MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

SOLVING GRAPH COLORING PROBLEM BY USING AN EVOLUTIONARY ALGORITHM

SERAP KORKMAZ

MASTER THESIS

Department of Computer Engineering

Thesis Supervisor
Assist. Prof. Betül DEMİRÖZ BOZ

ISTANBUL, 2020

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES
IN PURE AND APPLIED SCIENCES

SOLVING GRAPH COLORING PROBLEM BY USING AN EVOLUTIONARY ALGORITHM

SERAP KORKMAZ

(524114001)

MASTER THESIS
Department of Computer Engineering

Thesis Supervisor
Assist. Prof. Betül DEMİRÖZ BOZ

ISTANBUL, 2020

MARMARA UNIVERSITY
INSTITUTE FOR GRADUATE STUDIES IN PURE AND APPLIED SCIENCES

Serap KORKMAZ, a Master of Science student of Marmara University Institute for Graduate Studies in Pure and Applied Sciences, defended her thesis entitled "**Solving Graph Coloring Problem by Using An Evolutionary Algorithm**", on Jan 27, 2020 and has been found to be satisfactory by the jury members.

Jury Members

Assist.Prof. Betül BOZ (Advisor)
Marmara University .....(SIGN).....

Assist.Prof. Fatma CORUT ERGİN (Jury Member)
Marmara Üniversitesi .....(SIGN).....

Assist.Prof. Berna KİRAZ (Jury Member)
Fatih Sultan Mehmet Üniversitesi .....(SIGN).....

APPROVAL

Marmara University Institute for Graduate Studies in Pure and Applied Sciences Executive Committee approves that Serap KORKMAZ be granted the degree of Master of Science in Department of Computer Engineering, Computer Engineering Program on 12.02.2020 (Resolution no: 2020/06-07).

Director of the Institute

Prof. Dr. Bülent EKİCİ
Enstitü Müdürü

TABLE OF CONTENTS

ACKNOWLEDGMENT i

ÖZET ii

ABSTRACT iii

SYMBOLS iv

ABBREVIATIONS v

LIST OF FIGURES vi

LIST OF TABLES viii

1. INTRODUCTION 1

2. RELATED WORK 4

3. PROPOSED WORK 6

3.1 Initial Population 8

3.1.1 Sample scenario for creation of an initial solution 9

3.2 Crossover Operators 12

3.2.1 Weight centered crossover operator (WCCO) 12

3.2.1.1 Determination of the suitable previous color classes 13

3.2.1.2 Sample scenario for weight centered crossover operator 15

3.2.2 Conflict centered crossover operator (CCCO) 22

3.2.2.1 Sample scenario for conflict centered crossover operator 24

3.3 Mutation Operator 30

3.3.1 Sample Scenario for Mutation Operator 31

4. EXPERIMENTAL STUDY 35

4.1 Experiments for Overall Algorithm 35

4.2 Experiments for Mutation Operator 38

4.3 Experiments for Crossover Operators 40

5. CONCLUSION 44

6. FUTURE WORK 45

REFERENCES 46

ACKNOWLEDGMENT

I would like to express my sincere thanks to my advisor Assist. Prof. Dr. Betül BOZ for her guidance and support during my thesis study.

I want to thank my thesis jury members for evaluating my thesis study.

ÖZET

SOLVING GRAPH COLORING PROBLEM BY USING AN EVOLUTIONARY ALGORITHM

Grafik renklendirme problemi düğümler ve kenarlardan oluşan yapılar olan grafiklerin renklendirilmesi problemidir. Renklendirme işlemi düğümlerin renklendirilmesi veya kenarların renklendirilmesi gibi farklı şekillerde yapılabilmektedir. Grafik renklendirme problemi problemin türüne göre düğüm sayıları, düğümler arasındaki kenar sayıları ve kenarların düğümlerle bağlanma şekillleri, düğümlerin ağırlığının olup olmaması vs. gibi birçok etkene sahiptir ve bu etkenlerin çeşitliliğine göre problemin karmaşıklığı da değişmektedir. Grafik renklendirme probleminin düğüm renklendirilmesi, kenar renklendirilmesi gibi alt grupları vardır. Düğüm renklendirmesinin bir türü olan "ağırlıklı düğüm renklendirme problemi" bu çalışmada ele alınmaktadır. Bu problemde düğümlerin her birinin ağırlık değeri vardır ve aralarında kenar bulunan düğümler farklı renklerle renklendirilir. Her renk grubundaki en ağır düğüm ilgili renk grubunun temsilci düğümü olarak belirlenir ve temsilci düğümlerin ağırlıkları toplamı grafik renklendirilirken minimize edilmeye çalışılır.

Ağırlıklı düğüm renklendirme probleminin çözülmesi gerçek hayat uygulamalarının bir kısmının çözümünde kullanılabilmesi açısından önem taşır. Buna ek olarak problemin polinom zamanda bilinen bir çözümünün olmaması problemle ilgili üretilen çözüm yöntemlerini değerli kılar. Bu çalışmada ağırlıklı düğüm renklendirme probleminin çözümünde kullanmak amacıyla geliştirdiğimiz algoritmayı sunmaktayız. Geliştirdiğimiz algoritma birçok problemde yaygın olarak kullanılan ve doğadan esinlenen genetik algoritma tabanlı bir algoritmadır. Doğada olduğu gibi, genetik algoritmada da bireylerden yeni nesiller oluşur. Bireylerden yeni nesiller oluşurken crossover (çaprazlama) ve mutasyon ile genetik aktarım ve çeşitlilik sağlanır. Çalışmamızda farklı özelliklere öncelik veren ancak birlikte çalışan 2 crossover tekniğini ve bu tekniklerin uygulanması neticesinde elde edilen çözümlere uygulanan bir mutasyon tekniğini önermekteyiz. Önermiş olduğumuz teknikleri literatürde yer alan 3 çalışma ile karşılaştırmış bulunmaktayız. Karşılaştırmada kullanılan materyallerle ilgili bilgileri, karşılaştırma tekniğimizi, elde edilen sonuçları ve bu sonuçların karşılaştırmalarına çalışmamızın ilgili bölümlerinde yer vermekteyiz.

ABSTRACT

SOLVING GRAPH COLORING PROBLEM BY USING AN EVOLUTIONARY ALGORITHM

Graph coloring problem is the problem of coloring graphs which are structures consisting of vertices and edges. Coloring can be done in different ways such as, edge coloring or vertex coloring. Graph coloring problem includes many factors such as number of vertices, number of edges between vertices, how these edges are connected with the vertices and whether vertices have weight values or not depending on type of the problem. Consequently, degree of complexity of the problem changes according to variety of the factors. Graph coloring problem contains subcategories such as edge coloring and vertex coloring. In this study, Weighted Vertex Coloring Problem which is a type of vertex coloring is studied. In weighted vertex coloring problem, each vertex has a weight value and vertices connected with an edge are colored with different colors. Heaviest vertex in each color class becomes the representative vertex of that color class and sum of weights of the representative vertices is tried to be minimized while coloring the graph.

Solving the weighted vertex coloring problem is important because of being used in solving some real life applications. In addition, being an NP-hard problem makes solution methods offered for this problem important. In this study, we offer an algorithm to solve the weighted vertex coloring problem. Our proposed algorithm is based on Genetic Algorithm which is a nature-inspired algorithm and widely used in the literature to solve problems. In Genetic Algorithm, individuals form offspring as in nature. While forming offspring, crossover and mutation operators provide transfer of genes to offspring and variety. In our study, we propose 2 crossover operators working together but giving priority to different features and a mutation operator which is applied on solutions obtained by application of the proposed crossover operators. The proposed techniques are compared with 3 other techniques in the literature. Information about materials used in the comparison, comparison techniques, obtained results and comparison of the results are provided related sections in our study.

SYMBOLS

G **:** Graph

E **:** Edges

V **:** Vertices

w **:** Weight of a vertex

ABBREVIATIONS

AFISA **:** Adaptive Feasible and Infeasible Search Algorithm

CCCO **:** Conflict Centered Crossover Operator

EA **:** Evolutionary Algorithms

ES **:** Evolution Strategies

GA **:** Genetic Algorithm

GCP **:** Graph Coloring Problem

ILP **:** Integer Linear Programming

MWSS **:** Maximum Weight Stable Set

SCP **:** Set Covering Problem

SR **:** Success Rate

VCP **:** Vertex Coloring Problem

WCCO **:** Weighted Centered Crossover Operator

WVCP **:** Weighted Vertex Coloring Problem

LIST OF FIGURES

Figure 1. 1 Steps of Basic Genetic Algorithm 2
Figure 3. 1 General schema of the proposed algorithm 7
Figure 3. 2 Representation of a sample initial solution 8
Figure 3. 3 General schema of creation of initial population 9
Figure 3. 4 Sample Graph 10
Figure 3. 5 Construction steps of the sample initial solution 11
Figure 3. 6 Coloring the graph according to the initial solution obtained in Figure 3.5 11
Figure 3. 7 General schema of Weight Centered Crossover Operator (WCCO) 14
Figure 3. 8 Determination of Previous Color Classes for WCCO 15
Figure 3. 9 Randomly Selected Parents 16
Figure 3. 10 Construction of set of vertices for the first color class 16
Figure 3. 11 Parents After Construction of the set of vertices 16
Figure 3. 12 Weight Values and Number of Edges of the Vertices in the Set of Vertices to be Placed 17
Figure 3. 13 Illustration of Construction of the First Color Class 17
Figure 3. 14 Parents After Application of Fifty Percentage Rule 18
Figure 3. 15 Set of Vertices to be Placed Created According to Fifty Percentage Rule 18
Figure 3. 16 Weight Values and Number of Edges of the Vertices in the Set of Vertices to be Placed 18
Figure 3. 17 Illustration of construction of the second color class 19
Figure 3. 18 Illustration of Construction of the Third Color Class 20
Figure 3. 19 Summarized construction of offspring by applying WCCO 21
Figure 3. 20 Conflict Centered Crossover Operator 23
Figure 3. 21 Parents After Construction of Set of Vertices to be Placed 24
Figure 3. 22 Construction of Set of Vertices for the First Color Class 24
Figure 3. 23 Properties of the vertices **(a)** vertices and list of vertices conflicting with them in the set of vertices to be placed. **(b)** Updated version of Figure 3.23(a) after placement of vertex with number 3 in the pool. **(c)** Updated version of Figure 3.23(b) after placement in the pool of vertex with number 13. **(d)** Updated version of Figure 3.23(c) after placement of vertex with number 4 in the pool. 25
Figure 3. 24 Properties of the remaining vertices **(a)** updated version of Figure 3.23 (d) after placement of vertex with number 6 in the pool. **(b)** updated version of Figure 3.24 (a) after placement of conflict free vertex with number 1. 26
Figure 3. 25 Illustration of Construction of the First Color Class (Conflict Centered) . 26
Figure 3. 26 Parents after construction of set of vertices to be placed (for 2nd color class) 26
Figure 3. 27 Set of vertices to be placed(2nd color class) 27
Figure 3. 28 Illustration of construction of the second color class (Conflict Centered) 27
Figure 3. 29 Parents after construction of set of vertices to be placed 28
Figure 3. 30 Set of vertices to be placed(3rd color class) 28
Figure 3. 31 Weight values and list of conflicting vertices in the set of vertices to be placed 28
Figure 3. 32 Illustration of construction of the third color class 28
Figure 3. 33 Summarized construction of offspring by using CCCO 29
Figure 3. 34 Offspring after application of mutation operator by selecting the third color class 32

Figure 3. 35 Final Structure of the Offspring 33
Figure 3. 36 Summarized construction of the sample mutation operator..... 34
Figure 4. 1 Experiment on Conflict Centered Crossover Operator 42
Figure 4. 2 Experiment on Weight Centered Crossover Operator..... 42
Figure 4. 3 Experiment on Crossover Operators Working Together..... 42

LIST OF TABLES

Table 1 Benchmark Instances 36
Table 2 Comparison of Mutation Operators 39
Table 3 Updated Version of Table 1 41

1. INTRODUCTION

Graph Coloring Problem (GCP) is the problem of coloring graphs ($G = (V, E)$) which are structures consisting of vertices V ($V = \{v_1, v_2, \ldots, v_n\}$ where n is equal to number of vertices) and edges E ($E \subset V \times V$). Coloring can be done in different ways such as, edge coloring and vertex coloring [1, 32]. GCP includes many factors such as number of vertices, number of edges between vertices, how these edges are connected with the vertices and whether vertices have weight values or not depending on type of the problem. Consequently, degree of complexity of the problem changes according to variety of the factors. In this study, Weighted Vertex Coloring Problem (WVCP) which is a type of Vertex Coloring Problem (VCP) is studied. In VCP, all vertices of a graph required to be colored. If there is an edge between any two vertices, these vertices are called conflicting vertices and they are colored with different colors. Vertices colored with the same color construct a color class and combination of color classes forms a candidate solution of the problem. In WVCP, in addition to VCP, each vertex of the undirected graph has a positive weight value (w). Heaviest vertex in each color class becomes the representative vertex of that color class and cost of a color class is equal to weight of the representative vertex of that color class. Fitness value of a solution is sum of costs of its color classes. In WVCP, sum of costs of color classes is tried to be minimized. Solving the WVCP is important because of being used in solving some real life applications. In addition, being an NP-hard [2, 3] problem makes solution methods offered for this problem important. Many methods are offered to solve GCP such as exact algorithms and greedy algorithms. In this study, we proposed an Evolutionary Algorithm (EA) based method to solve WVCP.

EA are algorithms whose working mechanisms are based on natural evolution [4, 5, 6] and they are widely used in the literature to solve variety number of problems [41]. EA include different subcategories such as Genetic Algorithms (GAs) [7, 8, 9], Evolution Strategies (ES) [38], Genetic Programming [39] and Evolutionary Programming [40]. Genetic Algorithm (GA) is a nature inspired algorithm and first offered by John Holland in 1975 [7]. GA includes some of the features that are important for maintaining the continuity of the species, which plays an important role in the life cycle of living things. Population, individuals who make up the population, the transfer of genes to offspring

hold an important place in GA. In GA, individuals actually represent candidate solutions, and the population actually represents the set of solutions formed by candidate solutions. In addition to creating individuals to generate a population, the steps of genetic algorithms include calculating the fitness values of individuals, application of crossover and mutation operators and selection of individuals who will continue to exist in the population. The crossover step allows individuals to pass on their genetic characteristics to their offspring, and the mutation step changes the chromosomes of the offspring and contributes to genetic diversity. Crossover and mutation steps along with replacement of individuals may result in creation of better offspring. Keeping better offspring in population may result in creation of even better individuals over next generations and may provide increased ability to survive. Steps of the basic GA is illustrated in Figure 1.1.

