color class.

Analysis: Steps (a), (b), (c) require $\Theta(n)$ complexity. Steps (d) and (e) require $\Theta(n^2)$ complexity. Step (f) requires $\Theta(n)$ complexity. Thus the complexity of multi-parent crossover is $\Theta(n^2)$.

8. WIPX Crossover

The WIPX operator selects the color classes from the randomly selected parents based on a scoring function. The procedure is described in the following steps.

a. Select the color classes from the randomly chosen parents based on a scoring function.
b. Determine the score of the color set as:

$$\text{Number of conflicts} - \frac{1}{|\mathrm{v}|}(\text{classSize} + \text{classDegree}/(\mathrm{n} * \mathrm{m}))$$

where classSize is the cardinality of the color set, and classDegree is the sum of the degrees of the color class vertices.
c. Apply the multi-parent crossover operation.

Analysis: Step (a) requires $\Theta(n)$ complexity. Step (b) requires $\Theta(n^2)$ complexity. Step (c) requires $\Theta(n^2)$ complexity. Thus the complexity of WIPX crossover is $\Theta(n^2)$.

9. GPX Crossover

GPX operator partitions $V(G)$ into k sets and generates the offspring. The procedure is described in the following steps.

a. Choose two parent configurations $p_1 = ({v_1}^1, {v_2}^1, {v_3}^1 \ldots {v_k}^1)$ and $p_2 = ({v_1}^2, {v_2}^2, {v_3}^2 \ldots {v_k}^2)$ where k is the number of color classes.

b. For each j ($1 \leq j \leq k$) do If j is odd then s =1, else s = 2; Select i such that ${v_i}^s$ has maximum vertices; $v_j = {v_i}^s$; Remove v_j from both p_1 and p_2
c. Choose a random color class and assign $V-(v_1 \cup v_2 \cup v_3 \ldots \cup v_k)$.
d. Generate an offspring $o_1 = (v_1, v_2, v_3 \ldots v_k)$.

Analysis: Step (a) requires $\Theta(n)$ complexity. Step(b) requires $\Theta(n^2)$ complexity. Steps (c) and (d) require $\Theta(n)$ complexity. Thus GPX crossover takes $\Theta(n^2)$ complexity.

10. PCPX Crossover

PCPX crossover operator generates offspring based on partitions of $V(G)$ and its penalty. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Consider one parent at a time.
c. Extract a subpartition $V'(G)$, which results in a minimum penalty among $V(G)$ of the main partition.
d. Color $V'(G)$ for the selected parent.
e. Remove $V'(G)$ in both parents.
f. Find the largest color partition for the next parent.
g. Repeat the procedure until all vertices in the offspring are colored.

Analysis: Step (a) requires $\Theta(n)$ complexity. Step(b) takes constant running time. Steps (c), (d), (e), (f) and (g) take $\Theta(n^2)$ complexity. Thus PCPX crossover takes $\Theta(n^2)$ complexity.

11. DBX Crossover

DBX crossover operator applies the heuristics of LDO in order based crossover or POP. The procedure is described in the following steps.

a. Apply the heuristics of LDO in order based crossover or POP operator.
b. Select a pair of random individuals, say P_1 and P_2.
c. Select a random crossover point r in P_1.
d. Copy the vertex colors starting at an initial point to r in P_1 into the offspring O_1;
e. Choose a vertex u in P_2 and v in O_1.
f. If u is not assigned an order in O_1 and if degree(v) < degree(u) then Find the vertex v having minimum order in O_1 with degree(v) < degree(u); Increment the vertex orders by 1 for all vertices having order $\geq v$; Assign vertex u to the order of vertex v; Else assign the least available order to vertex u.

Analysis: Step (a) requires $\Theta(n^2)$ complexity. The selection and copy operations in steps (b), (d), and (e) require $\Theta(n)$ complexity. Step(c) takes a constant running time. Step (f) requires $\Theta(n^2)$ complexity. Thus DBX takes $\Theta(n^2)$ complexity.

12. MPGX Crossover

MPGX operator incorporates problem-specific knowledge. The procedure is described in the following steps.

a. Select a pair of random individuals, say P_1 and P_2;
b. Choose a random crossover point;
c. Assign count = Pc * n;
d. While count ≠ 0 do Select a random vertex i; c_1= color[i] in P_1; c_2 = color[i] in P_2; If $c_1 \neq c_2$ and if penalty of P_1 reduces with c_2 then replace c_1 by c_2; If $c_1 \neq c_2$ and if penalty of P_2 reduces with c_1 then replace c_2 by c_1; count = count − 1

e. Generate offspring.

Analysis: Step (a) requires $\Theta(n)$ complexity. Steps (b), (c), and (e) take a constant running time. Step (d) takes $\Theta(n^2)$ complexity. Thus MPGX crossover takes $\Theta(n^2)$ complexity.

13. POP Crossover

POP crossover performs an order based permutation operation. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Choose a random point r in between [1, n].
c. Apply order based crossover.
d. Swap the first portion of strings and apply permutation on the remaining unused vertices copied in the sequence.

Analysis: Steps (a) and (c) require $\Theta(n)$ complexity. Step (b) requires a constant running time. Step (d) requires $\Theta(n^2)$ complexity. Thus the complexity of POP crossover is $\Theta(n^2)$.

14. MOX Crossover

The MOX operator performs merging of two-parent gene sequences and generates the order based offspring. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Merge p_1 and p_2 randomly into a single list, which consists of $2n$ elements.
c. Assign the first occurrence of each gene to offspring$_1$.
d. Assign the second occurrence of each gene to offspring$_2$.

Analysis: Steps (a) and (b) require $\Theta(n)$ complexity. Steps (c) and (d) take $\Theta(n^2)$ complexity. The complexity of MOX crossover is $\Theta(n^2)$.

15. MIS Crossover

The MIS crossover operator groups the color sets of parents and generates the offspring. The procedure is described in the following steps.

a. Select two parent gene sequences, say p_1 and p_2.
b. Group the color sets in both p_1 and p_2.
c. Copy the whole color sets from p_1 and p_2 into a single merged list.
d. Assign the first occurrence of each gene into offspring$_1$.
e. Assign the second occurrence of each gene into offspring$_2$.

