IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received June 25, 2020, accepted July 8, 2020, date of publication July 13, 2020, date of current version July 22, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.3008886

Integrated Crossover Based Evolutionary Algorithm for Coloring Vertex-Weighted Graphs

BETÜL BOZ[1], (Member, IEEE), AND GIZEM SÜNGÜ[2]
[1]Computer Engineering Department, Marmara University, 34722 Istanbul, Turkey
[2]Institute of Information Technologies, Gebze Technical University, 41400 Kocaeli, Turkey

Corresponding author: Betül Boz (betul.demiroz@marmara.edu.tr)

This work was supported by the Marmara University Bilimsel Araştırma Projeleri (BAP) Project under Grant FEN-A-120514-0158.

ABSTRACT Graph coloring is one of the main optimization problems widely studied in the literature. In this study, we propose a novel evolutionary algorithm called Integrated Crossover Based Evolutionary Algorithm with its unique crossover operator and local search technique for coloring vertex-weighted graphs. The integrated crossover operator targets to use the domain-specific information in the individuals and the local search technique aims to explore neighborhood solutions using weighted-swap operations. The performance of the proposed work is evaluated on synthetic benchmarks and DIMACS instances by comparing it with leading evolutionary algorithms from the literature. The experimental study indicates that our algorithm outperforms the related work in 71% of the test cases and achieves the same result in 17% of the test cases provided in the synthetic benchmarks. The experiments performed on DIMACS benchmarks denote that our algorithm finds the best number of colors in 70 out of 73 graphs, so the proposed work is very successful in coloring vertex-weighted graphs within a reasonable amount of time.

INDEX TERMS Graph coloring problem, vertex-weighted graphs, crossover operator, evolutionary algorithms, k-coloring.

I. INTRODUCTION

The theory of graph coloring deals with partitioning a set of vertices to disjoint color classes under the condition that no vertices sharing an edge can be assigned to the same class. The objective of the classical graph coloring problem is to determine the smallest color value k to obtain a legal solution. The graph *k-coloring* problem (k-GCP) seeks to find a feasible coloring of a graph for a given value of k. If the solution is conflict-free, then a legal k-coloring can be obtained. Graph coloring problem can be solved using a series of k-coloring problems. Starting with a sufficiently large k value, the k value can be decreased each time a legal coloring can be found. This process is repeated until an illegal solution is obtained. The objective of k-GCP is to minimize the number of conflicting edges for a fixed k value.

There are numerous graph coloring problems with different objectives. The equitable coloring problem [1] involves a legal coloring by assigning vertices to k independent color classes where the number of vertices in these classes can differ by at most one, whereas the minimum sum coloring problem has the objective of minimizing the sum of colors assigned to the vertices. Weights can also be added to the vertices in graph coloring problems [2]. Weighted vertex coloring problem aims to obtain a legal k-coloring with the objective of minimizing the sum of the costs of its color classes. The cost of a color class is determined by the vertex having maximum weight in the class.

Graph coloring problem is commonly used to model many real-world problems such as scheduling, resource allocation and register allocation [3]–[5]. Most of these problems have limited number of resources. Since k-coloring considers a fixed color value k, this value can refer to the number of resources available in the system, thus k-coloring can be used for the solution of these problems. In most of the cases, the number of colors, k, may not be sufficient to obtain a legal coloring, so some vertices will be uncolored. The importance of the vertices may not be equal, thus a weight value can be used to denote their importance. Our proposed work considers k-coloring (k-GCP) problem using a vertex-weighted graph with the objective of minimizing the total weight of the uncolored vertices for a given k value. Vertex weighted k-coloring problem uses an undirected and weighted graph $G = (V, E, w)$, where V denotes the set of vertices,

The associate editor coordinating the review of this manuscript and approving it for publication was Emanuele Crisostomi.

E indicates the set of edges and w is the set of weight values of the vertices in V which emphasize their order of importance. The objective of k-GCP is to color the vertices in V using a predefined number of colors. If the given number of colors cannot color all of the vertices, some vertices will be uncolored. The uncolored vertices are defined as *conflicting vertices*. The fitness function $f(k)$ is equal to the total of the uncolored vertex weights when a predefined number of k colors are used. The objective of k-GCP is to minimize the fitness value $f(k)$.