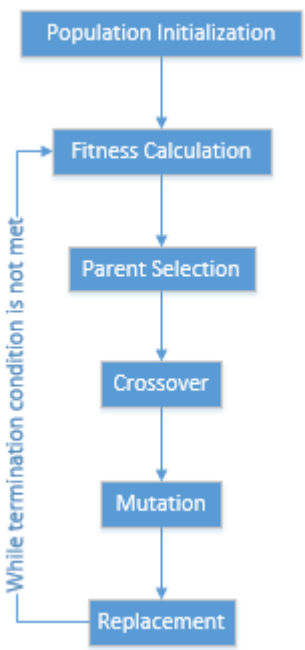


Figure 1. 1 Steps of Basic Genetic Algorithm

Our proposed algorithm is a GA based algorithm. For this purpose, we designed an algorithm which contains 2 different pool-based crossover operators [10] and a mutation operator. Both of our crossover operators use a structure called as pool to keep vertices that cannot be placed in any color class in different steps of forming an offspring. Our first crossover operator is conflict centered and the second one is weight centered. Conflict centered crossover operator forms an offspring by using sets of vertices. While

creating color classes of offspring from a set of vertices, number of conflicts of each vertex with other vertices in the same set is considered. Vertices with no conflicts are placed first in color classes. In addition, vertices having less number of conflicts are assigned to color classes before vertices having greater number of conflicts. On the other hand, in our weight centered crossover operator, weights of the vertices are considered first. While creating color classes of offspring from a set of vertices, vertices with a larger weight in the same set are tried to be placed first in color classes of offspring. Our mutation operator is applied on each offspring formed by the crossover operators. The proposed mutation operator tries to minimize the sum of costs of color classes of offspring by trying to place representative vertices of color classes with smaller costs in color classes with greater cost values in a conflict free way. We tested our proposed algorithm on 46 DIMACS instances and compared our results with the other works in the literature [11, 12, 13].

2. RELATED WORK

GCP has been worked by many researches in the literature in different application areas. One of the application areas of the GCP is register allocation problem in which variables are tried to be placed into registers [14, 15, 16, 17, 18, 19]. An example of application of EA to register allocation problem is offered in [14]. In [14], a hybrid EA is proposed which contains a special crossover operator and combines EA with local search. Another example of hybrid evolutionary algorithms for graph coloring can be found in [37]. Examples of other application areas of GCP are scheduling [20], timetabling [21], frequency assignment [22], network [31]. Exact, greedy and heuristic methods are some of the methods that are used to solve GCP. A greedy approach can be found in [20], a branch-and-price approach is offered in [23], exact approaches can be found in [35, 36] and examples of the heuristic approach can be found in [24 , 33 , 34]. Proposed algorithm DSATUR (in [24]) is a well-known algorithm offered for GCP. DSATUR, considers degree of vertices and saturation degree of vertices while coloring the graph. Saturation degree of a vertex is defined by the authors as the number of different colors to which it is adjacent (colored vertices). DSATUR, starts coloring with the vertex having the maximal degree and it considers vertices with maximal saturation degree for coloring remaining vertices in the uncolored subgraph. In addition to maximal saturation degree, maximal degree is also considered and the vertex with maximal degree is selected if there are more than one vertices having the same maximal saturation degree. This process continues until all vertices are colored. Like DSATUR for GCP, heuristic methods are also offered particularly for WVCP.

One of the recently offered heuristic methods in the literature for WVCP is Adaptive Feasible and Infeasible Search Algorithm (AFISA) [11]. AFISA searches for both feasible and infeasible solutions to solve WVCP. AFISA starts with feasible initial solution created in a greedy way and applies tabu search [25, 26, 27, 28] for search operation. Even if the algorithm starts with feasible solutions, infeasible solutions are also contained due to the search operation. Since infeasible solutions are also contained, a new evaluation function is introduced to deal with infeasible solutions. The introduced evaluation function adds a penalty function to objective function. Number of conflicts in a solution is taken into account in penalty function which can limit the boundaries of the

infeasible search space. In addition, a parameter is also introduced in the new evaluation function to increase or decrease the effect of the infeasible solutions adaptively during the search operation which is conducted using a tabu search mechanism. In AFISA, performance of the algorithm is compared with other studies [12, 13, 29] in the literature.

One of the comparison algorithms in AFISA is 2-Phase algorithm which is proposed in [13]. In [13], the authors offered three Integer Linear Programming (ILP) models for WVCP and the 2-Phase algorithm which is also a heuristic approach like AFISA. 2-Phase is constructed by using Model3 which is the third ILP model offered by the authors. 2-Phase uses Lagrangian-based heuristic algorithm instead of exactly solving Set Covering Problem (SCP) formulation in the model. 2-Phase consists of two steps and the first step is used to produce independent sets (columns). To produce independent sets, seven different methods are applied in sequence. These methods of the first step include different variations of DSATUR. Second phase starts after the first phase and it is used for column optimization. SCP columns are solved using the Lagrangian heuristic algorithm offered in [30]. The authors also offer two post optimization techniques two improve the solutions obtained from first and second phases of the algorithm but the details of the post optimization techniques are not covered here. Another reference study considered in AFISA is [12] in which WVCP is reformulated and solved on an associated graph.

3. PROPOSED WORK

Our proposed algorithm starts with the creation of the initial population which consists of individuals that have conflict free vertices in their color classes. A predefined total number of iterations determined as the termination condition of our algorithm. While termination condition is not met, one of the crossover operators is applied on population for a predefined number of iterations (stopping criteria of crossover operators). When stopping criteria is met for the selected crossover operator, termination condition of the algorithm is checked. If the termination condition is not met, the algorithm continues with the other crossover operator until the stopping criteria of the operator is met. This cycle continues until the termination condition of the algorithm is met. In our implementation, the same stopping criteria is used for both operators and the first operator applied is the conflict centered crossover operator. When the algorithm stops, the best solution found is reported.

In crossover operators, two individuals (parents) are selected from the population randomly and pool-based crossover operation is performed on the parents to form an offspring. Then, mutation operator is performed on the offspring. Fitness value of the offspring before mutation is compared with the fitness value after mutation and the offspring with smaller fitness value is selected. If the fitness value of the offspring is less than the fitness value of at least one of its parents, it replaces the parent with the greater fitness value. In each iteration of the crossover operators, all individuals of the population are used. Therefore, number of offspring formed in each iteration is equal to half of the population size. General schema of the proposed algorithm is given in Figure 3.1.

```
Input: Graph G = (V , E)
Output: The best solution found
s ← predetermined population size
P ← Ø /*P symbolizes the initial population*/
M ← Ø /*M symbolizes set of matched parents*/
i=1, j = s/2
p1 ← Ø , p2 ← Ø /*p1 and p2 are individuals*/
o1 ← Ø , o2 ← Ø  , o3 ← Ø/*o1, o2 and o3 are offspring*/
P ← Create the initial population
while termination condition is not met
   while stopping criteria is not met for crossover operator
          M ← Ø
          for i = 1,2,...,j
             p1 ← select an individual randomly from {P\M}
             M ← M ∪ p1
             p2 ← select an individual randomly from {P\M}
             M ← M ∪ p2
              /*form an offspring*/
             o1 ← apply pool-based crossover operation on p1 and p2
             o2 ← apply mutation on o1
             if fitness value of o1 is different from fitness value of o2
                o3 ← select the one with smaller fitness value
             else
                o3 ← select either o1 or o2 randomly
             end if
             /*determine the parent to be replaced*/
             if fitness values of p1 and p2 are different
               select the parent with bigger fitness value
             else
                select either p1 or p2 randomly
             end if

             if fitness value of o3 is less than fitness value of the selected parent
                replace the selected parent with o3
             end if
          end for
   end while
end while
```

Figure 3. 1 General schema of the proposed algorithm

3.1 Initial Population

Initial population consists of a predefined number of individuals (candidate solutions). Individuals of the initial population are created randomly and they consist of color classes containing vertices. Vertices in color classes are represented by integer values (vertex numbers). An individual contains all vertices in related graph and each vertex is assigned to only one color class. A color class of an individual cannot contain vertices having an edge between them (conflicting vertices). Therefore, initial population consists of individuals whose color classes are conflict free. A sample initial solution is illustrated in Figure 3.2. In Figure 3.2, a sample graph and an initial solution created for the graph is given. The solution consists of four color classes (color classes are represented with different colors (blue, pink, yellow and orange) for illustrative purposes) and numbers in each color class corresponds to vertex numbers given in the graph.

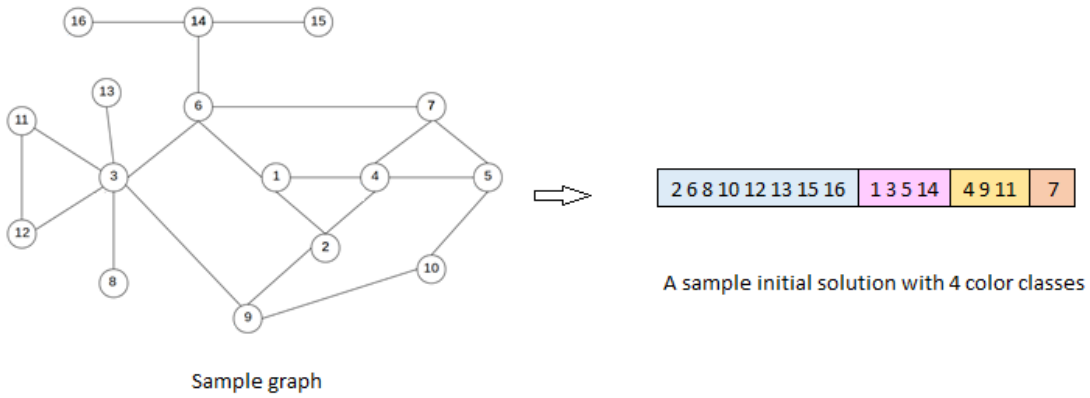


Figure 3. 2 Representation of a sample initial solution

To create an individual, each time an unassigned vertex is selected randomly and assigned a color. For each randomly selected vertex, previously created color classes are checked one by one starting from the first color class. If there is a previously created color class that does not have any conflicting vertices with the randomly selected vertex, the vertex is assigned to that color class and remaining color classes are not checked.

If there is no suitable color class, a new color class is created and the vertex is assigned to that color class. These steps are repeated until all vertices are assigned a color. When all vertices are assigned a color, the heaviest vertex in each color class is determined as

the cost of the related color classes. Sum of cost of color classes of each individual is calculated and assigned as fitness value of the related individual. General schema of the creation of the initial population is given in Figure 3.3.

```
Input: Graph G = (V , E)
Output: Initial Population P
C ← Ø /*set of colored vertices*/
v ← Ø /*v symbolizes a vertex*/
i = 1, j = 0
c = 0 /*number of created color classes*/
s ← predetermined population size

for i = 1,2, ..., s
   C ← Ø
   c = 0
   while V\C != Ø  /*while all vertices are not colored*/
      v ← select a vertex randomly from {V\C}
      for j = 1, ..., c
         if previously created j^th color class does not have any conflicting vertex with v
            assign v to j^th color class
            C ← C ∪ v   /*add v to C also*/
            break
         end if
      end for

      if v ∉ C
         create a new color class
         assign v to newly created color class
         C ← C ∪ v
         c ← c + 1
      end if
   end while
end for
```

Figure 3. 3 General schema of creation of initial population

3.1.1 Sample scenario for creation of an initial solution

To create an initial solution, the sample graph in Figure 3.4 is used. The undirected graph contains 16 vertices and the vertices are numbered starting from 1. Numbers in parenthesis represent weight values of the vertices. For example, weight value of vertex with number 1 is 4 and weight value of vertex with number 2 is equal to 3.

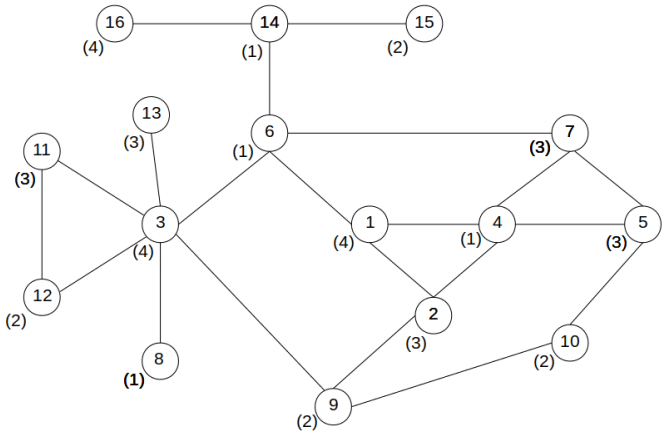


Figure 3. 4 Sample Graph

An empty color class is created and first three randomly selected vertices with numbers 2, 15 and 6 are placed in the newly created color class (first color class of the initial solution), respectively. Vertex with number 3 cannot be placed in the first color class because of conflicting with vertex with number 6. Consequently, another color class is created and it is placed in that color class (second color class of the initial solution). Vertices with numbers 13, 16 and 10 are placed in the first color class of the initial solution, respectively. On the other hand, vertices with numbers 5 and 1 are conflicting with vertices in the first color class. Since there is no vertex conflicting with these vertices in the second color class, they are placed in the second color class, respectively. Vertex with number 12 placed in first color class and vertex with number 14 placed in the second color class. Vertex with number 4 conflicts with vertices both in first and second color classes so third color class is created to place the node in. Vertices with numbers 11 and 9 conflict with vertices in both first and second color classes and they do not conflict with vertex with number 4. As a result, they are placed in the third color class, respectively. Vertex with number 7 conflicts with vertices in first, second and third color classes so fourth color class is created to place the vertex in. Vertex with number 8 is placed in the first color class and an initial solution is created with four color classes. Heaviest vertex (representative vertex) in first color class is vertex 16 and its weight value is 4. Heaviest vertices of the second color class are vertices with numbers 1 and 3 and they have the same weight value which is 4. Representative vertices of the third and fourth color classes are vertices with numbers 11 and 7,

respectively. These vertices have the same weight value which is 3. As a result, costs of the color classes are 4, 4, 3 and 3, respectively. Therefore, fitness value of the solution is equal to 14. Creation of the initial solution is illustrated in Figure 3.5.