Analysis: Steps (a), (b) and (c) require $\Theta(n)$ complexity. Steps (d) and (e) take $\Theta(n^2)$ complexity. The complexity of MIX crossover is $\Theta(n^2)$.

16. SPCGX Crossover

SPCGX operation is applied to the selected gene sequences, i and j, to generate two new gene sequences as offspring. The offspring, i' and j' are generated by a chosen crossover probability p_c. SPCGX operator is defined as follows [50]:

a. Let $i = (i_1, i_2, i_3 \ldots i_q \ldots i_r \ldots i_n)$ and $j = (j_1, j_2, j_3 \ldots j_s \ldots j_t \ldots j_n)$ be the two selected sequences.
b. Generate random probability $p_{cr.}$
c. If $p_{cr} > p_c$ go to step (d).

d. Find all pairs of conflicting edges in i and j.
e. Let (q, r) and (s, t) be such two pairs of conflicting edges (i.e., $i_q = i_r$ and $j_s = j_t$) in i and j, then generate two offspring: $(i_1, i_2, i_3 \ldots i_q \ldots i_r{+}1 \ldots i_n)$ and $(j_1, j_2, j_3 \ldots j_s \ldots j_t{+}1 \ldots j_n)$.
f. Repeat step (e) of crossover operation to all the identified conflicting pairs in step (d) of crossover operation and generate updated gene sequences $i' = (i_{1'}, i_{2'}, i_{3'} \ldots i_{f'} \ldots i_{g'} \ldots i_{n'})$ and $j' = (j_{1'}, j_{2'}, j_{3'} \ldots j_{k'} \ldots j_{l'} \ldots, j_{n'})$.

Analysis: Step (a) requires $\Theta(n)$ complexity. Steps (b), (c), and (e) take a constant running time. Steps (d) and (f) require $\Theta(n^2)$ complexity. SPCGX takes $\Theta(n^2)$ complexity.

17. SPCGX-CGR Crossover

SPCGX is applied with a CGR procedure with a chosen p_c. SPCGX-CGR is given in the following steps [50]:

a. Apply the steps (a) to (e) of SPCGX crossover.
b. Check if the new genes at r and t (i.e., i_r and j_t) corresponding to an edge (r, t) are in conflict with its corresponding adjacent genes. If so, generate $i_r = i_r + 1$ and $j_t = j_t + 1$ such that $i_r + 1 \leq \chi(G)$ and $j_t + 1 \leq \chi(G)$.
c. Repeat step (f) of SPCGX with CGR procedure to all the conflicting pairs and generate updated gene sequences: $i' = (i_{1'}, i_{2'}, i_{3'} \ldots i_{n'})$ and $j' = (j_{1'}, j_{2'}, j_{3'} \ldots j_{n'})$.

Analysis: Steps (a) and (c) require $\Theta(n^2)$ complexity. Step (b) takes a constant running time. SPCGX-CGR takes $\Theta(n^2)$ complexity.

18. SPCGEX Crossover

The SPCGEX operator is applied on the selected gene sequences $i = (i_1, i_2, i_3 \ldots i_n)$ and $j = (j_1, j_2, j_3 \ldots j_n)$ as follows [2]:

a. Randomly select any two most conflicting vertices, r, and t in gene sequences i and j.
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 $1 + j_t \leq$ *minimum color*.
c. Repeat the steps (a) and (b) for fixed iterations to obtain the updated offspring i' and j' as $(i_{1'}, i_{2'}, i_{3'} \ldots i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'} \ldots j_{n'})$.

Analysis: Steps (a) and (c) require $\Theta(n^2)$ complexity. Step (b) takes a constant running time. SPCGEX takes $\Theta(n^2)$ complexity.

19. ESPCGEX Crossover

The steps in ESPCGEX crossover are defined as follows [2].

a. Perform SPCGEX crossover steps (a) and (b).
b. If the new genes i_r and j_t are conflicting, then apply SPCGEX at the conflicting edges (q, r) and (s, t).
c. Repeat the steps (a) and (b) for fixed iterations to obtain the updated offspring i' and j' as $(i_{1'}, i_{2'}, i_{3'} \ldots i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'} \ldots j_{n'})$.

Analysis: Steps (a), (b) and (c) require $\Theta(n^2)$ complexity. Hence ESPCGEX takes $\Theta(n^2)$ complexity.

20. MSPCGX Crossover

The steps in MSPCGX crossover are defined as follows [2].

a. Find the conflicting edges that count to every vertex in gene sequences j and i.

b. Determine the conflicting vertices in j and i.
c. Select arbitrarily conflicting vertices, r in i and t in j.
d. Update i and j as ($i_1, i_2, i_3 \ldots i_r{+}1 \ldots i_n$) and ($j_1, j_2, j_3 \ldots j_t{+}1 \ldots j_n$) such that *minimum color* $\geq i_r + 1$ and *minimum color* $\geq j_t + 1$.
e. If the new genes i_r and j_t are conflicting, then generate $i_r = i_r + 1$ and $j_t = j_t + 1$ such that minimum color $\geq i_r + 1$ and minimum color $\geq j_t + 1$.
f. Repeat the steps (d)–(e) for remaining conflicting vertices and update i' and j' as ($i_{1'}, i_{2'}, i_{3'} \ldots i_{n'}$) and ($j_{1'}, j_{2'}, j_{3'} \ldots j_{n'}$).

Analysis: Steps (a), (b), (e) and (f) require $\Theta(n^2)$ complexity. Step (c) takes $\Theta(n)$ complexity. Step (d) takes a constant running time. MSPCGX takes $\Theta(n^2)$ complexity.

Table 1. Asymptotic Complexity of Well-known and Recent Crossover Operators.