The graph coloring problem is proved to be an NP-Complete problem [6] and many heuristics for graph coloring problem [7]–[11], equitable coloring problem [12], [13], k-coloring problem [14], [15], minimum sum coloring problem [16] and weighted vertex coloring problem [17] have been proposed in the literature. In this study, we propose a hybrid evolutionary algorithm [18] for the vertex-weighted k-coloring problem. Our algorithm is called Integrated Crossover Based Evolutionary Algorithm (InCEA) with its novel crossover operator and local search technique. The Integrated crossover operator groups maximum number of non-conflicting vertices to the color classes of the offspring with the successful usage of the problem-specific information in the parents. Two randomly selected color classes from the parents are merged to form each color class of the offspring incrementally. The conflicting vertices are thrown to the pool and every time a new color class of the offspring is created, the vertices in the pool try their chance to find a non-conflicting color class. If there are vertices available in the pool at the end of crossover operator, the local search technique tries to place these vertices to one of the color classes of the offspring using a weighted-swap operation such that the fitness value is minimized.

In our experimental study, we compared our algorithm InCEA with related work from the literature. The results obtained from synthetically generated benchmarks and DIMACS instances indicate that InCEA outperforms related work in most of test cases with respect to the fitness values and computation times. The performance of the algorithms on 73 DIMACS instances which are challenge graphs for graph coloring problem, are also provided in this study. Since the minimum number of colors needed to color the graphs for these benchmarks have not been found in the literature, this paper also reports the minimum number of colors used to color these instances. As indicated in the experimental study, our evolutionary algorithm can obtain the best results very fast as compared to the algorithms in the literature in most of the test cases provided.

The main contributions of this work can be listed as follows:

- *The Integrated Crossover Operator* targets to use the problem-specific information in the individuals with the help of a pool and a search back operation.
- *Weighted-Swap Operation in the Local Search Technique* aims to explore neighbor solutions and increase the chance to reach to the global optimum.
- No additional computation for the local search technique or fitness calculation is needed since the pool already holds the uncolored vertex(es), so the algorithm gets rid of the burden of exhaustive searches.

In the rest of the paper, we first present the related studies and problem definition of k-coloring problem in Section II and III, respectively. In Section IV, we explain the components of our proposed work in detail. Next, we compare and discuss the performance of our algorithm with the algorithms from the literature on various test instances in Section V. Finally, we summarize our proposed work and give suggestions for possible future directions in Section VI.

II. RELATED WORK

Many real-world problems such as scheduling, register allocation and resource allocation can be represented with the graph coloring problem, where the vertices denote the objects and the edges indicate the constraints. In all of these problems, there are limited number of resources (colors), so some vertices will be uncolored. Weights can be added to the vertices to denote their importance, and a vertex-weighted graph can be used to model these problems.

In register allocation problem [19], the aim is to assign maximum number of variables to minimum number of registers so that the variables can be accessed very fast by the CPU. The problem can be represented using an undirected graph, where the variables are denoted by the vertices, the interferences between variables are denoted by an edge. The objective of the problem is to minimize *the number of registers used* (colors) and to *select the variables that are accessed less frequently*, which can directly be applied to the selection of uncolored vertices and calculation of the fitness function in k-GCP [20].

Resource allocation problem can also be solved using k-GCP. One of the most important resource allocation problems in computer networks [21]–[25] is the bandwidth allocation in wireless networks. In this problem, the network with its sensors can be represented as an undirected graph and vertices, respectively. If the distance between any two sensors is greater than a predefined threshold, these sensors cannot share the same band, so their representative vertices are connected by an edge. The objective of the problem is to allocate *minimum number of bands* (colors) to all *sensors*. Also the weight of the vertices denotes *the quality of thePsensors*.

Scheduling is a challenging problem for education [26], computation [27], manufactory [28] and communication [29], [30]. Generally, the problems focus on an element such as a student, a production, an employee or a job with their various attributes. The elements are denoted as vertices and the conflicts between these elements are represented with edges in the graph. The importance of the elements are denoted with the vertex weights. Time, machines, resources, vehicles are used as colors and the objective function can be maximization of time efficiency, the utilization of the resources etc. Therefore, the GCP is a good fit for scheduling problems [31].

Once the real-world problems are modeled with GCP, many exact [32]–[34] and heuristic approaches [35]–[38] can be used due to its NP-Hard complexity [6], [39], [40]. Although the proposed exact algorithms for solving graph coloring problem [41] can find the best solutions for small instances, they are expensive in terms of memory and time consumption for large instances [42].

Evolutionary algorithm is one of the heuristic approaches for the graph coloring problem [2]. Evolutionary algorithms mimic the natural process with its crossover and mutation operators. The algorithms concern a population which keeps a predefined number of candidate solutions which are denoted as individuals for a given vertex-weighted graph [43].

In the literature, there are evolutionary algorithms that solve the real-world problems which are encoded into k-GCP such as Hybrid Evolutionary Algorithm (HEA) [44] and Cost-oriented Memetic Algorithm (COMA) [45].