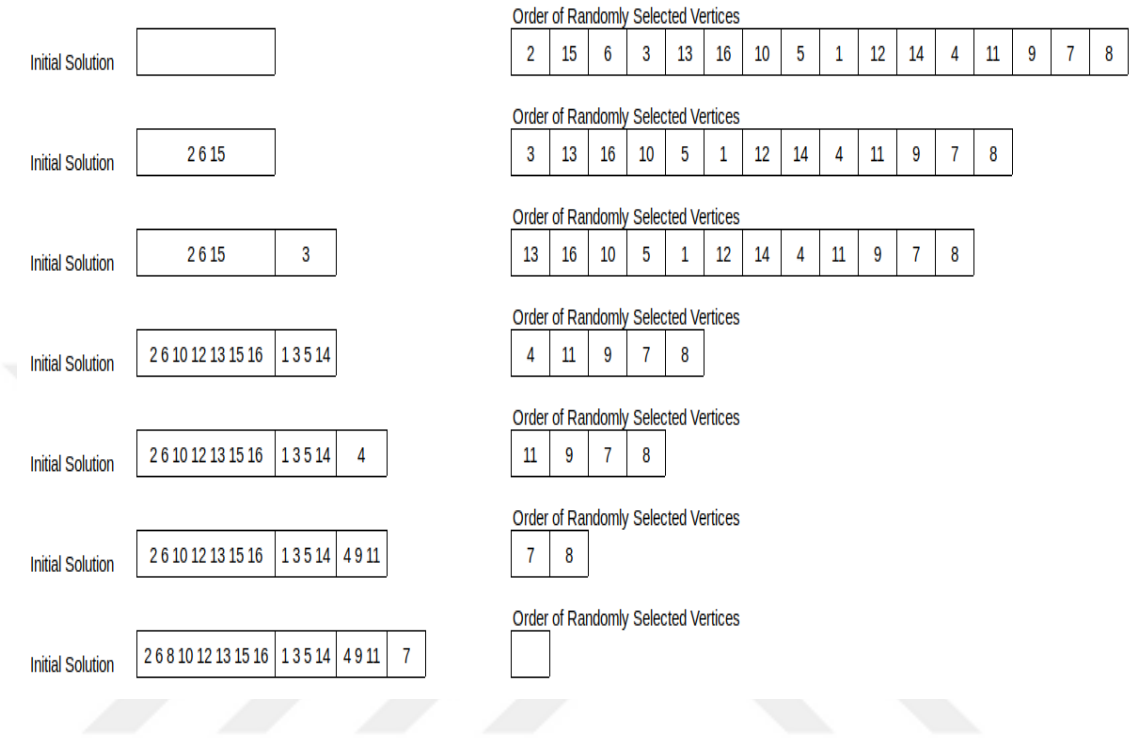


Figure 3. 5 Construction steps of the sample initial solution

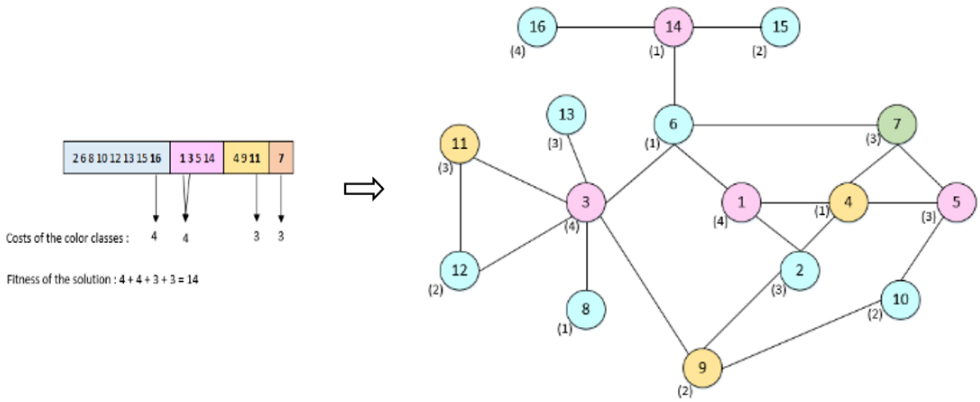


Figure 3. 6 Coloring the graph according to the initial solution obtained in Figure 3.5

3.2 Crossover Operators

We proposed two pool based crossover operators in which a structure called as pool is created to hold vertices that cannot be placed in any color class of offspring at different stages of the crossover process. Our crossover operators work together but give priority to different features. Details of the crossover operators are explained in related subsections.

3.2.1 Weight centered crossover operator (WCCO)

In crossover operator, two randomly selected solutions (parents) are used to create an offspring and color classes of offspring formed by combining color classes of parents. For this purpose, an empty color class is created for the offspring and one color class from each parents is selected randomly. Selected color classes are marked as selected to prevent them from being selected again. Vertices of these color classes are put together and vertices that exist in both color classes are counted only once. Vertices in the pool are also combined with these vertices. If any of combined vertices exist in any unselected color classes of the parents, they are deleted from the related color classes. Heaviest vertex is selected each time among the combined vertices and tried to be placed in newly created color class of the offspring.

While selecting the heaviest vertex, if there are more than one vertices having the same maximum weight, the one with the highest number of edges in the graph is selected among them. In case of having the same highest number edges in the graph, one of them is selected randomly. If the selected vertex has no edges between previously placed vertices in the color class, it is placed in the color class. Otherwise, it is tried to be placed in a suitable previously created color class of the offspring (determination of previously created suitable color classes is explained in the next section). If there is no suitable color class, it is placed in the pool.

When all of the combined vertices are placed in either color classes of the offspring or the pool, the same steps are followed starting from selection of random color classes from parents. Selection of color classes from parents finishes when all color classes of

either parents are selected or fifty percent of vertices of all color classes of either parents is deleted. If fifty percent condition is met, instead of combining two color classes together, all unplaced vertices including the vertices in the pool are combined together and the same placement procedure is followed. General schema of weight centered crossover operator is given in Figure 3.7.

3.2.1.1 Determination of the suitable previous color classes

When a vertex is tried to be placed in previously created color classes, suitable color classes are determined according to their cost values, number of vertices that conflict with the vertex in them and weights of the conflicting vertices. Color classes whose cost values are less than weight of the vertex and color classes containing more than one vertices conflicting with the vertex are not considered. If there is a color class with no edge between its vertices and the vertex, the vertex is placed in that color class. Otherwise, color classes where only one of its vertices conflicting with the vertex are checked according to following rules. If weight of conflicting vertex of a color class is greater than the weight of the vertex to be placed, that color class is eliminated. Furthermore, if a conflicting vertex has the same weight as the vertex but has greater than or equal number of edges between other vertices in the graph than the vertex has, the color class to which that conflicting vertex belongs to is ignored. Lastly, color class of a vertex can be changed at most two times. Therefore, if a conflicting vertex obeys the rules above but its color class has been changed twice before, it cannot be moved again and its color class is eliminated. General schema of determination of the suitable previous color classes is given in Figure 3.8.

```
Input: Two randomly selected parents
Output: Offspring
p ← Ø /*p symbolizes the pool*/
o ← Ø /*o symbolizes the set of color classes of the offspring*/
s ← Ø /*o symbolizes a color class*/
c ← Ø , n ← Ø /*c and n symbolize a set of vertices and n is used for the offspring*/
v ← Ø , r ← Ø /*v and r symbolize any vertex*/
k = 0 /*number of color class match operations*/
while all vertices are not placed
        c ← Ø
        if  fifty percent of vertices of all color classes of either parents is deleted
                c ← combine all vertices that are not deleted
                c ← c ∪ {p \ c}
                p ← Ø
        else if k is less than number of color classes of both parents
                select one previously unselected color class from each parents randomly
                c ← combine vertices in the selected color classes
                if s(c) > 0
                        c  ← c ∪ p
                        p ← Ø
                end if
        else
                if there is still unplaced vertices
                        c ← all unplaced vertices
                        p ← Ø
                end if
        end if
        if s(c) > 0
                n ← Ø
        while s(c) > 0
                v ← find the heaviest vertex in c
                if there is no vertex in n conflicting with v
                        n ← n ∪ v
                        c ← c \ v
                else
                        s ← find a suitable previously created color class of o
                        if s(s) = 0
                                p ← p ∪ v
                        else
                                s ← s ∪ v
                                if there is a conflicting vertex r in s
                                        s ← s \ r
                                        p ← p ∪ r
                                        increment movement count of r
                                end if
                        end if
                end if
        end while
                o ← o ∪ n
        end if
        k ← k + 1
end while
```

Figure 3. 7 General schema of Weight Centered Crossover Operator (WCCO)

```
Input: offspring
Output: a suitable previously created color class if exists
k ← number of color classes of the offspring – 1
c_i ← Ø  /*c_i symbolizes the i^th color class*/
f ← Ø, u ← Ø /*u and f are sets of color classes*/
i = 0

for i = 1, 2 , … , k
   if cost of the color class c_i is greater than or equal to the vertex
      if there is no conflicting vertex in c_i
         f ← f ∪ c_i
      else
         if number of conflicting vertices with the selected vertex is equal to 1 in c_i
            if weight of the conflicting vertex is not greater than weight of the vertex
               if weight of the conflicting vertex is equal to weight of the vertex
                  if number of edges of the conflicting vertex is less than number of edges of the vertex
                     u ← u ∪ c_i
                  end if
                  else
                     u ← u ∪ c_i
                  end if
               end if
         end if
      end if
   end if
end for

if s(f) is greater than 0
         return a randomly selected color class from f
else if s(u) is greater than zero
         return a randomly selected color class from u
else
         return Ø
end if
```

Figure 3. 8 Determination of Previous Color Classes for WCCO

3.2.1.2 Sample scenario for weight centered crossover operator

Steps of forming an offspring are illustrated in Figure 3.10, Figure 3.13, Figure 3.17 and Figure 3.18 using the same graph which is used in section 3.1.1. Two parents from the population are selected randomly and operator is applied on them to form an offspring. Randomly selected parents are illustrated in Figure 3.9. Both parents have four color classes. When our weight centered crossover operator is applied on the selected parents, the offspring is formed by creating 3 color classes. Creation of color classes of the offspring is explained in detail in the following subsections.

Forming first color class of the offspring

To form the first color class of the offspring, third color class of first parent and first color class of the second parent are selected randomly. Vertices in these color classes are combined together and set of vertices to be placed in the offspring is obtained (Figure 3.10).

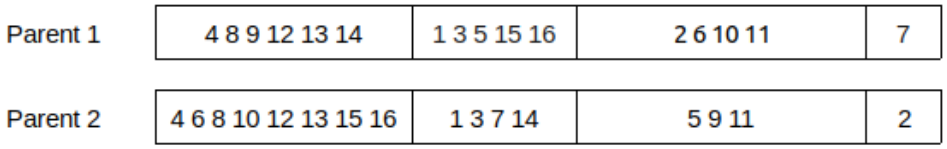


Figure 3. 9 Randomly Selected Parents

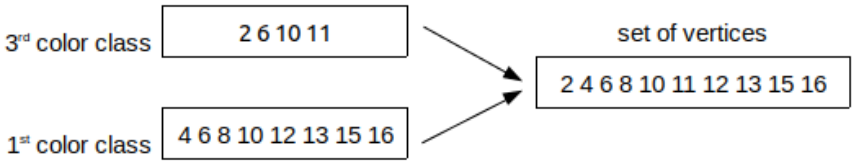


Figure 3. 10 Construction of set of vertices for the first color class

As illustrated in Figure 3.10, vertices that exist in both color classes are added to the set at once. For example, 6th vertex exists in both 3rd color class of the first parent and 1st color class of the second parent. Therefore, it is not placed in the set of vertices twice. If vertices in the set exist in any unselected color classes of the parents, they are deleted from the related color classes. For example, vertices with numbers 4, 8, 12, 13 and 15,16 are removed from first and second color classes of the first parent, respectively. Furthermore, vertices with numbers 11 and 12 are removed from third and fourth color classes of the second parent, respectively. An illustration of this process is given in Figure 3.11.

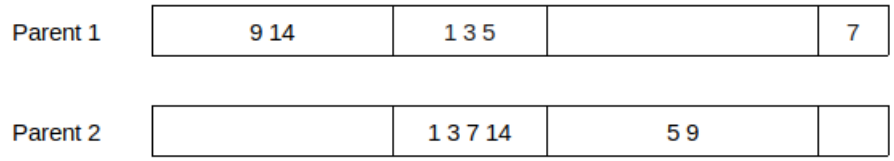


Figure 3. 11 Parents After Construction of the set of vertices

Heaviest vertex among the set is determined. In the sample set, vertex number of the heaviest vertex is 16 and maximum weight is equal to four. It is placed in the color class and maximum weight is updated as 3. Since there are more than one vertices having the same maximum weight, their number of edges is checked and the one with maximum number of edges is selected among them. Maximum weighted vertices in the set and corresponding number of edges are shown in Figure 3.12.

vertices	16	2	11	13	10	12	15	4	6	8
weights	4	3	3	3	2	2	2	1	1	1
edges	1	3	2	1	2	2	1	4	4	1

Figure 3. 12 Weight Values and Number of Edges of the Vertices in the Set of Vertices to be Placed

Vertex with number 2 is selected to be placed in the offspring as a result of having the highest number of edges. Consequently, vertex with number 4 is deleted from the set and placed in the pool, because it conflicts with vertex with number 2. Then, number of edges of the vertices with number 11 and 13 are compared and the vertex with number 11 is selected. This vertex is also placed in the offspring since it does not conflict with the previously placed vertices. Vertex with number 12 is deleted from the set and placed in the pool because of conflicting with the vertex with number 11. Afterwards, vertex with number 13 is placed in the color class and maximum weight is updated as 2. Finally, vertices with numbers 10, 15, 6 and 8 are also placed in the color class, respectively. Construction of first color class of the offspring is illustrated in Figure 3.13.

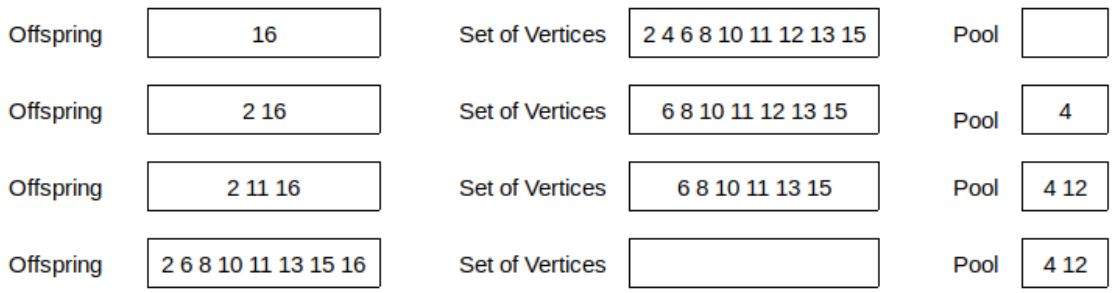


Figure 3. 13 Illustration of Construction of the First Color Class

Forming second color class of the offspring

After the construction of the first color class, fifty percentage rule is checked. If all color classes of a parent looses fifty percentage of their vertices, the parent obeys the fifty percentage rule. If at least one of the parents obey the fifty percentage rule, all vertices that are not placed in any color classes are combined together and crossover operator applied on them. While checking fifty percentage rule, color classes with just one vertex are counted as if they have lost fifty percentages of their vertices. Furthermore, color classes having odd number of vertices are counted as if they have lost fifty percentages of their vertices, if number of their deleted vertices is equal to half of their total number of vertices minus one. For example, if a color class has five vertices and two of them are deleted, the color class is counted as if it has lost fifty percentage of its vertices. First parent obeys the fifty percentage rule and second color class of the offspring is constructed accordingly.