SNo	Crossover Operator	Asymptotic Complexity
1	Order-based Uniform Crossover	$\Theta(n)$
2	Penalty Based Crossover	$\Theta(n^2\,\chi(G))$
3	One-point Crossover	$\Theta(n)$
4	Two-point Crossover	$\Theta(n)$
5	Graph Adapted Crossover	$\Theta(n^2)$
6	UIS Crossover	$\Theta(n\,\chi(G))$
7	Multi-Parent Crossover	$\Theta(n^2)$
8	WIPX Crossover	$\Theta(n^2)$
9	GPX Crossover	$\Theta(n^2)$
10	PCPX Crossover	$\Theta(n^2)$
11	DBX Crossover	$\Theta(n^2)$
12	MPGX Crossover	$\Theta(n^2)$
13	POP Crossover	$\Theta(n^2)$
14	MOX Crossover	$\Theta(n^2)$
15	MIS Crossover	$\Theta(n^2)$
16	SPCGX Crossover	$\Theta(n^2)$
17	SPCGX-CGR Crossover	$\Theta(n^2)$
18	SPCGEX Crossover	$\Theta(n^2)$
19	ESPCGEX Crossover	$\Theta(n^2)$
20	MSPCGX Crossover	$\Theta(n^2)$

5. Asymptotic Analysis of Some Existing Well-known and Recent Mutation Operators

Some of the existing well-known and recent mutation operators for solving GCP are presented in this section. The asymptotic complexity of these operators is also analyzed. The asymptotic complexity of these operators is also analyzed in Table 2.

1. Random Mutation

The random mutation operator randomly swaps the color of each vertex with a low mutation probability p_m. The procedure is described in the following steps.

a. Select crossover offspring o_1.
b. Randomly swap the color class of each vertex with a low p_m.

Analysis: Step (a) requires a constant running time. Step (b) takes $\Theta(n)$ complexity. Thus the random mutation takes $\Theta(n)$ complexity.

2. Polynomial Mutation

The polynomial mutation operator rounds off real values of colors to the nearest integer. The procedure is described in the following steps.

a. Select crossover offspring o_1.
b. Choose the random point and round off real values of colors to the nearest integer.

Analysis: Steps (a) and (b) require a constant running time. Thus the polynomial mutation takes $\Theta(c)$ complexity.

3. Problem-Specific Mutation 1

The problem-specific mutation operator is defined with problem-specific knowledge. It is operating on the offspring generated by MPGX operator. The procedure is described in the following steps.

a. Select crossover offspring o_1.
b. Find the missing color c_1 in the range of maximum color [1, maximum color] used.
c. Identify all the vertices $V'(G)$ assigned with maximum color.
d. Replace colors of $V'(G)$ by c_1.

Analysis: Step (a) requires a constant running time. Steps (b) and (c) take $\Theta(n^2)$ complexity. Step (d) takes $\Theta(n)$ complexity. This mutation takes $\Theta(n^2)$ complexity.

4. Problem-Specific Mutation 2

The problem-specific mutation operator is defined with problem-specific knowledge. It operates on the offspring generated by MPGX operator. The procedure is described in the following steps.

a. Choose color c_1 randomly within one less than [1, maximum color].
b. Identify all the vertices $V'(G)$ assigned to a maximum color.
c. Replace colors of $V'(G)$ by c_1.

Analysis: Step (a) requires a constant running time. Step (b) takes $\Theta(n^2)$ complexity. Step (c) takes $\Theta(n)$ complexity. This mutation takes $\Theta(n^2)$ complexity.

5. Problem-Specific Mutation 3

The problem-specific mutation operator is defined with problem-specific knowledge. It operates on the offspring generated by MPGX operator. The procedure is described in the following steps.

a. Choose a vertex v randomly.
b. Choose a color c randomly for v.
c. If the penalty is reduced to v with color c, then set color[v] = c.

Analysis: Steps (a) and (b) require a constant running time. Step (c) takes $\Theta(n^2)$ complexity. This mutation takes $\Theta(n^2)$ complexity.

6. OM

The OM operator generates a random number r and performs r interchanges between vertices. The procedure is described in the following steps.

a. Let (v_1, v_2, v_3 … v_n) be the crossover offspring.
b. Generate a random number r;
c. Generate the offspring by performing r interchanges between vertices.

Analysis: Steps (a) and (b) require a constant running time. Performing r interchanges in Step(c) can also be done in linear time. The OM mutation takes $\Theta(n)$ complexity.

7. BM

The BM operation performs a translation of blocks of $k > 0$ successive vertices. The procedure is described in the following steps.

a. Let $(v_1, v_2, v_3 \ldots v_n)$ be the crossover offspring.
b. If $k = 2$ and $i \in [1, n-1]$, $j \in [1, n]$ are randomly generated then generate the offspring $(v_1, \ldots .v_{i-1}, v_{i+2} \ldots v_j, v_i, v_{i+1}, v_{j+1} \ldots)$.

Analysis: Step (a) requires a constant running time. Step (b) can take $\Theta(n)$ complexity. Thus BM operation takes $\Theta(n)$ complexity.

8. CSM

The CSM randomly selects the conflicting edge and moves the genes to new positions. The procedure is described in the following steps.

a. Let $(v_1, v_2, v_3 \ldots v_n)$ be the crossover offspring.
b. Choose a randomly selected bad edge $e = (v_i, v_j)$ and $e' = (v_k, v_l)$ be the next bad edge such that $(i < j < k < l)$.
c. Move the vertices from positions $i + 1$ to $l - 1$ to randomly chosen new positions.

Analysis: Step (a) requires a constant running time. Steps (b) and (c) need $\Theta(n)$ complexity.

9. BESM

The BESM operation reduces the number of conflicting edges. The procedure is described in the following steps.

a. Let $(v_1, v_2, v_3 \ldots v_n)$ be the crossover offspring.
b. Choose a random bad edge $e = (v_i, v_j)$.
c. Choose a direction either left-to-right or right-to-left randomly.
d. Let v_k be the farthest vertex adjacent to v_i so that $k > j$.
e. Let $e' = (v_l, v_m)$ be the first bad edge so that $l > k$.
f. If there is no vertex v_k or edge e' with the specified properties, the mutation is void. Otherwise, the vertex v_i is moved to the position $m + 1$.

Analysis: Steps (a), (c) require a constant running time. Step (b) takes $\Theta(n^2)$ complexity. Steps (d), (e) and (f) take $\Theta(n)$ complexity. Hence BESM takes $\Theta(n^2)$ complexity.