HEA and COMA create their initial population applying three metrics which are proposed in [44]. The details of these metrics are given in Section IV-A. Once the initial populations are created, both HEA and COMA apply their crossover operators to improve their individuals. The algorithms select two parents from their population and combine the color classes of these parents to create an offspring. The selection of the color classes differ in these algorithms. HEA uses conflict-free partition crossover (CFPX) which selects the color classes having the maximum conflict-free vertex subset. On the other hand, COMA selects the color class having the conflict-free subset with maximum weight and its crossover operator is named as cost-oriented crossover operator (COPX).

The offspring created by CFPX or COPX does not guarantee to produce a conflict-free solution. HEA and COMA apply the same local search operator which is proposed in [44]. They select one of the conflicting vertices from a color class of the offspring using the degree and weight values of the vertices. Both algorithms visit all color classes to place this conflicting vertex, and it is assigned to the color class yielding the maximum decrease in the fitness value. The number of iterations in this search phase is limited to 10% of the total number of conflicts in HEA, whereas COMA has no such limitation and visits all color classes for each conflicting vertex in the offspring. Conflict-based and spill-cost-based techniques are introduced in HEA, and the smallest fitness value obtained from these two approaches is selected as the fitness value of the offspring. Whereas, COMA uses conflict-based, cost-based and metric-based techniques to calculate the fitness value of the offspring.

III. PROBLEM STATEMENT

Let $G(V, E, w)$ be an undirected vertex-weighted graph, where V and E are the sets of vertices and edges, and w is the set of weight values of the vertices $\in V$. If V has n vertices, the cardinality of w, $|w|$, is also n and E can have at most $\frac{n\times(n-1)}{2}$ edges, where $0 \leq |E| \leq \frac{n\times(n-1)}{2}$.

Each vertex $v \in V$ belongs to a color class C_i, which is one of the disjoint independent sets of $C = \{C_1, C_2, \ldots, C_k\}$, $1 \leq k \leq n$.

If $u \in V$ is adjacent to $v \in V$, then there is an edge $\{u, v\} \in E$ and u and v cannot be in the same color (1). This is called *legal* or *k-feasible coloring*.

$$\forall v, \quad u \in C_i, \ \{u, v\} \notin E, \ i = 1, 2, \ldots, k \tag{1}$$

For a given color class value k, the aim of the k-coloring problem is to provide a color class for each vertex v obeying (1) and obtain a feasible coloring. If v cannot be assigned to a color class, then v is defined as a *conflicting vertex* and becomes *uncolored*. In this case, the solution is infeasible and the objective of the k-coloring problem is to minimize the sum of weights of the conflicting vertices using $f(k)$ (2).

$$\text{minimize } f(k) = \sum w(v), \quad v \notin C \tag{2}$$

IV. INTEGRATED CROSSOVER BASED EVOLUTIONARY ALGORITHM

The general procedure of our GA-based approach is given in Algorithm 1. At each generation of the algorithm, it performs the Integrated Crossover (InCX), which is presented in Algorithm 2. As part of the crossover operator, a search back operation given in Algorithm 3 is performed. Based on the output of the crossover operator, which is a single offspring, our proposed work targets to improve the offspring with the local search technique presented in Algorithm 4.

Algorithm 1 General Algorithm

```
Input: Graph G, population size p, number of color
       classes k, number of iterations t
Output: Individual S* with the best fitness value
Pop = {S_1, ..., S_p} ← InitialPopulation()
for i ← 1 to t do
    Randomly select individuals S_x and S_y from Pop
    where x ≠ y
    S_0, Pool ← CrossoverOperation(G, k, S_x, S_y)
    if Pool ≠ ∅ then
        S_0 ← LocalSearch(k, S_0, Pool)
    end
    Pop ← UpdatePopulation(S_0, S_x, S_y)
end
```

A. INDIVIDUAL REPRESENTATION AND INITIAL POPULATION GENERATION

In the proposed algorithm, each individual S_i in the population is represented with the partition method [46] where $S_i = \{C_1, C_2, C_3, \ldots, C_k\}$ and each color class C_j contains a group of non-conflicting vertices by considering a total of k color classes.