To construct the second color class, all vertices that are not deleted are combined together and set of vertices to be placed is obtained. Furthermore, vertices in the pool are deleted from the pool and are placed in the set of vertices. Construction of the set is illustrated in Figure 3.14 and Figure 3.15.

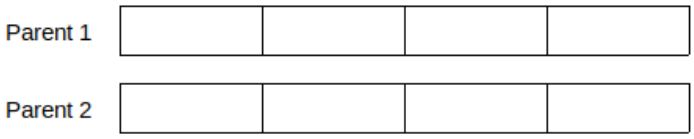


Figure 3. 14 Parents After Application of Fifty Percentage Rule

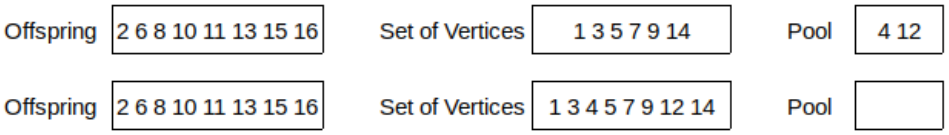


Figure 3. 15 Set of Vertices to be Placed Created According to Fifty Percentage Rule

Vertices in the set are placed in the color class following the rules of the operator. Weight and number of edges of each vertex in the set is given in Figure 3.16.

vertices	1	3	4	5	7	9	12	14
weights	4	4	1	3	3	2	2	1
edges	3	6	4	3	3	3	2	3

Figure 3. 16 Weight Values and Number of Edges of the Vertices in the Set of Vertices to be Placed

Vertices with numbers 3 and 1 are placed in the color class, respectively. Vertices with numbers 5 and 7 have both the same weight and number of edges. Therefore, vertex with number 7 is selected randomly and placed in the color class. Vertex with number 5 conflicts with vertex with number 7 so it cannot be placed in the color class. At this point, it is tried to be placed in previously created color classes. Cost of the first color class of the offspring is greater than weight value of the vertex and there exists just one vertex conflicting with it. Weight value of the conflicting vertex is smaller than weight of the vertex with number 5. Therefore, first color class is a suitable color class for the vertex and it is placed in first color class. To prevent conflict in the color class, the conflicting vertex is placed in the pool. Vertices with numbers 9, 12 and 4 are conflicting with the vertices in the second color class of the offspring and they are also conflicting with more than one vertices in the first color class. Consequently, these vertices are placed in the pool, respectively. Finally, vertex with number 14 is placed in second color class. Construction of the second color class is illustrated in Figure 3.17.

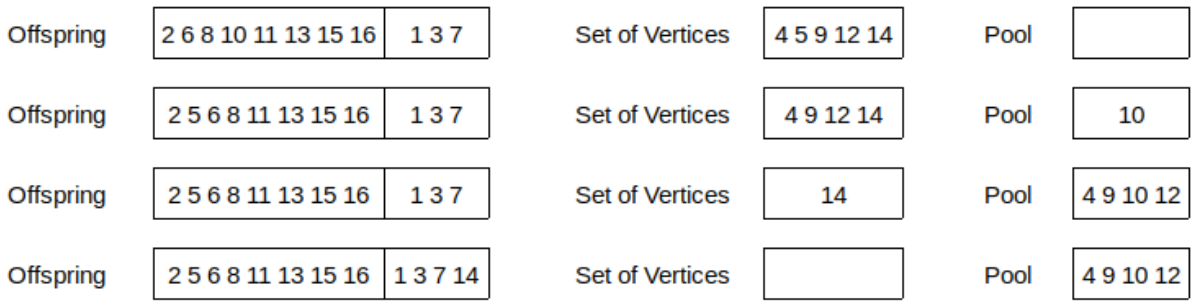


Figure 3. 17 Illustration of construction of the second color class

Forming third color class of the offspring

Our algorithm checks whether there are any vertices not placed in any color class after the fifty percentage rule. The pool contains vertices that are not placed in any color class. Therefore, they are combined together and tried to be placed following the same rules that are applied in previous steps. To place the vertices, a new color class is created. The maximum weight value is equal to two and vertex with number 9 is placed in newly created color class because of having more number of vertices than the vertices having the same weight value. Vertex with number 10 conflicts with vertex with number 9. Therefore, it cannot be placed in newly created color class and previously

created color classes are checked. Since the second color class does not contain vertices conflicting with it, it is placed in the second color class. Finally, vertices with numbers 12 and 4 are also placed in newly created color class, respectively. Construction of the third color class is illustrated in Figure 3.18 and construction of all color classes of the offspring is summarized in Figure 3.19.

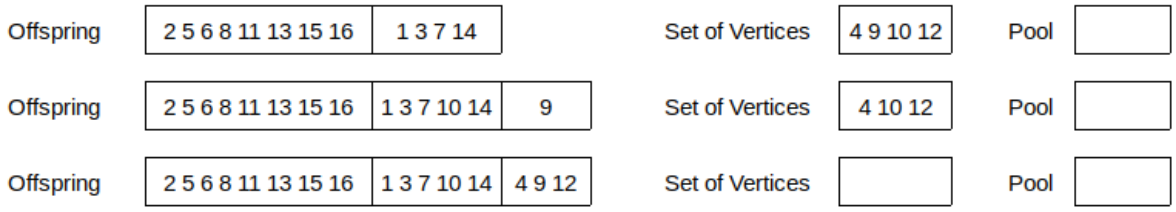


Figure 3. 18 Illustration of Construction of the Third Color Class

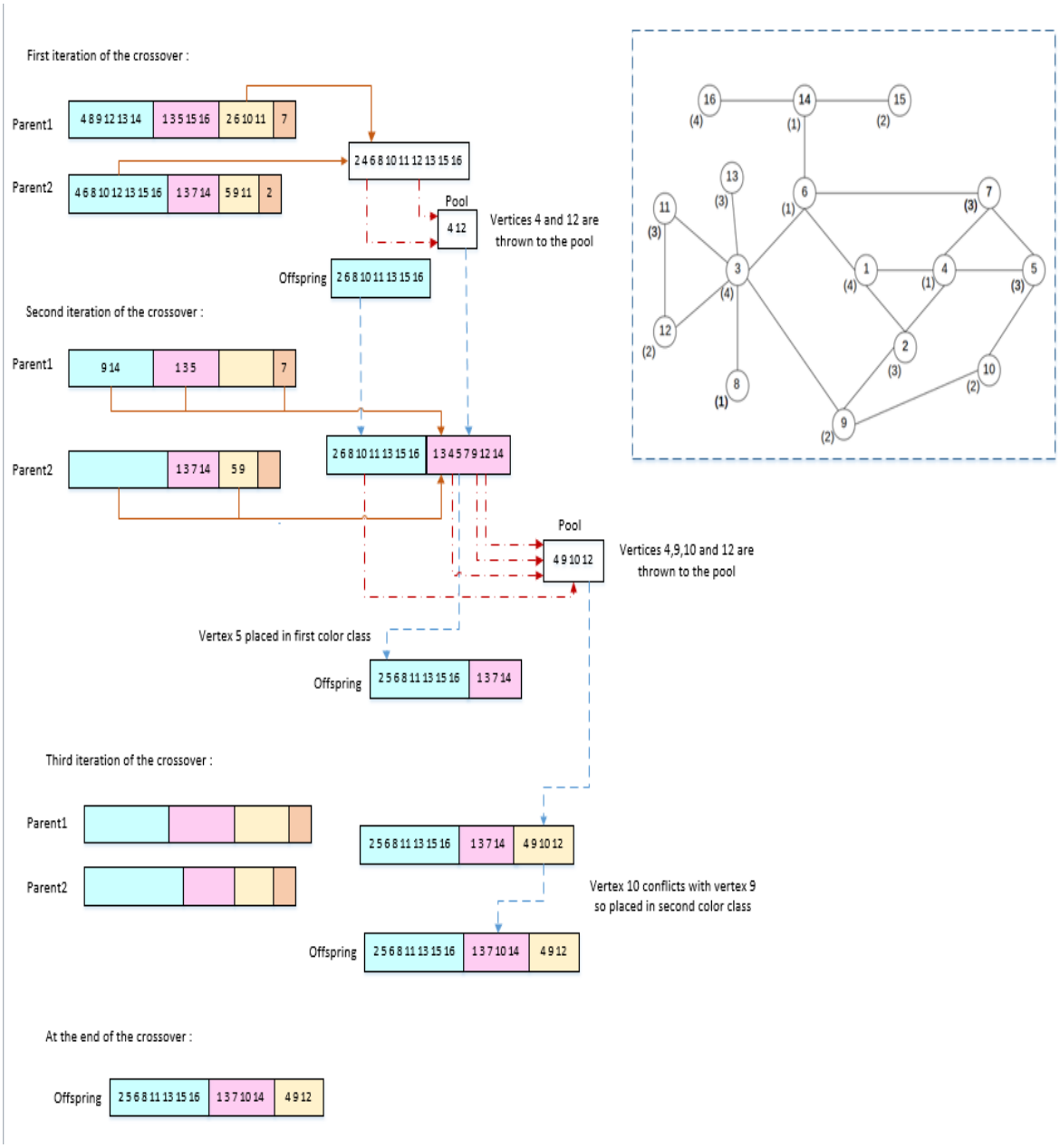


Figure 3. 19 Summarized construction of offspring by applying WCCO

3.2.2 Conflict centered crossover operator (CCCO)

In conflict centered crossover operator (Figure 3.20), two randomly selected solutions (parents) are used to create an offspring and color classes of offspring are formed by combining color classes of parents. For this purpose, an empty color class is created for the offspring and one color class from each parents is selected randomly. Selected color classes are marked as selected to prevent them from being selected again. Vertices of these color classes are put together and vertices that exist in both color classes are counted only once. Vertices in the pool are also combined with these vertices. If any of combined vertices exist in any unselected color classes of the parents, they are deleted from the related color classes.

For each vertex in combined vertices, number of vertices among combined vertices conflicting with the vertex are calculated. Conflict free vertices are selected each time among the combined vertices and placed in newly created color class (current color class) of the offspring. Vertex with maximum number of conflicts is placed in the pool. While selecting the vertex with maximum number of conflicts, if there are more than one vertices having the same maximum number of conflicts, the one with minimum weight is selected among them. Number of conflicts is recalculated for each remaining vertex and if any conflict free vertices exist they are also placed in current color class.

When all of the combined vertices are placed in either current color class of the offspring or in the pool, vertices remaining in the pool are tried to be placed in previously created color classes that do not contain vertices conflicting with them. For this purpose, each previously created color classes are visited and the first conflict free color class, if exists, is selected. The same steps are repeated, starting from selection of random color classes from parents, until stopping criteria is met. The stopping criteria is defined as generation of x number of color classes for the offspring where x denotes the minimum number of color classes one of the parents have.

```
Input: Two randomly selected parents
Output: Offspring
V ← set of all vertices
P ← Ø /*P symbolizes the pool*/
O ← Ø /*O symbolizes the set of color classes of the offspring*/
C ← Ø , n ← Ø , f ← Ø /*C, f and n symbolize a set of vertices*/
v ← Ø /*v symbolizes any vertex*/
k ← number of color classes of the parent that has the minimum number of color classes

for i = 1, 2 , …, k
   C ← Ø
   select one previously unselected color class from each parents randomly
   C ← combine vertices in the selected color classes
   if s(C) > 0
      n ← Ø
      C  ← C ∪ P
      P ← Ø
      calculate number of conflicts for each vertex v ϵ C
      n ← conflict free vertices in C
      C ← C \ n
      V ← V \ n
      while s(C) > 0
         v ← vertex with the maximum number of conflicts in C
         P ←P ∪ v
         calculate number of conflicts for each vertex v ϵ C
         if there exist any conflict free vertices in C
            f ← conflict free vertices in C
            n ← n ∪ f
            C ← C \ f
            V ← V \ f
         end if
      end while
      O ←O ∪ n
      for each vertex v ϵ P
         if there exists a color class n ϵ O that has not any vertices conflicting with v
            n ← n ∪ v
            P ← P \ v
            V ← V \ v
         end if
      end for
   end if
end for
if s(V) > 0
   for each vertex v ϵ V
      if there exists a color class n ϵ O with no vertices conflicting with v and cost of n >= weight of v
         n ←n ∪ v
      else
         if there exists a color class n ϵ O with no vertices conflicting with v and cost of n < weight of v
            n ← n ∪ v
         else
            create a new color class c (c ← Ø)
            c ← v
            O ← O ∪ c
         end if
      end if
   end for
end if
```

Figure 3. 20 Conflict Centered Crossover Operator

When the stopping criteria is met, unplaced vertices are tried to be placed. For this purpose, each color classes having cost value greater than or equal to weight of the unplaced vertex are checked. If there is a color class which has no conflicting vertex with the unplaced vertex, the unplaced vertex is placed in that color class. Otherwise, other color classes are searched. If there exists a color class meeting the condition, unplaced vertex is placed in that color class. If there are more than one color classes meeting the condition, the one with the greatest cost is selected. If the vertex is still unplaced, a new color class is created for it.

3.2.2.1 Sample scenario for conflict centered crossover operator

In this scenario, the same graph and parents that are used in weight centered crossover operator are used again. When the conflict centered crossover operator is applied on the parents, offspring is formed with three color classes.

Forming first color class of the offspring

Second color class of first parent and first color class of the second parent are selected randomly. Vertices in these color classes are combined together and set of vertices to be placed is obtained. This process is illustrated in Figure 3.21 and Figure 3.22.

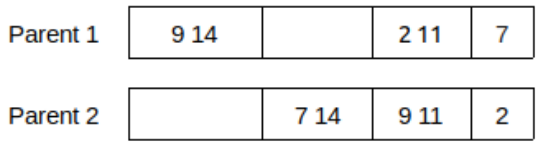


Figure 3. 21 Parents After Construction of Set of Vertices to be Placed

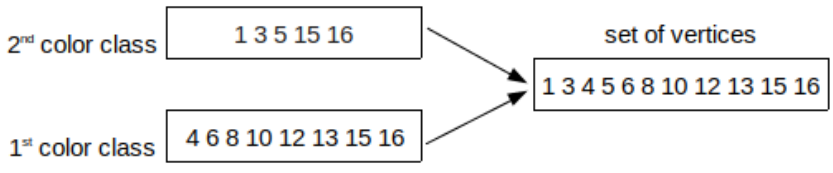
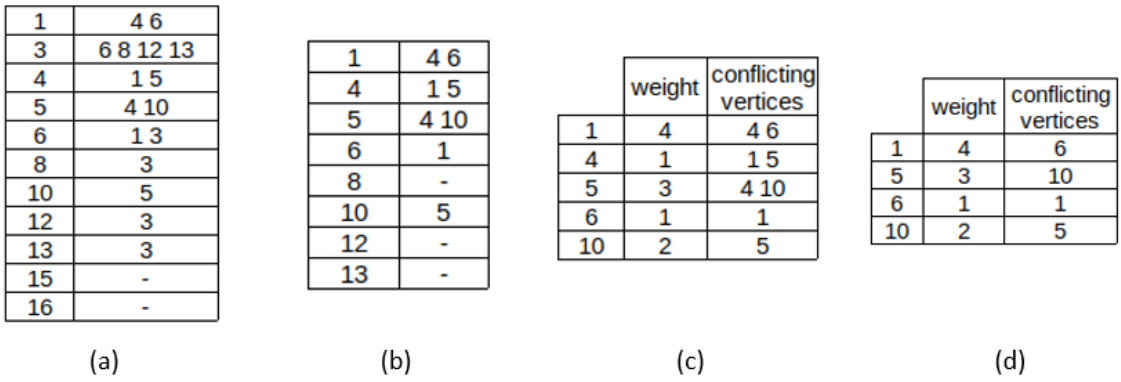


Figure 3. 22 Construction of Set of Vertices for the First Color Class

Vertices that exist in any unselected color classes of the parents are deleted from the related color classes. For example, vertices with numbers 6 and 10 are removed from third color class of first parent and vertices with number 1 and 3 are removed from second color class of second parent. Vertices conflicting with each other in the set are illustrated in Figure 3.23.