10. CGM

The CGM operates on offspring i' and j'. The steps are given below [49]:

a. Select any two conflicting genes $i'_{f'} = i'_{g'}$ and $j'_{k'} = j'_{l'}$ in i' and j'.
b. Update i' and j' as $(i_{1'}, i_{2'}, i_{3'} \ldots i_{f'}-1 \ldots i_{n'})$ and $(j_{1'}, j_{2'}, j_{3'} \ldots j_{k'}-1 \ldots j_{n'})$ such that $1 \leq i_{f'}-1$ and $1 \leq j_{k'}-1$.
c. Update i'' and j'' as $(i_{1''}, i_{2''}, i_{3''} \ldots i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''} \ldots j_{n''})$ by repeating steps (a)-(b) for fixed iterations.

Analysis: Steps (a) and (c) require $\Theta(n^2)$ complexity. Step (b) requires a constant running time. Hence CGM mutation takes $\Theta(n^2)$ complexity.

11. CGM-CGR Mutation

The CGM-CGR mutation is applied to the offspring i' and j' and its steps are given below [50]:

a. Perform CGM steps (a)–(b).
b. If the new genes $i_{f'}$ and $j_{k'}$ are conflicting then update $i_{f'} = i_{f'}-1$ and $j_{k'} = j_{k'}-1$ such that $1 \leq i_{f'}-1$ and $1 \leq j_{k'}-1$.
c. Update i'' and j'' as $(i_{1''}, i_{2''}, i_{3''} \ldots i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''} \ldots j_{n''})$ by repeating steps (a)–(b) for fixed iterations.

Analysis: Steps (a), (b) and (c) require $\Theta(n^2)$ complexity. Hence CGM-CGR mutation takes $\Theta(n^2)$ complexity.

12. ECGM

The ECGM operation is defined as follows [2]:

a. Perform CGM steps (a) and (b).
b. If new genes $i'_{f'}$ and $j'_{k'}$ are conflicting, then perform CGM at f' and k'.
c. Update i'' and j'' as $(i_{1''}, i_{2''}, i_{3''} \ldots i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''} \ldots j_{n''})$ by repeating steps (a) and (b) for fixed iterations.

Analysis: Steps (a), (b) and (c) require $\Theta(n^2)$ complexity. Hence ECGM takes $\Theta(n^2)$ complexity.

13. MCGM

The MCGM operation is defined as follows [2]:

a. Find the conflict edges count to every vertex in i' and j'.
b. Determine the conflict vertices in i' and j'.
c. Select arbitrarily two conflicting vertices f' in i' and k' in j'.
d. Update i' and j' as $(i_1', i_2', i_3' \ldots i_f'-1 \ldots i_n')$ and $(j_1', j_2', j_3' \ldots j_k'-1 \ldots j_n')$ such that $1 \leq i_f'-1$ and $1 \leq j_k'-1$.
e. If new genes $i'_{f'}$ and $j'_{k'}$ are conflicting then perform CGM at f' and k'.
f. Update i'' and j'' as $(i_{1''}, i_{2''}, i_{3''} \ldots i_{n''})$ and $(j_{1''}, j_{2''}, j_{3''} \ldots j_{n''})$ by repeating steps (d) and (e) for fixed iterations.

Analysis: Steps (a), (b), (e) and (f) require $\Theta(n^2)$ complexity. Steps (c) and (d) take $\Theta(c)$ complexity. Hence MCGM takes $\Theta(n^2)$ complexity.

Table 2. Asymptotic Complexity of Well-known and Recent Mutation Operators.

SNo	Mutation Operator	Asymptotic Complexity
1	Random Mutation	$\Theta(n)$
2	Polynomial Mutation	$\Theta(c)$
3	Problem-Specific Mutation 1	$\Theta(n^2)$
4	Problem-Specific Mutation 2	$\Theta(n^2)$
5	Problem-Specific Mutation 3	$\Theta(n^2)$
6	OM	$\Theta(n)$
7	BM	$\Theta(n)$
8	CSM	$\Theta(n)$
9	BESM	$\Theta(n^2)$
10	CGM	$\Theta(n^2)$
11	CGM-CGR	$\Theta(n^2)$
12	ECGM	$\Theta(n^2)$
13	MCGM	$\Theta(n^2)$

6. Global Convergence of Evolutionary Algorithms and Stochastic Convergence of Recent Genetic Operators

The necessary and sufficient condition for the global convergence of evolutionary algorithms is proved in this section [61,62]. The stochastic convergence of some recent genetic operators for solving GCP is also proved in the present section [2].

6.1. Markov Chain Model of Evolutionary Algorithms

Consider the following optimization problem:

Optimize $\{f(x); x \in S\}$ where S is a measurable space, $f(x)$ is the objective or fitness function where the absolute value of $(f(x))$ is $< +\infty$.

Represent the optimal solution set by $S^* = \{x^*$ such that $f(x^*) =$ optimize $f(x), x \in S\}$.

Let $\mu(S)$ be a measure to space S with $\mu(S^*) > 0$.

Consider the set $S^*_\delta = \{x$ such that $f(x^*) - f(x) < \delta\}$ where $\delta > 0$ is a small number.

Choose an appropriate δ to satisfy $\mu(S^*_\delta) > 0$.

The space S is called an individual space where each state s in S is called an individual.

N represents population size.

$X = S^N$ is the population space.

Denote the elements in x as $x = \{x_1, x_2, x_3 \dots x_N\}$.

Denote the optimal solution set $X^* = \{x^*$ such that $\exists\, x_i \in x$: $x_i \in S^*\}$.

An evolutionary algorithm to solve this optimization problem is formulated as follows:

a. Initialization: Generate initial population P_0 at generation $g = 0$.
b. Crossover operation: Update the new population P_t by a crossover operator $P_C(g)$.
c. Mutation operation: Update the new population P_t by a mutation operator $P_M(g)$.
d. Selection operation: Select a new population from $P_M(g)$ by a selection operator $P_S(g)$.
e. Termination: If termination conditions hold, then stop; otherwise $g = g + 1$ and update P_g.