The initial population contains a predefined number of candidate solutions generated using the spill degree metric [44], which orders the vertices in three different ways. The spill

Algorithm 2 Integrated Crossover Operator

Input: Graph G, number of color classes k, first parent $S_1 = \{C_0^1, C_1^1, \ldots, C_{k-1}^1\}$, second parent $S_2 = \{C_0^2, C_1^2, \ldots, C_{k-1}^2\}$
Output: An offspring $S_0 = \{C_0, C_1, \ldots, C_{k-1}\}$, an updated pool P

```
Create an empty pool P := ∅
for i ← 0 to k-1 do
    Mark C_i^1 and C_i^2 as unselected
end
for i ← 0 to k-1 do
    Set i^th color class of S_0 C_i: C_i := ∅
    Select an unselected color class C_x^1 from S_1
    Select an unselected color class C_y^2 from S_2
    Mark C_x^1 and C_y^2 as selected
    for each unassigned vertex v_ua in C_x^1 and C_y^2 do
        Put v_ua into C_i: C_i ← C_i U v_ua
        Mark v_ua as assigned vertex v_a
    end
    if P ≠ ∅ then
        Put ∃ vertex in P v_p into C_i: C_i ← C_i U v_p
        Remove v_p from P: P ← P / v_p
    end
    while C_i is not conflict-free do
        Calculate the maximum conflicting vertex as v_max in G
        Throw v_max into P: P ← P U v_max
        Remove v_max from C_i: C_i ← C_i / v_max
    end
    if i ≥ 1 then
        SearchBackOperation(i, S_0, P, G)
    end
end
```

Algorithm 3 Search Back Operation

Input: Number of combinations that are currently completed i, the offspring $S_0 = \{C_0, \ldots, C_{i-1}\}$, the pool P, the graph G
Output: S_0, P

```
for each vertex v_p in P do
    for j ← 0 to i-1 do
        Set a variable CF := true
        for each vertex v in C_j do
            if Conflict(v, v_p) then
                CF ← false
                break
            end
        end
        if CF then
            Remove v_p from P: P ← P / v_p
            Put v_p into C_j: C_j ← C_j U v_p
            break
        end
    end
end
```

degree metric uses the weights of the vertices, $w(v_i)$, and the degree of the vertices denoted as $d(v_i)$. For diversifying the population, spill degrees of 40%, 40%, 20% of all vertices are set using (3a, 3b and 3c), respectively.

$$S_1(v_i) = w(v_i) \times d(v_i) \tag{3a}$$
$$S_2(v_i) = w(v_i) \times d(v_i)^2 \tag{3b}$$
$$S_3(v_i) = w(v_i) \tag{3c}$$

The vertices are ranked in decreasing order of their spill degrees and they are placed into the color classes of the individuals. For an individual generation, the vertex with the maximum spill degree from the set of unassigned vertices is selected and is placed to a color class. Starting from the first color class, the algorithm finds a color class where the vertex provides a conflict-free set with the other vertices assigned to the same class. The vertex is assigned to the first conflict-free color class. If no such class can be found, then the vertex is placed to a random color class. This process is repeated until all vertices are mapped to a color class.

B. INTEGRATED CROSSOVER OPERATOR

In this work, we propose a novel crossover operator which targets to guide the individuals to reach the global optimum with the problem-specific information in the color classes. The objective of the integrated crossover operator is to combine the color classes of the parents to increase the number of non-conflicting vertices in each color class such that total number of uncolored vertices would be minimized.

The two color classes selected from the parents are combined to increase the chance of finding a larger and better grouping of the non-conflicting vertices. Our integrated crossover operator proposes a pool to keep the conflicting vertices that cannot be placed to the current color class. The offspring will typically have more than one color class, so the vertices thrown to the pool have a chance to be placed into the previously generated color classes. On the other hand, our integrated crossover operator has a search back operation for minimizing the number of vertices remaining in the pool by placing the vertices in the pool to a conflict-free color class.

The previous studies in the literature selected the color class with the maximum conflict-free set from one of the parents and this set is directly copied to the offspring. They did not consider to combine the problem-specific information in both of the color classes due to conflicts.

1) POOL BASED CROSSOVER OPERATOR

The integrated crossover operator begins with the random selection of two parent configurations where the first parent and the second parent are represented as $S_1 = \{C_0^1, C_1^1, \ldots, C_{k-1}^1\}$ and $S_2 = \{C_0^2, C_1^2, \ldots, C_{k-1}^2\}$, respectively in the crossover algorithm (see Algorithm 2). The operator builds an offspring $S_0 = \{C_0, C_1, \ldots, C_{k-1}\}$ and