1	4 6
3	6 8 12 13
4	1 5
5	4 10
6	1 3
8	3
10	5
12	3
13	3
15	-
16	-

(a)

1	4 6
4	1 5
5	4 10
6	1
8	-
10	5
12	-
13	-

(b)

	weight	conflicting vertices
1	4	4 6
4	1	1 5
5	3	4 10
6	1	1
10	2	5

(c)

	weight	conflicting vertices
1	4	6
5	3	10
6	1	1
10	2	5

(d)

Figure 3. 23 Properties of the vertices **(a)** vertices and list of vertices conflicting with them in the set of vertices to be placed. **(b)** Updated version of Figure 3.23(a) after placement of vertex with number 3 in the pool. **(c)** Updated version of Figure 3.23(b) after placement in the pool of vertex with number 13. **(d)** Updated version of Figure 3.23(c) after placement of vertex with number 4 in the pool.

First conflict free vertices with numbers 15 and 16 are placed in first class of the offspring. Vertex with number 3 is placed in the pool because of conflicting with more number of vertices than other vertices in the set does. Number of conflicts in the set is calculated again and conflict free vertices with numbers 8, 12 and 13 are placed in the first color class. Vertices with numbers 1, 4 and 5 have the same number of conflicts so the vertex with number 4 is sent to pool because of having minimum weight value among them. Number of conflicts between the vertices is calculated again. Since all vertices have the same number of conflicts, vertex with number 6 is placed in the pool because of having minimum weight value among them. Conflict free vertex with number 1 is placed in the second color class. Properties of the remaining vertices are illustrated in Figure 3.24.

	weight	conflicting vertices
1	4	-
5	3	10
10	2	5

(a)

(b)

	weight	conflicting vertices
5	3	10
10	2	5

Figure 3. 24 Properties of the remaining vertices **(a)** updated version of Figure 3.23 (d) after placement of vertex with number 6 in the pool. **(b)** updated version of Figure 3.24 (a) after placement of conflict free vertex with number 1.

Vertices with number 5 and 10 conflicts with each other and vertex with number 10 is placed in the pool as a result of having smaller weight value. Construction of the first color class is illustrated in Figure 3.25.

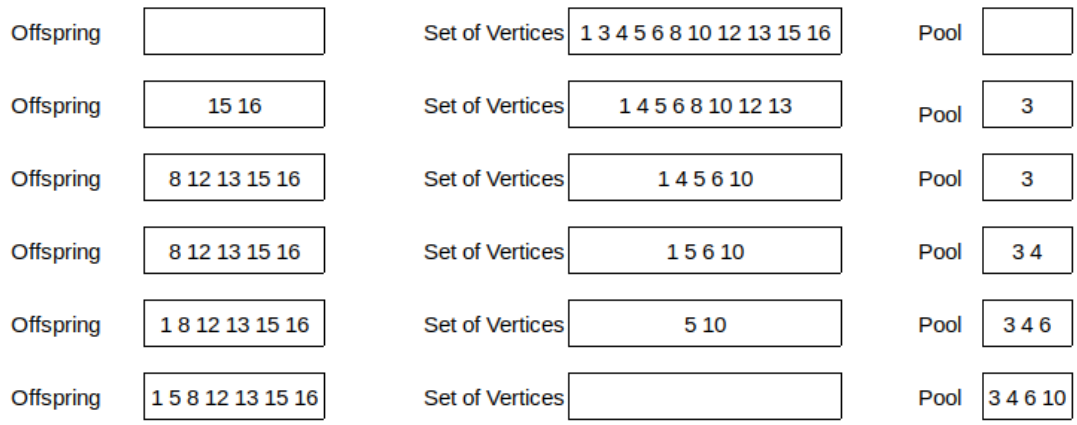


Figure 3. 25 Illustration of Construction of the First Color Class (Conflict Centered)

Forming second color class of the offspring

First color class of first parent and fourth color class of the second parent are selected randomly. Vertices in these color classes are combined together and set of vertices to be placed is obtained. Furthermore, vertices in the pool are deleted from the pool and placed in the set of vertices to be placed. Construction of the set is illustrated in Figure 3.26 and Figure 3.27.

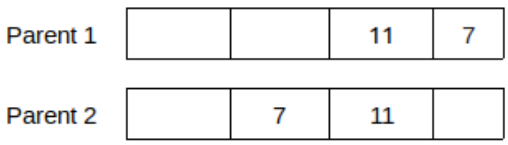


Figure 3. 26 Parents after construction of set of vertices to be placed (for 2nd color class)

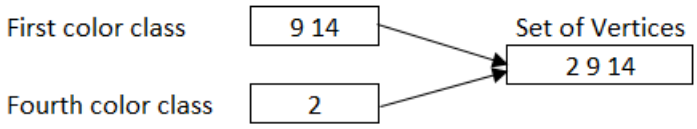


Figure 3. 27 Set of vertices to be placed(2nd color class)

Vertex with number 9 is placed in the pool because of having maximum number of conflicts. Number of conflicts in the set is calculated again and conflict free vertex with number 10 is placed in the second color class. Vertex with number 6 is placed in the pool and conflict free vertices with numbers 3 and 14 are placed in the second color class. Vertices with numbers 2 and 4 are conflicting with each other so vertex with number 4 is placed in the pool because of having smaller weight value. Vertex with number 9 is placed in conflict free first color class. Illustration of construction of the second color class is given in Figure 3.28.

Offspring	1 5 8 12 13 15 16		Set of Vertices	2 9 14	Pool	3 4 6 10
Offspring	1 5 8 12 13 15 16		Set of Vertices	2 3 4 6 9 10 14	Pool	
Offspring	1 5 8 12 13 15 16		Set of Vertices	2 3 4 6 10 14	Pool	9
Offspring	1 5 8 12 13 15 16	10	Set of Vertices	2 3 4 6 14	Pool	9
Offspring	1 5 8 12 13 15 16	3 10 14	Set of Vertices	2 4	Pool	6 9
Offspring	1 5 8 12 13 15 16	2 3 10 14	Set of Vertices		Pool	4 6 9
Offspring	1 5 8 9 12 13 15 16	2 3 10 14	Set of Vertices		Pool	4 6

Figure 3. 28 Illustration of construction of the second color class (Conflict Centered)

Forming third color class of the offspring

Third color class of first parent and second color class of the second parent are selected randomly. Vertices in these color classes are combined together and set of vertices to be placed is obtained. Furthermore, vertices in the pool are deleted from the pool and are added to the set of vertices to be placed. Construction of the set is illustrated in Figure 3.29 and Figure 3.30.

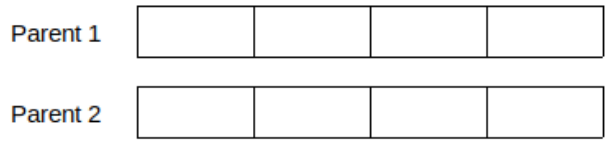


Figure 3. 29 Parents after construction of set of vertices to be placed

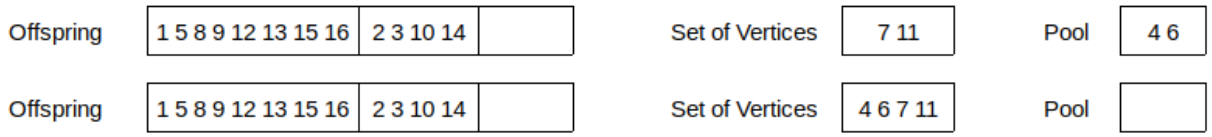


Figure 3. 30 Set of vertices to be placed(3rd color class)

Conflict free vertex with number 11 is placed in third color class of the offspring. Vertex with number 7 is placed in the pool because of having maximum number of conflicts. Consequently, conflict free vertices with numbers 4 and 6 are placed in the third color class. Finally, vertex with number 7 is placed in conflict free second color class. Weights and list of conflicting vertices of the vertices in the set is given in Figure 3.31. Construction of the third color class is illustrated in Figure 3.32.

	weights	conflicting vertices
4	1	7
6	1	7
7	3	4 6
11	3	-

Figure 3. 31 Weight values and list of conflicting vertices in the set of vertices to be placed

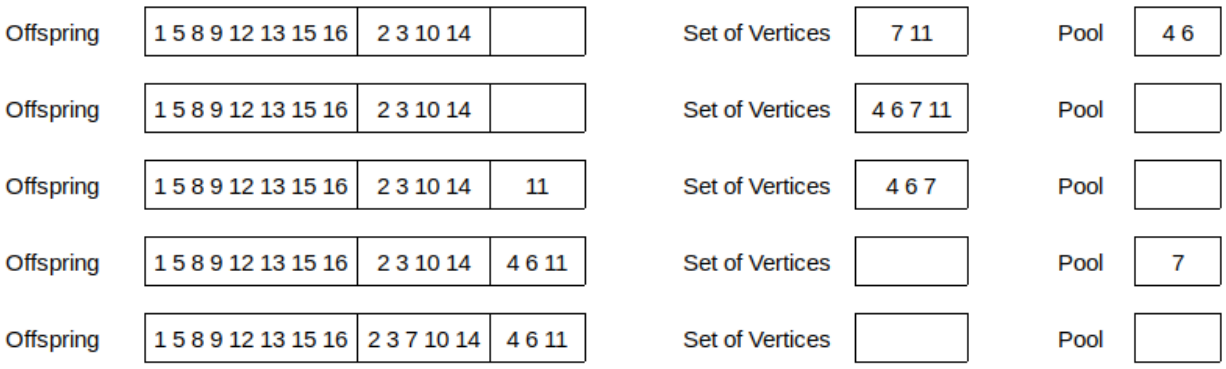


Figure 3. 32 Illustration of construction of the third color class

The offspring consists of three color classes and costs of these color classes are 4, 4 and 3, respectively. Therefore, fitness value of the offspring is equal to 11. Construction of all color classes of the offspring is illustrated in Figure 3.33.

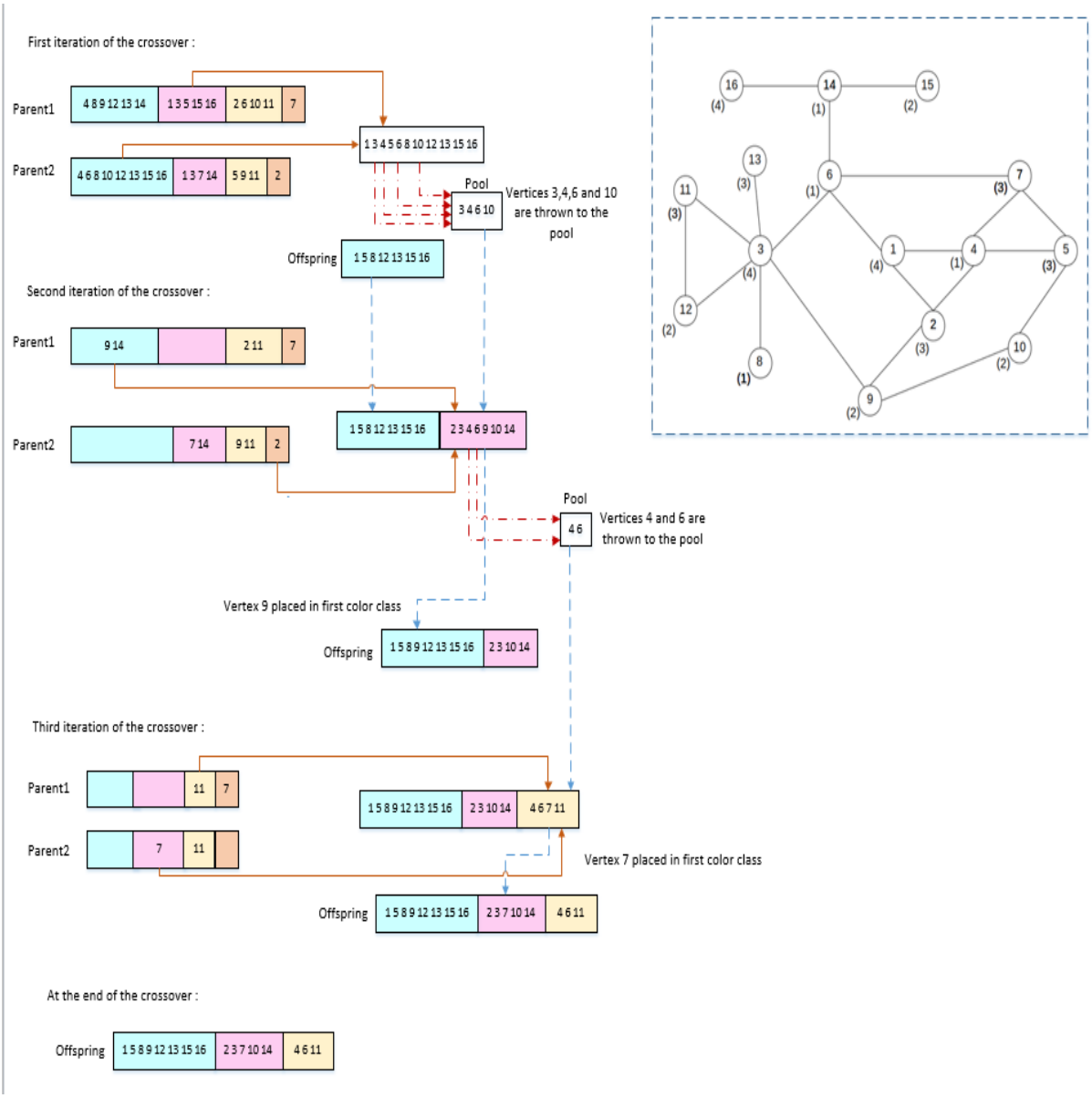


Figure 3. 33 Summarized construction of offspring by using CCCO

3.3 Mutation Operator

Our mutation operator visits the color classes starting from color classes with small costs and tries to place their representative vertices in other color classes in a conflict free way. If the representatives of some color classes can be placed in color classes with greater cost values, cost of these color classes can be reduced. As a result, fitness value of the individual can be reduced and a better individual is obtained.