Since the state of P_{g+1} is only dependent on the state of P_g, then $\{P_g$: $g = 0, 1, 2 \dots\}$ can be modeled by a Markov chain with the following transition function [61,62]:

$$P(g;\, x,\, dy) \;=\; P(P_g{+}1 = dy \text{ such that } P_g = x) = \int_{u \in X} \int_{v \in X} P_C(g; x,\, du) P_M(g; u,\, dv) P_S(g; v,\, dy) \quad (1)$$

Then $\{P_g$: $g = 0, 1, 2 \dots\}$ converges to X^*, if for any initial population P_0,

$$\lim_{g \to +\infty} \mu_g(X^*) = 1 \text{ where } \mu_g(X^*) = \mu\left(P_g \in\; X^*\right).$$

6.2. Convergence Conditions of Evolutionary Algorithms

The necessary and sufficient condition for the convergence of evolutionary algorithms is as follows [61,62]:

Let $\{P_g$: $g = 0, 1, 2 \dots\}$ be the Markov chain given by (1). Define $\alpha(g)$ as the difference between flow from the non-optimal solution set to the optimal solution set and vice versa.

$$\alpha(g) = \int_{x \notin X^*} P(P_{g+1} \in X^* | P_g = x)\, \mu_g(dx) - \int_{x \in X^*} P(P_{g+1} \notin X^* | P_g = x)\, \mu_g(dx)$$

Then $\{P_g$: $g = 0, 1, 2 \dots\}$ converges to the optimal solution set X^* iff

$$\sum_{g=0}^{+\infty} \alpha(g) = 1 - \mu_0(X^*)$$

Proof: For any given generation g, we have

$$\mu_{g+1}(X^*) = \int_{x \notin X^*} P(P_{g+1} \in X^* | P_g = x)\, \mu_g(dx) + \int_{x \in X^*} P(P_{g+1} \notin X^* | P_g = x)\, \mu_g(dx)$$

$$\mu_{g+1}(X^*) = \mu_g(X^*) - \int_{x \in X^*} P(P_{g+1} \notin X^* | P_g = x)\, \mu_g(dx) + \int_{x \notin X^*} P(P_{g+1} \in X^* | P_g = x)\, \mu_g(dx)$$

$$\mu_{g+1}(X^*) - \mu_g(X^*) = \alpha(g)$$

$$\mu_{g+1}(X^*) - \mu_0(X^*) = \sum_{k=0}^{g} \alpha(k)$$

$$\lim_{g \to +\infty} \mu_{g+1}(X^*) = 1 \iff \sum_{k=0}^{+\infty} \alpha(k) = 1 - \mu_0(X^*)$$

The initial population P_0 is not optimal, that is, $\mu_0(X^*) = 0$. Then

$$\sum_{g=0}^{+\infty} \alpha(g) = 1$$

6.3. Stochastic Convergence of Recent Evolutionary Operators

Some of the recent genetic operators like SPCGX, SPCGEX, ESPCGEX, ECGM, MSPCGX, and MCGM modify P_g through successive stochastic transformations, which can be analyzed by the Markovian model. The subsequent value of P_{g+1} is generated stochastically, and the evolutionary algorithm converges when the reachability and monotone conditions are fulfilled [2,41,63]. Some of the recent genetic operators converge stochastically.

6.3.1. Analyzing the Reachability Condition from j'

If j and j'' be any two individuals in the finite search space S, then j'' is reachable from individual j. That is, $j' = (j_{1'}, j_{2'}, j_{3'} \ldots j_{n'})$ 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 < Probability {j'' = (*crossover* (j) & *mutation* (j'))} < 1) then the individual j'' is reachable from individual j' [2].

First, consider the effect of crossover operation. Let S be the finite search space of the problem. Consider a gene sequence $j \in S$ such that $j = (j_1, j_2, j_3 \ldots j_n)$. The proposed crossover operators (incremental operators) SPCGX and SPCGEX produce a new gene as $j_r = j_r + 1$ if *minimum color* $\geq j_r + 1$. Hence the new gene takes one of the values in (j_r, j_r+1). The CGR operator further checks the conflict at vertex r. If there is a conflict, then the new gene at vertex r is generated as $j_r = j_r + 1$ such that *minimum color* $\geq j_r + 1$. Clearly, j_r assumes the value in (j_r, $j_r + 1$, $j_r + 2$). Thus the required probability of assigning a new gene at the conflicting vertex is 1/3 for a single crossover operation. Assume that the crossover operation is performed for k times.

$$\Rightarrow \quad \text{Probability } \{j' = \textit{crossover } (j)\} = 1/3^k > 0\ (1 \leq j, j' \leq N),\ \mathrm{k} > 0 \tag{2}$$

Now consider the effect of mutation operation. The recent mutation operators set a value of the conflicting gene in $[1, f(\mathrm{G})]$. Thus the required probability of assigning a gene at the conflicting vertex is 1/f(G) for a single mutation operation. Assume that the mutation operation is performed for l times.

$$\Rightarrow \quad \text{Probability } \{j'' = \textit{mutation } (j')\} = 1/f(\mathrm{G})^l > 0,\ l > 0. \tag{3}$$

For the chosen crossover probability p_c and mutation probability p_m, the probability of generating new gene sequence j'' is computed using the multiplication theorem of probability. The crossover and mutation operations are mutually independent. Hence the required probability of generating j'' is computed as follows:

Probability of $\{j'' = (crossover\ (j)\ \&\ mutation\ (j'))\} \geq$

$$\text{(Crossover probability)} \times \text{(Probability of generating } j' \text{ from } j) \times \text{(Mutation probability)} \times \text{(Probability of generating } j'' \text{ from } j')$$

That is

$$\text{Probability of } \{j'' = (\text{crossover } (j) \ \&\ \text{mutation } (j'))\} \geq p_c.\ \text{Probability } \{j' = \text{Crossover } (j)\}.p_m.\ \text{Probability } \{j'' = \text{Mutation } (j')\} \tag{4}$$

$$\text{Probability of } (j'' = (crossover(j)\ \&\ mutation(j'))) \} \ \geq\ p_c\, p_m \frac{1}{3^k} \frac{1}{(f(G))^l} > 0$$

Hence the gene sequence j'' is reachable from j'.