In our mutation operator, a clone of the offspring is created and changes are applied on this clone. A color class of the clone which is not selected previously and whose cost is the minimum among other unselected color classes is selected each time. If there are more than one unselected color classes having the same minimum cost, one of them is selected randomly. Number of vertices whose weights are equal to cost of the selected color class (representative vertices) is determined. If number of representative vertices of the selected color class is less than three, representative vertices are tried to be placed to other color classes. For this purpose, other color classes whose weights are greater than or equal to the color class are visited for each representative vertex of the color class. If there is a color class which has no vertices conflicting with the representative vertex, representative vertex is placed in that color class. Otherwise, color classes which have at most two vertices conflicting with the representative vertex are checked and color classes having any representative vertices that are heavier than the representative vertex and conflicting with it are eliminated. Furthermore, in our mutation operation some vertices are marked as tabu vertices during different steps of the mutation operation and color classes of tabu vertices cannot be changed. Therefore, color classes containing tabu vertices that conflict with the representative vertex are also eliminated.

The color class which has the minimum number of vertices conflicting with the representative vertex, if exists, is selected among the possible color classes. If there are more than one color classes having the same minimum number of vertices conflicting with the representative vertex, sum of the edges of vertices conflicting with the representative vertex of each color class is calculated. Then, the color class which has the minimum sum of edges is selected. The representative vertex is placed in the selected color class and it is marked as a tabu vertex. The vertices conflicting with it are removed from the color class.

After placement of representative vertices, placement of removed vertices starts. To place a removed vertex in a suitable color class, color classes whose costs are greater than or equal to weight of the removed vertex are considered. In addition, if one of representative vertices of a color class conflicts with the removed vertex, the color class is ignored. Moreover, if a color class contains a tabu vertex that conflicts with the removed vertex, the color class is not considered. If a conflict free color class is found, the removed vertex is placed in it and the vertex is marked as a tabu vertex. Otherwise, color class which has the minimum number of vertices that conflicts with the removed vertex is selected as the suitable color class. If there are more than one color class having the same minimum number of conflicting vertices, the one which has the minimum total sum of the edges of its vertices conflicting with the removed vertex is selected and removed vertex is placed in that color class. If there is no suitable color class whose cost is greater than or equal to the removed vertex, color classes whose costs are less than weight of the removed vertex are considered. Color classes having more than 2 vertices conflicting with the removed vertex are eliminated. In addition, color classes having any tabu vertices conflicting with the moved vertex are also not considered. The color class with minimum number of vertices conflicting with the removed vertex is selected and removed vertex is placed in that color class. In addition, it is marked as a tabu vertex. If the vertex is still not placed in any color classes, a new color class is created to place the vertex and the vertex is marked as a tabu vertex. If a removed vertex is placed in a color having vertices conflicting with it, the conflicting vertices are removed from the related color class and tried to be placed again like other removed vertices. This process continues until all removed vertices are placed again.

After placement of all removed vertices, the mutation operator repeats the same steps starting from the selection of a color class from the clone which is not selected previously and whose cost is the minimum among other unselected color classes. The operator stops when all color classes meeting the selection criteria are selected.

3.3.1 Sample Scenario for Mutation Operator

In this sample scenario, offspring obtained in the sample scenario of conflict centered crossover operator is used. Color class having the minimum cost value is third color class and representative vertex of the color class is vertex with number 11. Cost values

of first and second color classes are greater than weight of the vertex and the vertex conflicts with just one vertex in both color classes. However, the vertex conflicts with the representative vertex of the second color class. Therefore, second color class cannot be a suitable color class for the vertex to be placed in. As a result, the vertex is placed in the first color class and marked as a tabu vertex. Vertex with number 12 is removed from the first color class because of conflicting with the vertex. Vertex with number 12 (removed vertex) conflicts with the representative vertex of the second color class whose cost value is greater than the weight value of the vertex. Consequently, it is marked as a tabu vertex and placed in the third color class which is conflict free. This process is illustrated in Figure 3.34.

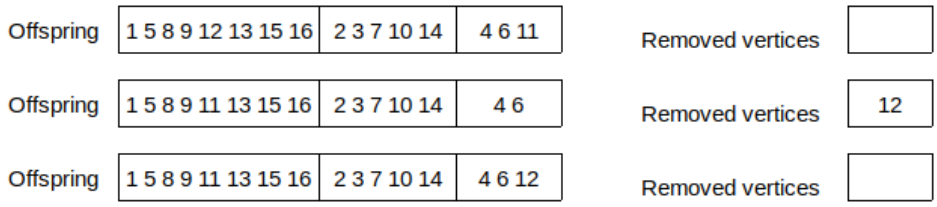


Figure 3. 34 Offspring after application of mutation operator by selecting the third color class

First and second color classes have the same cost value so first color class is selected randomly. Representative vertices of the first color class are 1 and 16. Vertex with number 1 conflicts with the vertex with number 2 in second color class. Consequently, the representative vertex is placed in the second color class and vertex with number 2 is removed from the second color class. Other representative vertex with number 16 conflicts with vertex with number 14 in second color class. Vertex with number 14 is removed from the second color class and the representative vertex is placed in the second color class. After placement of representative vertices, placement of removed vertices starts. For this purpose, removed vertex with number 2 is placed in the first color class and conflicting vertex with number 9 is removed from the first color class. Other removed vertex with number 14 conflicts with just one vertex in both first and third color classes. Conflicting vertex in the third color class has four edges and conflicting vertex in the first color class has one edge. Therefore, the removed vertex is placed in the first color class and conflicting vertex with number 15 is removed from the first color class. Removed vertices with numbers 9 and 15 are placed in conflict free

third and second color classes, respectively. After the mutation operation, color classes of the offspring have cost values 3, 4 and 2, respectively. As a result, obtained offspring has fitness value of 9 which is the best fitness value that can be found for the sample graph. Construction of the offspring for the explained steps above are illustrated in Figure 3.35 and summarized construction of the offspring by using the mutation operator is given in Figure 3.36.

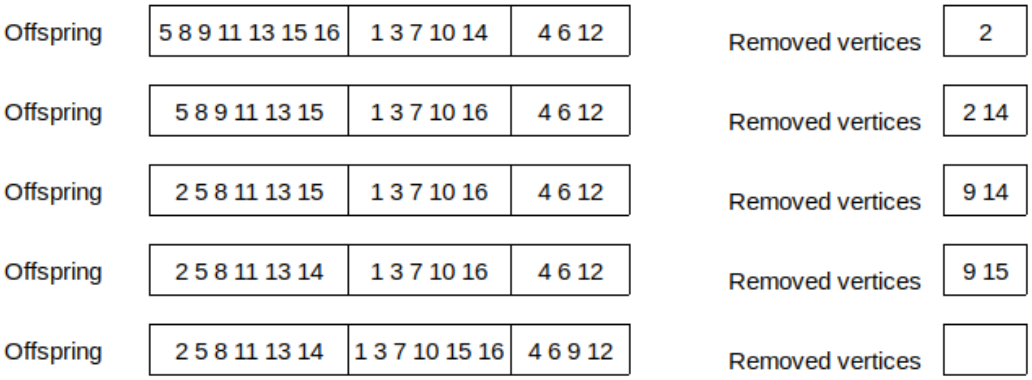


Figure 3. 35 Final Structure of the Offspring

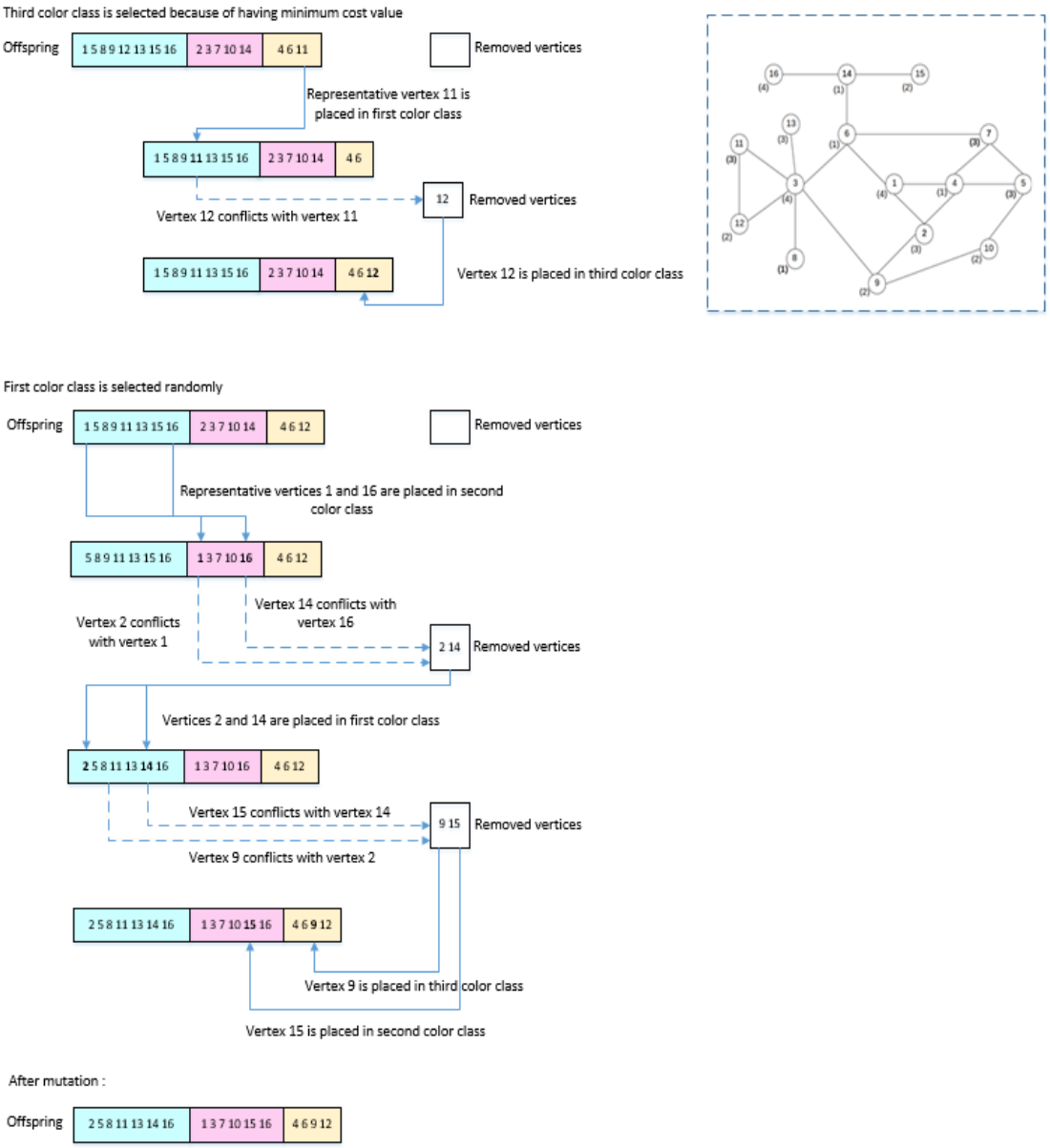


Figure 3. 36 Summarized construction of the sample mutation operator

4. EXPERIMENTAL STUDY

We tested our proposed algorithm on 46 DIMACS benchmark instances. Number of vertices for DSJC instances is 125. Number of vertices for GEOM instances varies between 30 and 120. In addition, number of vertices can be 50, 75 or 100 for selected R instances. We compared our results with the results of the studies [11, 12, 13] in the literature.

4.1 Experiments for Overall Algorithm

To test our algorithm, we selected population size as 100. The termination condition of the algorithm is determined as 8000 iterations. The stopping criteria for crossover operators is selected as 1000 iterations. Consequently, both conflict centered crossover operator and weight centered crossover operators are applied 4 times before the termination condition (8000 iterations) is reached. First crossover operator applied is the conflict centered crossover operator. For each DIMACS instance we run our algorithm 20 times and the best solution found among 20 runs is reported. In addition to best solution, average execution time and number of occurrences of the best solutions found are also reported for each instance. Results of the experiments are listed in Table 1. Results of AFISA, MWSS and 2-Phase are obtained from [11]. In Table 1, name of each benchmark instance, number of vertices (v) and number of edges (e) in related instance and best known value for each instance are listed, respectively. In addition, number of occurrences of best solutions (success rate (SR)), average of best values (Avg) through 20 runs are given for AFISA and our proposed algorithm. Average of total execution time in seconds of our algorithm is also available in Table 1.

Table 1 Benchmark Instances

Benchmark	v	e	Best Known Value	Proposed Algorithm				AFISA			MWSS	2-Phase
				Best	SR	Avg	Avg Time	Best	SR	Avg	Best	Best
DSJC125_1g	125	736	23	24	0	24.1	349.4	23	2	24	25	24
DSJC125_1gb	125	736	90	90	3	92.8	331.1	90	1	92.5	95	95
DSJC125_5g	125	3891	71	73	0	74.2	1098.9	71	2	72.3	78	76
DSJC125_5gb	125	3891	243	242	1	245.3	1130.9	243	1	250.2	263	251
DSJC125_9g	125	6961	169	169	6	169.7	4725.0	169	3	169.9	169	169
DSJC125_9gb	125	6961	604	606	0	606.8	4445.2	604	3	605.5	604	605
GEOM 100	100	547	65	65	20	65.0	349.6	65	20	65	65	65
GEOM 100 A	100	992	89	89	2	89.9	586.5	89	10	89.5	89	89
GEOM 100 B	100	1050	32	32	20	32.0	534.6	32	1	33.1	32	32
GEOM 110	110	638	68	68	20	68.0	389.3	68	20	68	68	69
GEOM 110 A	110	1207	97	97	6	97.7	699.4	97	6	97.8	97	97
GEOM 110 B	110	1256	37	37	20	37.0	643.1	37	14	37.9	37	37
GEOM120	120	773	72	72	20	72.0	446.9	72	20	72	72	72
GEOM120 A	120	1434	105	106	0	106.6	790.8	105	1	106.3	105	105
GEOM120 B	120	1491	35	36	0	36.0	753.2	35	7	37.3	35	35
GEOM 30 B	30	81	12	12	20	12.0	148.3	12	20	12	12	12
GEOM 40 B	40	157	16	16	20	16.0	176.4	16	20	16	16	16
GEOM 50 B	50	249	18	18	20	18.0	203.7	18	20	18	18	18
GEOM 60 B	60	366	23	23	20	23.0	244.2	23	20	23	23	23
GEOM 70	70	267	47	47	20	47.0	224.2	47	20	47	47	47
GEOM 70 A	70	459	73	73	20	73.0	298.0	73	20	73	73	73
GEOM 70 B	70	488	24	24	20	24.0	287.0	24	20	24	24	24
GEOM 80	80	349	66	66	20	66.0	259.4	66	20	66	66	66
GEOM 80 A	80	612	76	76	20	76.0	365.4	76	19	76.1	76	76
GEOM 80 B	80	663	27	27	20	27.0	360.1	27	5	27.8	27	27
GEOM 90	90	441	61	61	20	61.0	295.0	61	16	61.2	61	61
GEOM 90 A	90	789	73	74	0	74.0	438.5	73	3	74	73	73
GEOM 90 B	90	860	30	30	20	30.0	431.5	30	19	30.1	30	30
R100-1G	100	509	21	22	0	22.0	277.2	21	1	22	21	22
R100-1GB	100	509	81	82	0	83.8	273.1	81	1	83.8	81	-
R100-5G	100	2456	59	59	20	59.0	691.6	59	5	60.1	59	-
R100-5GB	100	2456	221	222	0	223.4	680.6	221	1	224.1	225	-
R100-9G	100	4438	141	141	20	141.0	2221.9	141	15	141.3	141	-
R100-9GB	100	4438	518	519	0	520.7	2147.1	518	1	549.3	518	-
R50-1G	50	108	14	14	20	14.0	166.2	14	20	14	14	14
R50-1GB	50	108	53	53	20	53.0	170.6	53	20	53	53	53
R50-5G	50	612	37	37	20	37.0	280.6	37	20	37	37	37
R50-5GB	50	612	135	135	9	136.6	275.6	135	14	135.3	135	137
R50-9G	50	1092	74	74	20	74.0	543.8	74	20	74	74	74
R50-9GB	50	1092	262	262	19	262.1	539.8	262	20	262	262	262
R75-1G	70	251	18	18	17	18.2	199.0	18	12	18.4	18	19
R75-1GB	70	251	70	70	10	71.0	199.9	70	19	70.1	70	72
R75-5G	75	1407	51	51	12	51.4	439.1	51	12	51.4	51	53
R75-5GB	75	1407	186	186	3	187.6	477.0	186	2	189	186	190
R75-9G	75	2513	110	110	20	110.0	1377.6	110	20	110	110	110
R75-9GB	75	2513	396	396	20	396.0	1351.3	396	12	396.4	396	399

We found a new best value for DSJC125_5GB benchmark instance. The best value found by our algorithm is 242 whereas, this value is equal to 243 for AFISA, 263 for MWSS and 251 for 2-Phase algorithms. We have obtained worse results than the best known values of 10 benchmarks instances. We have obtained best value as 73 for DSJC125_5G instance and our best value is worse than the value AFISA founds which is 71. On the other hand, our best value is better than the best values found by MWSS and 2-Phase which are 78 and 76, respectively. We found worse value than AFISA for R100_5GB instance. The best value found by our algorithm is 222 whereas the best value found by AFISA is 221. Conversely, this value is better than the best value found by MWSS which is 225. Our algorithm found best value as 24 for DSJC125_1G instance which is found as 23 by AFISA. Our best value is equal to the best value found by 2-Phase algorithm and less than the best value which is 25 found by MWSS. Remaining seven instances are DSJC125_9GB, GEOM120A, GEOM120B, GEOM90A, R100_1G, R100_1GB and R100_9GB. Our best value is equal to 606 for DSJC125_9GB but best known value is equal to 604. We obtained one more greater value than each of the best known values for GEOM120A, GEOM120B, GEOM90A, R100_1G, R100_1GB and R100_9GB instances. We found 106, 36, 74, 22, 82 and 519 for these instances, respectively. The best known values for these instances are 105, 35, 73, 21, 81 and 519, respectively. On the other hand, for 36 benchmark instances out of 46 instances we found the same values as the best known values in the literature.

In addition to best values found for benchmark instances, we also checked the SR values (number of best values found) over 20 runs for each instance to evaluate performance of our algorithm. We first checked number of benchmark instances for which the best known value is found 20 times out of 20 runs. AFISA finds the best known value 20 times over 20 runs for 17 benchmark instances. On the other hand, our algorithm finds the best known value 20 times over 20 runs for 25 benchmark instances. Both AFISA and our algorithm have the same SR value for 16 benchmark instances which are GEOM100, GEOM110, GEOM120, GEOM30B, GEOM40B, GEOM50B, GEOM60B, GEOM70, GEOM70A, GEOM70B, GEOM80, R50-1G, R50-1GB, R50-5G, R50-9G and R75-9G instances. The only benchmark instance for which SR value is equal to 20 for AFISA but not equal to 20 for our algorithm is R50-9GB instance whereas, our SR value is equal to 19 for R50-9GB instance. On the other hand, for eight

benchmark instances SR value of our algorithm is equal to 20 while SR value of the AFISA is less than 20. SR values of AFISA for GEOM100B, GEOM110B, GEOM80A, GEOM80B, GEOM90, GEOM90B, R100_9G and R75_9GB instances are 1, 14, 19, 5, 16, 19, 15 and 12, respectively.

To check the performance of our algorithm, we also calculated average of best values found over 20 runs for each benchmark instances. Both AFISA and our algorithm has the same average value for 20 instances out of 46 instances. Our algorithm has worse average value than AFISA for 9 benchmark instances and better average value for 17 instances. In addition, if we eliminate the cases where the difference between average value of AFISA and average value of our algorithm is less than 1, in this case, our algorithm has worse average value for 3 instances and better average value for 6 instances.

4.2 Experiments for Mutation Operator

Our mutation operator tries to place representative vertices to color classes whose cost value are greater than or equal to weight of the representative vertices. If the representative vertices can be placed in color classes whose cost values are greater than or equal, cost of color classes from which they are removed can be reduced. Furthermore, reduced cost values of color classes may result in smaller fitness values. We tested our algorithm with mutation operator and reported the results in Table 1. We wanted to analyze the performance of our mutation operator if representative vertices are let to be placed in color classes whose cost value is equal to one or two less than the weight of the representative vertex to be placed. In this way, we tried to understand whether increasing cost values of some color classes by one or two may results in more suitable placement of remaining vertices therefore reducing the cost values of color classes more than the increased cost values. If total of increase in cost values of color classes is less than total of decrease in cost values of color classes, fitness value of solutions can be reduced.

To test our algorithm, we selected population size as 100. The termination condition of the algorithm is determined as 8000 iterations. The stopping criteria for crossover operators is selected as 1000 iterations. First crossover operator applied is the conflict

centered crossover operator. For each DIMACS instance we run our algorithm 20 times. Our results are reported in Table 2. In Table 2, our original mutation operator is represented as Mutation0. The mutation operator which considers color classes with weight value one less than weight values of representative vertices is represented as Mutation1 and the mutation operator which considers color classes with weight value two less than weight values of representative vertices is represented as Mutation2.

Table 2 Comparison of Mutation Operators

	v	e	Best Known Value	Mutation 0			Mutation1			Mutation2		
				Best	SR	Avg	Best	SR	Avg	Best	SR	Avg
DSJC125_1g	125	736	23	24	0	24.1	**23**	1	24.0	**23**	2	24.3
DSJC125_1gb	125	736	90	90	3	92.8	90	5	92.3	90	2	92.8
DSJC125_5g	125	3891	71	73	0	74.2	74	0	74.9	74	0	75.0
DSJC125_5gb	125	3891	242	242	1	245.3	242	1	244.9	242	2	245.7
DSJC125_9g	125	6961	169	169	6	169.7	169	6	169.7	169	8	169.6
DSJC125_9gb	125	6961	604	606	0	606.8	606	0	606.6	606	0	606.8
GEOM 100	100	547	65	65	20	65.0	65	20	65.0	65	20	65.0
GEOM 100 A	100	992	89	89	2	89.9	89	1	90.0	89	3	89.9
GEOM 100 B	100	1050	32	32	20	32.0	32	20	32.0	32	20	32.0
GEOM 110	110	638	68	68	20	68.0	68	20	68.0	68	20	68.0
GEOM 110 A	110	1207	97	97	6	97.7	97	5	97.8	97	5	97.8
GEOM 110 B	110	1256	37	37	20	37.0	37	20	37.0	37	20	37.0
GEOM120	120	773	72	72	20	72.0	72	20	72.0	72	20	72.0
GEOM120 A	120	1434	105	106	0	106.6	106	0	106.6	106	0	106.6
GEOM120 B	120	1491	35	36	0	36.0	36	0	36.0	36	0	36.0
GEOM 30 B	30	81	12	12	20	12.0	12	20	12.0	12	20	12.0
GEOM 40 B	40	157	16	16	20	16.0	16	20	16.0	16	20	16.0
GEOM 50 B	50	249	18	18	20	18.0	18	20	18.0	18	20	18.0
GEOM 60 B	60	366	23	23	20	23.0	23	20	23.0	23	20	23.0
GEOM 70	70	267	47	47	20	47.0	47	20	47.0	47	20	47.0
GEOM 70 A	70	459	73	73	20	73.0	73	20	73.0	73	20	73.0
GEOM 70 B	70	488	24	24	20	24.0	24	20	24.0	24	20	24.0
GEOM 80	80	349	66	66	20	66.0	66	20	66.0	66	20	66.0
GEOM 80 A	80	612	76	76	20	76.0	76	20	76.0	76	20	76.0
GEOM 80 B	80	663	27	27	20	27.0	27	20	27.0	27	20	27.0
GEOM 90	90	441	61	61	20	61.0	61	20	61.0	61	20	61.0
GEOM 90 A	90	789	73	74	0	74.0	**73**	1	74.0	**73**	1	74.0
GEOM 90 B	90	860	30	30	20	30.0	33	20	30.0	30	20	30.0
R100-1G	100	509	21	22	0	22.0	22	0	22.0	22	0	22.0
R100-1GB	100	509	81	82	0	83.8	82	0	83.6	82	0	84.0
R100-5G	100	2456	59	59	20	59.0	59	20	59.0	59	19	59.1
R100-5GB	100	2456	221	222	0	223.4	**220**	1	223.6	222	0	223.6
R100-9G	100	4438	141	141	20	141.0	141	20	141.0	141	20	141.0
R100-9GB	100	4438	518	519	0	520.7	519	0	520.8	519	0	520.6
R50-1G	50	108	14	14	20	14.0	14	20	14.0	14	20	14.0
R50-1GB	50	108	53	53	20	53.0	53	20	53.0	53	20	53.0
R50-5G	50	612	37	37	20	37.0	37	20	37.0	37	18	37.1
R50-5GB	50	612	135	135	9	136.6	135	7	136.9	135	7	137.0
R50-9G	50	1092	74	74	20	74.0	74	20	74.0	74	20	74.0
R50-9GB	50	1092	262	262	19	262.1	262	18	262.1	262	18	262.1
R75-1G	70	251	18	18	17	18.2	18	16	18.2	18	16	18.2
R75-1GB	70	251	70	70	10	71.0	70	7	71.7	70	9	71.5
R75-5G	75	1407	51	51	12	51.4	51	16	51.2	51	16	51.2
R75-5GB	75	1407	186	186	3	187.6	186	1	188.7	186	5	188.1
R75-9G	75	2513	110	110	20	110.0	110	20	110.0	110	20	110.0
R75-9GB	75	2513	396	396	20	396.0	396	20	396.0	396	20	396.0

In these experiments, we found a new best value for R100_5GB benchmark instance as 220 by using Mutation1 operator. The new best value is represented in bold in the table. Furthermore, we found the best known value for DSJC125_1G instance by using Mutation1 and Mutation2 operators. In addition to DSJC125_1G instance, we found the best known value for GEOM90A by using Mutation1 and Mutation2 operators. These values are also represented in bold in the table. With these experiments, number of benchmark instances for which better values than the values in the literature found are increased to 2. Moreover, number of instances for which the best known solution is not obtained decreased from 10 to 7. We have updated the related values in Table 1 and the new version of Table 1 is given as Table 3.

4.3 Experiments for Crossover Operators

We have applied two crossover operators which are weight centered crossover operator and conflict centered crossover operator to solve the WVCP. We wanted to observe behavior of the crossover operators. For this purpose, we conducted extra experiments and we did not apply the mutation operator to observe behavior of the crossover operators. We have conducted 3 groups of experiments. In first experiment group, only weight centered crossover operator is applied on population during execution of the algorithm. In second experiment group, only conflict centered crossover operator is applied on population during execution of the algorithm. Finally, in third experiment group, weight centered and conflict centered crossover operators are applied one after another on population during execution of the algorithm. To test our algorithm, we selected population size as 100 and the termination condition of the algorithm is determined as 10000 iterations for three experiment groups. The stopping criteria for crossover operators is selected as 1000 iterations for third experiment group. Consequently, both conflict centered crossover operator and weight centered crossover operators are run 5 times before the termination condition (10000 iterations) is reached and first crossover operator applied is the conflict centered crossover operator. We conducted our experiments on four DIMACS instances which are DSJC125_5GB, R75-5GB, R100-5GB and R100-9GB instances. We run our algorithm 20 times for each instance. Obtained results are illustrated in Figure 4.1, Figure 4.2 and Figure 4.3.