6.3.2. Analyzing whether P_g is Monotone

During the iterations of an evolutionary algorithm, P_g is modified by successive probabilistic transformations. The proportionate fitness selection selects one worst and two better gene sequences in every generation as follows:

a. Evaluate $F_g(a)$ for each gene sequence a $(1 \leq a \leq N)$ in P_g.
b. Determine $p(a)$ using

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

c. Compute $E(a) = N\ p(a)$ for all a $(1 \leq a \leq N)$.
d. Select the better gene sequences i and j and the worst gene sequence w.

The probability of selection of i and j is always greater than zero. P_g is then updated by performing the elitism operation.

The recombination and mutation operations are performed based on the values of p_{cr} and p_{mr}. For small graphs, some of the recent operators monotonically reduce F_g (i) and F_g (j) during the modification of population by a finite number of probabilistic transformations [2]:

1. The values of F_g (i) and F_g (j) monotonically decreases in subsequent generations and converge to a better near-optimal solution.

   That is, either $F_0(i) \geq F_1(i) \geq F_2(i) \geq \ldots \geq (F_q(i) = 0)$ or $F_0(j) \geq F_1(j) \geq F_2(j) \geq \ldots \geq (F_q(j) = 0)$.

2. F_g (i) and F_g (j) monotonically decrease for a finite number of generations (say t).

i.e., $F_0(i) \geq F_1(i) \geq F_2(i) \geq \ldots \geq F_t(i)$ and $F_0(j) \geq F_1(j) \geq F_2(j) \geq \ldots \geq F_t(j)$.

Then, based on the values of p_{cr} and p_{mr}, the fitness-function values again increase for a finite number of generations (say s).

i.e., $F_t(i) \leq F_{t+1}(i) \leq \ldots \leq F_s(i)$ and $F_t(j) \leq F_{t+1}(j) \leq \ldots \leq F_s(j)$.

For complex graphs, the successive probabilistic transformations find a better near-optimal solution after a finite number of generations. Hence, the successive probabilistic changes utilize P_g as a monotone in some of the subsequent generations to achieve a better near-optimal solution. These successive probabilistic changes, applying for over a large number of generations, for example, 1000 runs each with 5,000,000 generations of the genetic algorithm, achieve a better near-optimal solution [2].

7. Conclusions and Future Work

GCP is an *NP*-hard combinatorial optimization problem and can be applied to innumerable engineering problems. $\chi(G)$ can be obtained using different approximations and evolutionary methods. In comparison to other methods, genetic algorithms are useful in solving multi-objective optimization problems with vast search space [2]. Hence, it is expected to design new evolutionary operators to achieve faster stochastic convergence with a minimal number of generations.

This paper explored the asymptotic analysis of some well-known and recent evolutionary operators for finding chromatic numbers. The asymptotic analysis of different crossover and mutation operators helps in minimizing problem search space and computational complexity. The choice of the right genetic operators facilitates an evolutionary algorithm to achieve faster convergence with lesser population size (N) through an effective distribution of promising genes. The selection of evolutionary operators also plays an essential role in reducing the bounds for *minimum color* obtained for the benchmark graphs [2]. The necessary and sufficient condition for global convergence of the evolutionary algorithm and the stochastic convergence of some recent evolutionary operators for solving GCP are analyzed in this paper.

Our current work enlightens some new research directions [64–66].

i. The recent genetic operators can be combined with better local search strategies in order to further reduce the computational complexity of GCP.
ii. The self-adaptive evolutionary operators with heuristics can be designed to find chromatic numbers as well as can be applied to channel allocation problem (CAP) and scheduling problems. CAP is an extension of GCP, which assigns channels to mobile stations.
iii. The performances of some recent operators can be evaluated under different parameter values, such as graph density, N, p_c, and p_m.
iv. Some of the recent operators can be combined to reduce $\overline{g}$.
v. The stochastic convergence of genetic algorithms can be extended to multi-objective optimization problems, and the necessary conditions for the global convergence of multi-objective optimization problems can be analyzed mathematically.
vi. An algebraic framework that may lead to a unification of different evolutionary operators for combinatorial problems can be designed and analyzed [64–66].

Author Contributions: Methodology, Software, Writing-original draft and formal analysis, R.M.; formal analysis, writing-review and editing, G.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research received no external funding.

Acknowledgments: The authors would like to acknowledge the support rendered by the management of SASTRA Deemed 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 [2]. The authors would like to thank Gary Lewandowski and Michael Trick for uploading the graph repository in W3C [2,60].

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Balakrishnan, R.; Ranganathan, K. *A Textbook of Graph Theory*, 1st ed.; Springer-Verlag Publisher: New York, NY, USA, 2000.
2. Marappan, R.; Sethumadhavan, G. Solution to Graph Coloring using Genetic and Tabu Search Procedures. *Arab. J. Sci. Eng.* **2018**, *43*, 525–542. [CrossRef]
3. Garey, M.R.; Johnson, D.S. *Computers and Intractability: A Guide to the Theory of NP-Completeness*; Freeman and Company: San Francisco, CA, USA, 1979.
4. Maitra, T.; Pal, A.J.; Bhattacharyya, D.; Kim, T.H. Noise Reduction in VLSI Circuits using Modified GA Based Graph Coloring. *Int. J. Control Autom.* **2010**, *3*, 37–44.
5. Yoshino, J.; Ohtomo, I. Study on efficient channel assignment method using the genetic algorithm for mobile communication systems. *Soft Comput.* **2005**, *9*, 143–148. [CrossRef]