Table 3 Updated Version of Table 1

Benchmark	v	e	Best Known Value	Proposed Algorithm				AFISA			MWSS	2-Phase
				Best	SR	Avg	Avg Time	Best	SR	Avg	Best	Best
DSJC125_1g	125	736	23	**23**	0	24.1	349.4	23	2	24	25	24
DSJC125_1gb	125	736	90	90	3	92.8	331.1	90	1	92.5	95	95
DSJC125_5g	125	3891	71	73	0	74.2	1098.9	71	2	72.3	78	76
DSJC125_5gb	125	3891	**242**	242	1	245.3	1130.9	243	1	250.2	263	251
DSJC125_9g	125	6961	169	169	6	169.7	4725.0	169	3	169.9	169	169
DSJC125_9gb	125	6961	604	606	0	606.8	4445.2	604	3	605.5	604	605
GEOM 100	100	547	65	65	20	65.0	349.6	65	20	65	65	65
GEOM 100 A	100	992	89	89	2	89.9	586.5	89	10	89.5	89	89
GEOM 100 B	100	1050	32	32	20	32.0	534.6	32	1	33.1	32	32
GEOM 110	110	638	68	68	20	68.0	389.3	68	20	68	68	69
GEOM 110 A	110	1207	97	97	6	97.7	699.4	97	6	97.8	97	97
GEOM 110 B	110	1256	37	37	20	37.0	643.1	37	14	37.9	37	37
GEOM120	120	773	72	72	20	72.0	446.9	72	20	72	72	72
GEOM120 A	120	1434	105	106	0	106.6	790.8	105	1	106.3	105	105
GEOM120 B	120	1491	35	36	0	36.0	753.2	35	7	37.3	35	35
GEOM 30 B	30	81	12	12	20	12.0	148.3	12	20	12	12	12
GEOM 40 B	40	157	16	16	20	16.0	176.4	16	20	16	16	16
GEOM 50 B	50	249	18	18	20	18.0	203.7	18	20	18	18	18
GEOM 60 B	60	366	23	23	20	23.0	244.2	23	20	23	23	23
GEOM 70	70	267	47	47	20	47.0	224.2	47	20	47	47	47
GEOM 70 A	70	459	73	73	20	73.0	298.0	73	20	73	73	73
GEOM 70 B	70	488	24	24	20	24.0	287.0	24	20	24	24	24
GEOM 80	80	349	66	66	20	66.0	259.4	66	20	66	66	66
GEOM 80 A	80	612	76	76	20	76.0	365.4	76	19	76.1	76	76
GEOM 80 B	80	663	27	27	20	27.0	360.1	27	5	27.8	27	27
GEOM 90	90	441	61	61	20	61.0	295.0	61	16	61.2	61	61
GEOM 90 A	90	789	73	**73**	0	74.0	438.5	73	3	74	73	73
GEOM 90 B	90	860	30	30	20	30.0	431.5	30	19	30.1	30	30
R100-1G	100	509	21	22	0	22.0	277.2	21	1	22	21	22
R100-1GB	100	509	81	82	0	83.8	273.1	81	1	83.8	81	-
R100-5G	100	2456	59	59	20	59.0	691.6	59	5	60.1	59	-
R100-5GB	100	2456	**220**	220	0	223.4	680.6	221	1	224.1	225	-
R100-9G	100	4438	141	141	20	141.0	2221.9	141	15	141.3	141	-
R100-9GB	100	4438	518	519	0	520.7	2147.1	518	1	549.3	518	-
R50-1G	50	108	14	14	20	14.0	166.2	14	20	14	14	14
R50-1GB	50	108	53	53	20	53.0	170.6	53	20	53	53	53
R50-5G	50	612	37	37	20	37.0	280.6	37	20	37	37	37
R50-5GB	50	612	135	135	9	136.6	275.6	135	14	135.3	135	137
R50-9G	50	1092	74	74	20	74.0	543.8	74	20	74	74	74
R50-9GB	50	1092	262	262	19	262.1	539.8	262	20	262	262	262
R75-1G	70	251	18	18	17	18.2	199.0	18	12	18.4	18	19
R75-1GB	70	251	70	70	10	71.0	199.9	70	19	70.1	70	72
R75-5G	75	1407	51	51	12	51.4	439.1	51	12	51.4	51	53
R75-5GB	75	1407	186	186	3	187.6	477.0	186	2	189	186	190
R75-9G	75	2513	110	110	20	110.0	1377.6	110	20	110	110	110
R75-9GB	75	2513	396	396	20	396.0	1351.3	396	12	396.4	396	399

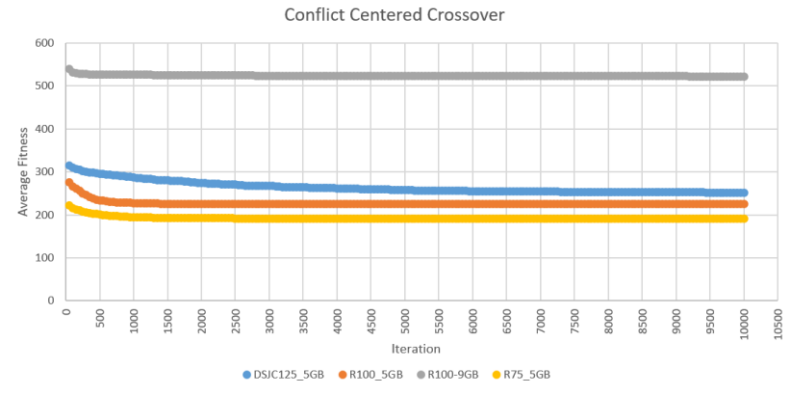


Figure 4. 1 Experiment on Conflict Centered Crossover Operator

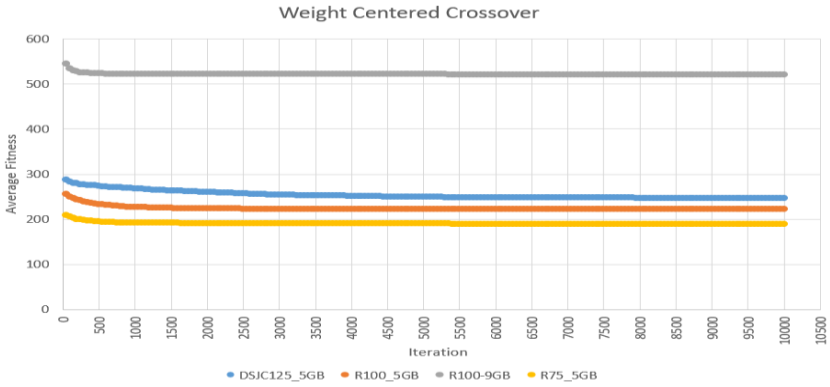


Figure 4. 2 Experiment on Weight Centered Crossover Operator

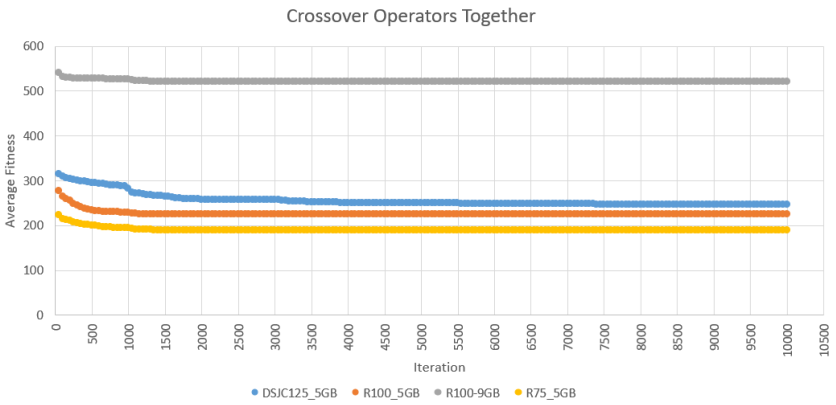


Figure 4. 3 Experiment on Crossover Operators Working Together

When we compared the results for crossover operators, we observed that improvement in fitness function is not observed after 8000 iterations for most of the cases. Therefore, we decided to limit the total number of iterations to 8000 in experiments conducted for mutation operators. When two crossover operators are applied together, sharper decrease in value of the fitness function can be observed when compared with the cases in which only one of the crossover operator is applied.

5. CONCLUSION

In our study, we tried to solve WVCP by using an EA based algorithm. For this purpose, we offered two pool based crossover operators and a mutation operator. We tested our algorithm by using 46 DIMACS benchmark instances that are used in the literature. We have compared our best values with best values of other algorithms which are AFISA, MWSS and 2-Phase.

Our algorithm was able to find the best known values for 37 benchmark instances. We found new best values for 2 benchmark instances which are DSJC125_5GB and R100_5GB. Therefore, our algorithm obtained better results than the results obtained by AFISA, MWSS and 2-Phase. On the other hand, our algorithm was not able to find the best known values for remaining 7 instances. We have found one more greater value than the best known values of these instances.

6. FUTURE WORK

In our algorithm, initial population is created randomly which results in a starting point far away from the best known solution. Therefore, our algorithm can be tested with different techniques to construct the initial population.

In addition to creation technique of initial population, another improvement may be obtained by changing the replacement method of individuals in the population. Only offspring with better fitness value than fitness value of at least one of its parents is let to participate in the population. If we change this constraint, variety in the population can be increased and better solutions may be obtained as a result of variety.

REFERENCES

[1] Jensen, T. R., Toft, B. (1995). Graph coloring problems. John Wiley & Sons.

[2] Garey, M. R., Johnson, D. S. (1979). Computers and Intractability: A Guide to the Theory of NP-Completeness. San Francisco: Freeman.

[3] Malaguti, E. (2009). The vertex coloring problem and its generalizations. 4OR: A Quarterly Journal of Operations Research, 7(1):101-104

[4] Bäck, T. (1996). Evolutionary algorithms in theory and practice: evolution strategies, evolutionary programming, genetic algorithms. Oxford university press.

[5] Bäck, T., Fogel, D. B., Michalewicz, Z. (1997). Handbook of evolutionary computation. CRC Press.

[6] Eiben, A. E., Smith, J. E. (2003). Introduction to evolutionary computing (Vol. 53, p. 18). Berlin: springer.

[7] Holland, J. H. (1975). Adaptation in natural and artificial systems. Ann Arbor: University of Michigan Press.

[8] Goldberg, D. E. (1989). Genetic algorithms in search. Optimization, and Machine Learning.

[9] Mitchell, M. (1998). An introduction to genetic algorithms. MIT press.

[10] Sungu, G., & Boz, B. (2015, July). An evolutionary algorithm for weighted graph coloring problem. In Proceedings of the Companion Publication of the 2015 Annual Conference on Genetic and Evolutionary Computation (pp. 1233-1236).

[11] Sun, W., Hao, J. K., Lai, X., Wu, Q. (2018). Adaptive feasible and infeasible tabu search for weighted vertex coloring. Information Sciences, 466, 203-219

[12] Cornaz, D., Furini, F., Malaguti, E. (2017). Solving vertex coloring problems as maximum weight stable set problems. Discrete Applied Mathematics, 217,151-162.

[13] Malaguti, E., Monaci, M., Toth, P. (2009). Models and heuristic algorithms for a weighted vertex coloring problem. Journal of Heuristics, 15(5), 503-526.

[14] Topcuoglu, H. R., Demiroz, B., Kandemir, M. (2007). Solving the register allocation problem for embedded systems using a hybrid evolutionary algorithm. IEEE Transactions on Evolutionary Computation, 11(5), 620-634.

[15] Chaitin, G. J., Auslander, M. A., Chandra, A. K., Cocke, J., Hopkins, M. E., Markstein, P. W. (1981). Register allocation via coloring. Computer languages, 6(1), 47-57.

[16] Briggs, P., Cooper, K. D., Kennedy, K., Torczon, L. (1989). Coloring heuristics for register allocation. In PLDI (Vol. 89, pp. 275-284).

[17] Briggs, P., Cooper, K. D., Torczon, L. (1994). Improvements to graph coloring register allocation. ACM Transactions on Programming Languages and Systems (TOPLAS), 16(3), 428-455.

[18] Chaitin, G. J. (1982). Register allocation and spilling via graph coloring. ACM SIGPLAN, pp. 98-105.

[19] Chow, F. C., & Hennessy, J. L. (1990). The priority-based coloring approach to register allocation. ACM Transactions on Programming Languages and Systems (TOPLAS), 12(4), 501-536.

[20] Leighton, F. T. (1979). A graph coloring algorithm for large scheduling problems. Journal of research of the national bureau of standards, 84(6), 489-506.

[21] de Werra, D. (1985). An introduction to timetabling. European journal of operational research, 19(2), 151-162.

[22] Gamst, A. (1986). Some lower bounds for a class of frequency assignment problems. IEEE transactions on vehicular technology, 35(1), 8-14.

[23] Furini, F., & Malaguti, E. (2012). Exact weighted vertex coloring via branch-and-price. Discrete Optimization, 9(2), 130-136.

[24] Brélaz, D. (1979). New methods to color the vertices of a graph. Communications of the ACM, 22(4), 251-256.

[25] Glover, F. (1989). Tabu search—part I. ORSA Journal on computing, 1(3), 190-206.

[26] Glover, F. (1990). Tabu search—part II. ORSA Journal on computing, 2(1), 4-32.

[27] Glover, F., Laguna, M. (1998). Tabu search. In Handbook of combinatorial optimization (pp. 2093-2229). Springer, Boston, MA.

[28] Hertz, A., & de Werra, D. (1987). Using tabu search techniques for graph coloring. Computing, 39(4), 345-351.

[29] Prais M., Ribeiro C. C. (2000). Reactive GRASP: An application to a matrix decomposition problem in TDMA traffic assignment. INFORMS Journal on Computing, 12(3):164-176.

[30] Caprara, A., Fischetti, M., Toth, P. (1999). A heuristic method for the set covering problem. Operations research, 47(5), 730-743.

[31] Xu, Y., Chen, Y., Zhang, P., Goebel, R. (2019). Approximation algorithms for vertex happiness. Journal of the Operations Research Society of China, 7(3), 429-448.

[32] Pardalos, P. M., Mavridou, T., & Xue, J. (1998). The graph coloring problem: A bibliographic survey. In Handbook of combinatorial optimization (pp. 1077-1141). Springer, Boston, MA.

[33] Welsh, D. J., & Powell, M. B. (1967). An upper bound for the chromatic number of a graph and its application to timetabling problems. The Computer Journal, 10(1), 85-86.

[34] Matula, D. W., Marble, G., & Isaacson, J. D. (1972). Graph coloring algorithms. In Graph theory and computing (pp. 109-122). Academic press.

[35] Mehrotra, A., & Trick, M. A. (1996). A column generation approach for graph coloring. informs Journal on Computing, 8(4), 344-354.

[36] Johnson, D. S., & Trick, M. A. (Eds.). (1996). Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge. American Mathematical Society

[37] Galinier, P., & Hao, J. K. (1999). Hybrid evolutionary algorithms for graph coloring. Journal of combinatorial optimization, 3(4), 379-397.

[38] Rechenberg, I. (1973). Evolutionsstrategie: Optimierung technischer Systeme nach Prinzipien der biologischen Evolution. Fromman-Hozlboog, Stuttgart

[39] Koza, J. R. (1992). Genetic programming: on the programming of computers by means of natural selection. MIT press.

[40] Fogel, L. J., Owens, A. J., & Walsh, M. J. (1966). Artificial intelligence through simulated evolution.

[41] Coello, C. A. C. (2005, January). An introduction to evolutionary algorithms and their applications. In International Symposium and School on Advanced Distributed Systems (pp. 425-442). Springer, Berlin, Heidelberg.

RESUME

SERAP KORKMAZ

Marmara University,
Computer Engineering Department, Faculty of Engineering,
Goztepe Campus, Kadikoy, Istanbul, Turkey
Phone: +90-216-777-34-58
e-mail: serap.korkmaz@marmara.edu.tr

EDUCATION

M.S., Computer Science Marmara University, Faculty of Engineering, Istanbul, Turkey, Ongoing.

Thesis Topic: Solving Graph Coloring Problem by Using an Evolutionary Algorithm,

Advisor: Assist. Prof. Betül DEMİRÖZ BOZ.

B.S., Computer Science Marmara University, Istanbul, Turkey, 2014

WORK EXPERIENCE

2015-..., Research Assistant, Marmara University, Computer Engineering Department, Istanbul, Turkey

RESEARCH INTERESTS

Evolutionary Algorithms, Graph Coloring Problems, Parallel Programming, Particle Swarm Optimization

FOREIGN LANGUAGES

English