6. Chen, X.; Zu, Y.; Xu, J.; Wang, Z.; Yao, B. Vertex-Distinguishing E-Total Colorings of Graphs. *Arab. J. Sci. Eng.* **2011**, *36*, 1485–1500. [CrossRef]
7. Abdelfattah, M.; Shawish, A. Automated Academic Schedule Builder for University's Faculties. In Proceedings of the World Congress on Engineering 2016, London, UK, 29 June–1 July 2016.
8. Saharan, S.; Kumar, R. Graph Coloring based Optimized Algorithm for Resource Utilization in Examination Scheduling. *Appl. Math. Inf. Sci.* **2016**, *10*, 1193–1201. [CrossRef]
9. Tawfiq, F.M.O.; Al-qahtani, K.K.S. Graph Coloring Applied to Medical Doctors Schedule. In Proceedings of the 10th International Conference on Advanced Engineering Computing and Applications in Sciences, Venice, Italy, 9–13 October 2016.
10. Thevenin, S.; Zufferey, N.; Potvin, J.Y. Graph multi-coloring for a job scheduling application. *CIRRELT* 2016. [CrossRef]
11. Mehrotra, A.; Trick, M.A. A Column Generation Approach for Graph Coloring. *Inf. J. Comput.* **1995**, *8*, 344–354. [CrossRef]
12. Szép, T.; Mann, Z.Á. Graph coloring: The more colors, the better? In Proceedings of the 11th International Symposium on Computational Intelligence and Informatics (CINTI), Budapest, Hungary, 18–20 November 2010.
13. Galinier, P.; Hertz, A. A survey of local search methods for graph coloring. *Comput. Oper. Res.* **2006**, *33*, 2547–2562. [CrossRef]
14. Johnson, D.S.; Aragon, C.R.; McGeoch, L.A.; Schevon, C. An Experimental Evaluation; Part II, Graph Coloring and Number Partitioning. *Oper. Res.* **1991**, *39*, 378–406. [CrossRef]
15. Johnson, D.S.; Trick, M.A. *Cliques, Coloring, and Satisfiability*; American Mathematical Society: Providence, RI, USA, 1993; Volume 26.
16. Mizuno, K.; Nishihara, S. Constructive generation of very hard 3-colorability instances. *Discret. Appl. Math.* **2008**, *156*, 218–229. [CrossRef]
17. Hertz, A.; de Werra, D. Using tabu search techniques for graph coloring. *Computing* **1987**, *39*, 345–351. [CrossRef]
18. Monasson, R. On the Analysis of Backtrack Procedures for the Coloring of Random Graphs. *Lect. Notes Phys.* **2004**, *650*, 235–254.
19. Eiben, A.E.; Van Der Hauw, J.K.; Van Hemert, J.I. Graph Coloring with Adaptive Evolutionary Algorithms. *J. Heuristics* **1998**, *4*, 25–46. [CrossRef]
20. Kumar, R.; Tolay, P.; Tiwary, S. Enhancing solution quality of the biobjective graph coloring problem using hybridization of EA: Biobjective graph coloring problem. In Proceedings of the Genetic and Evolutionary Computation Conference (GECCO), Atlanta, GA, USA, 12–16 July 2008; ACM Press: New York, NY, USA, 2008; pp. 547–554.
21. Deb, K. *Multi-Objective Optimization Using Evolutionary Algorithms*; John Wiley & Sons: Hoboken, NJ, USA, 2001.
22. Méndez-Díaz, I.; Zabala, P. A Branch and Cut algorithm for graph coloring. *Discret. Appl. Math.* **2006**, *154*, 826–847. [CrossRef]
23. Cases, B.; Hernandez, C.; Grana, M.; D'Anjou, A. On the ability of Swarms to compute the 3-coloring of graphs. In *Artificial Life*; MIT Press: Cambridge, MA, USA, 2008.
24. Graña, M.; Cases, B.; Hernandez, C.; D'Anjou, A. Further Results on Swarms Solving Graph Coloring. In Proceedings of the ICCSA 2010: Computational Science and Its Applications, Fukuoka, Japan, 23–26 March 2010; pp. 541–551.
25. 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.* **2011**, *38*, 5525–5531. [CrossRef]
26. Cui, G.; Qin, L.; Liu, S.; Wang, Y.; Zhang, X.; Cao, X. Modified PSO algorithm for solving planar graph coloring problem. *Prog. Nat. Sci.* **2008**, *18*, 353–357. [CrossRef]
27. Bui, T.N.; Nguyen, T.H.; Patel, C.M.; Phan, K.A.T. An ant-based algorithm for coloring graphs. *Discret. Appl. Math.* **2008**, *156*, 190–200. [CrossRef]
28. Zhou, Y.; Zheng, H.; Luo, Q.; Wu, J. An improved Cuckoo Search Algorithm for Solving Planar Graph Coloring Problem. *Appl. Math. Inf. Sci.* **2013**, *7*, 785–792. [CrossRef]
29. Prestwich, S. Generalised graph colouring by a hybrid of local search and constraint programming. *Discret. Appl. Math.* **2008**, *156*, 148–158. [CrossRef]

30. Demange, M.; Ekim, T.; Ries, B.; Tanasescu, C. On some applications of the selective graph coloring problem. *Eur. J. Oper. Res.* **2015**, *240*, 307–314. [CrossRef]
31. Demange, M.; Ekim, T.; Ries, B. On the minimum and maximum selective graph coloring problems in some graph classes. *Discret. Appl. Math.* **2016**, *204*, 77–89. [CrossRef]
32. Takeshita, L. Coloring Signed Graphs. 2016, pp. 1–7. Available online: https://math.mit.edu/~{}apost/courses/18.204-2016/18.204_Lynn_Takeshita_final_paper.pdf (accessed on 20 January 2020).
33. Macajov, E.; Raspaud, A.; Skoviera, M. The Chromatic Number of a Signed Graph. Cornell University Library. *arXiv* **2016**, arXiv:1412.6349.
34. Zhou, B. On the Maximum Number of Dominating Classes in Graph Coloring. *Open J. Discret. Math.* **2016**, *6*, 70–73. [CrossRef]
35. Gaspers, S.; Lee, E.J. Faster Graph Coloring in Polynomial Space. Cornell University Library. *arXiv* **2016**, arXiv:1607.06201.
36. 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.* **2017**, *21*, 313–340. [CrossRef]
37. Galán, S.F. Simple decentralized graph coloring. *Comput. Optim. Appl.* **2017**, *66*, 163–185. [CrossRef]
38. Aslan, M.; Baykan, N.A. A Performance Comparison of Graph Coloring Algorithms. *Int. J. Intell. Syst. Appl. Eng.* **2016**, 1–7. [CrossRef]
39. Fleurent, C.; Jacques, A. Ferland: Genetic and hybrid algorithms for graph coloring. *Ann. Oper. Res.* **1996**, *63*, 437–461. [CrossRef]
40. Mumford, C.L. New Order Based Crossovers for the Graph Coloring Problem. In *Parallel Problem Solving from Nature*; Springer: Berlin/Heidelberg, Germany, 2006; Volume 4193, pp. 880–889.
41. Han, L.; Han, Z. A Novel Bi-objective Genetic Algorithm for the Graph Coloring Problem. In Proceedings of the 2nd International Conference on Computer Modeling and Simulation, Sanya, China, 22–24 January 2010.
42. Hajduk, J.O. An Analysis of Tabu Search for the Graph Coloring Problem. Master's Thesis, Utrecht University, Utrecht, The Netherlands, 2010. Available online: www.cs.uu.nl/education/scripties/pdf.php?SID=INF/SCR-2009-095 (accessed on 20 January 2020).
43. Costa, D.; Hertz, A.; Dubuis, C. Embedding a sequential procedure within an evolutionary algorithm for coloring problems in graphs. *J. Heuristics* **1995**, *1*, 105–128. [CrossRef]
44. Dorne, R.; Hao, J.K. A new genetic local search algorithm for graph coloring. In Proceedings of the International Conference on Parallel Problem Solving from Nature, Amsterdam, The Netherlands, 27–30 September 1998; Springer: Berlin/Heidelberg, Germany, 1998; pp. 745–754.
45. Galinier, P.; Hao, J.K. Hybrid Evolutionary Algorithms for Graph Coloring. *J. Comb. Optim.* **1999**, *3*, 379–397. [CrossRef]
46. Lu, Z.; Hao, J.K. A memetic algorithm for graph coloring. *Eur. J. Oper. Res.* **2010**, *203*, 241–250. [CrossRef]
47. Porumbel, D.C.; Hao, J.K.; Kuntz, P. An evolutionary approach with diversity guarantee and well-informed grouping recombination for graph coloring. *Comput. Oper. Res.* **2010**, *37*, 1822–1832. [CrossRef]
48. 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.* **2013**, *13*, 2812–2822. [CrossRef]
49. Marappan, Raja.; Sethumadhavan, Gopalakrishnan. A new genetic algorithm for graph coloring. In Proceedings of the 5th International Conference on Computational Intelligence, Modelling and Simulation, Seoul, Korea, 24–26 September 2013; pp. 49–54.
50. Sethumadhavan, G.; Marappan, R. A Genetic Algorithm for Graph Coloring using Single Parent Conflict Gene Crossover and Mutation with Conflict Gene Removal Procedure. In Proceedings of the IEEE International Conference on Computational Intelligence and Computing Research 2013, Madurai, India, 26–28 December 2013; pp. 350–355.
51. Ryan, E.; Azad, R.M.A.; Ryan, C. On the performance of genetic operators and the random key representation. In Proceedings of the European Conference on Genetic Programming, Coimbra, Portugal, 5–7 April 2004; Springer: Berlin/Heidelberg, Germany, 2004; pp. 162–173.
52. Croitoru, C.; Luchian, H.; Gheorghie, O.; Apetrei, A. A New Genetic Graph Coloring Heuristic. 2002. Available online: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.14.19612002 (accessed on 20 January 2020).

53. Lewis, R.M.R. *A Guide to Graph Coloring, Algorithms and Applications*; Springer: Berlin/Heidelberg, Germany, 2016.
54. Mann, Z.Á. Complexity of Coloring Random Graphs: An Experimental Study of the Hardest Region. *J. Exp. Algorithmics* **2018**, *23*, 1–19. [CrossRef]
55. Mostafaie, T.; Khiyabani, F.M.; Navimipour, N.J. A systematic study on meta-heuristic approaches for solving the graph coloring problem. *Comput. Oper. Res.* **2019**. [CrossRef]
56. Srivastava, S.; Sundararaghavan, V. Graph Coloring Approach to Mesh Generation in Multiphase Media with Smooth Boundaries. *AIAA J.* **2020**, *58*, 198–205. [CrossRef]
57. Dukanovic, I.; Rendl, F. A semidefinite programming-based heuristic for graph coloring. *Discret. Appl. Math.* **2008**, *156*, 180–189. [CrossRef]
58. Hernández, H.; FrogSim, B.C. Distributed graph coloring in wireless ad hoc networks. *Telecommun. Syst.* **2014**, *55*, 211–223. [CrossRef]
59. Segundo, P.S. A new DSATUR-based algorithm for exact vertex coloring. *Comput. Oper. Res.* **2012**, *39*, 1724–1733. [CrossRef]
60. The Graph Coloring Instances. Available online: http://mat.gsia.cmu.edu/COLOR/instances.html (accessed on 20 January 2020).
61. Back, T. *Evolutionary Algorithms in Theory and Practice*; Oxford University Press: New York, NY, USA, 1996; pp. 21–28.
62. He, J.; Yu, X. Conditions for the Convergence of Evolutionary Algorithms. *J. Syst. Archit.* **2001**, *47*, 601–612. [CrossRef]
63. Rudolph, G. Finite Markov Chain Results in Evolutionary Computation: A Tour d'Horizon. *Fundam. Inform.* **1998**, *35*, 67–89. [CrossRef]
64. Marco, B.; Alfredo, M. Automatic algebraic evolutionary algorithms. In *Italian Workshop on Artificial Life and Evolutionary Computation*; Springer: Berlin/Heidelberg, Germany, 2017; pp. 271–283.
65. Marco, B.; Alfredo, M. Learning bayesian networks with algebraic differential evolution. In *International Conference on Parallel Problem Solving from Nature*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 436–448.
66. Marco, B.; Alfredo, M. MOEA/DEP: An algebraic decomposition-based evolutionary algorithm for the multiobjective permutation flow shop scheduling problem. In *European Conference on Evolutionary Computation in Combinatorial Optimization*; Springer: Berlin/Heidelberg, Germany, 2018; pp. 132–145.