Algorithm 4 Local Search Technique

Input: Number of color classes k, the offspring $S_0 = C_0, \ldots, C_{k-1}$, the pool P
Output: S_0
Set total := 0, $V_{\text{tabu}} := \emptyset$
Sort $\exists\ v_p$ in P in descending order according to $\text{w}(v_p)$
for *each vertex* v_p *in* P **do**
  Set index := -1 // index of the color class with the minimum weight value
  Set min := $\text{w}(v_p)$
  Set $V_{\text{removed}} := \emptyset$
  for $i \leftarrow 0$ **to** *k-1* **do**
    Set sum := 0, $V_{\text{conflict}} := \emptyset$
    for *each vertex v in* C_i **do**
      if *Conflict(v,* v_p*)* **then**
        sum $\leftarrow$ sum + $\text{w}(v)$
        $V_{\text{conflict}} \leftarrow V_{\text{conflict}} \cup v$
      end
    end
    if *min* > *sum* **then**
      min $\leftarrow$ sum
      index $\leftarrow i$
      $V_{\text{removed}} \leftarrow V_{\text{conflict}}$
    end
  end
  if *index* $\neq -1$ **then**
    $P \leftarrow P \cup V_{\text{remove}} / v_p$
    $C_{\text{index}} \leftarrow C_{\text{index}} \cup v_p / V_{\text{remove}}$
  else
    $V_{\text{tabu}} \leftarrow V_{\text{tabu}} \cup v_p$
    $P \leftarrow P / v_p$
    total $\leftarrow$ total + $\text{w}(v_p)$
  end
end
for *each vertex v in* V_{tabu} **do**
  Randomly assign v to C_x
end

prepares a pool P which will also be used by the local search technique.

The integrated crossover operator will iteratively create the offspring and at each iteration one color class of the offspring represented as C_i will be generated using two randomly selected partitions C_x^1 and C_y^2 from both parents. The vertices in these partitions are grouped to form the color class of the offspring with the conflict-free vertices. If there are any conflicting vertices v_{ua} that can not be placed to the current color class, the algorithm tries to assign these vertices to previously generated color classes of the offspring S_0 using search back operation which is described in detail in the following subsection. If no such color class can be found, then they are thrown to the *pool* to be combined with the vertices selected in the following iterations.

2) SEARCH BACK OPERATION

In our initial work, Pool Based Evolutionary Algorithm (PBEA) [47], we have proposed a primitive pool to store unassigned vertices. Even if some vertices in the pool can be placed to color classes of the offspring already generated in the previous iterations, no intelligent mechanism was proposed to detect them. As a result, the maintenance of the pool would became very hard as new conflicting vertices are added at each iteration in our previous study.

In this study we propose the search back operation in conjunction with the pool as part of the integrated crossover operator. The search back operation targets to remove vertices that have no conflicts with the previously generated color classes from the pool. As a result, it increases the number of non-conflicting vertices in each color class and decreases the number of vertices available in the pool.

In order to illustrate the execution of our integrated crossover operator (InCX), we consider an example weighted graph and two parent configurations in Fig. 1 and Fig. 2, respectively. Note that the same graph and parents are presented in a previous work [44].

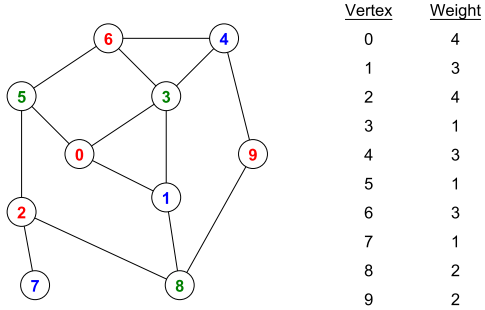


FIGURE 1. An example weighted graph [44].

Starting from step 0, the second color class C_1^1 of the first parent S_1 and the third color class C_2^2 of the second parent S_2 are selected randomly. The vertices 2, 7, 8 in C_1^1 and the vertices 0, 1, 9 in C_2^2 are combined and put into the first color class C_0 of the offspring S_0. These vertices are marked as *assigned* and they are removed from the color classes of the parents where they become invisible vertices for the next steps. The conflicts between the vertices in C_0 are calculated according to the graph in Fig. 1. The maximum conflicting vertex v_{max} with 3 conflicts is vertex 8 so it is directly thrown into the pool P. Vertex 8 has three edges with vertex 1, vertex 2 and vertex 9. After vertex 8 is removed from C_0, the conflicts of vertex 1, vertex 2 and vertex 9 are decreased by 1. The vertices 2, 7 and 1 have the maximum number of conflicts, so their weight values (given in Fig. 1) are compared. Since vertex 7 has the lowest weight value of 1, it is selected as v_{max} and it is thrown into P and removed from C_0. The only conflicting vertices remaining in C_0 are vertex 0 and vertex 1, where vertex 1 is thrown into P and

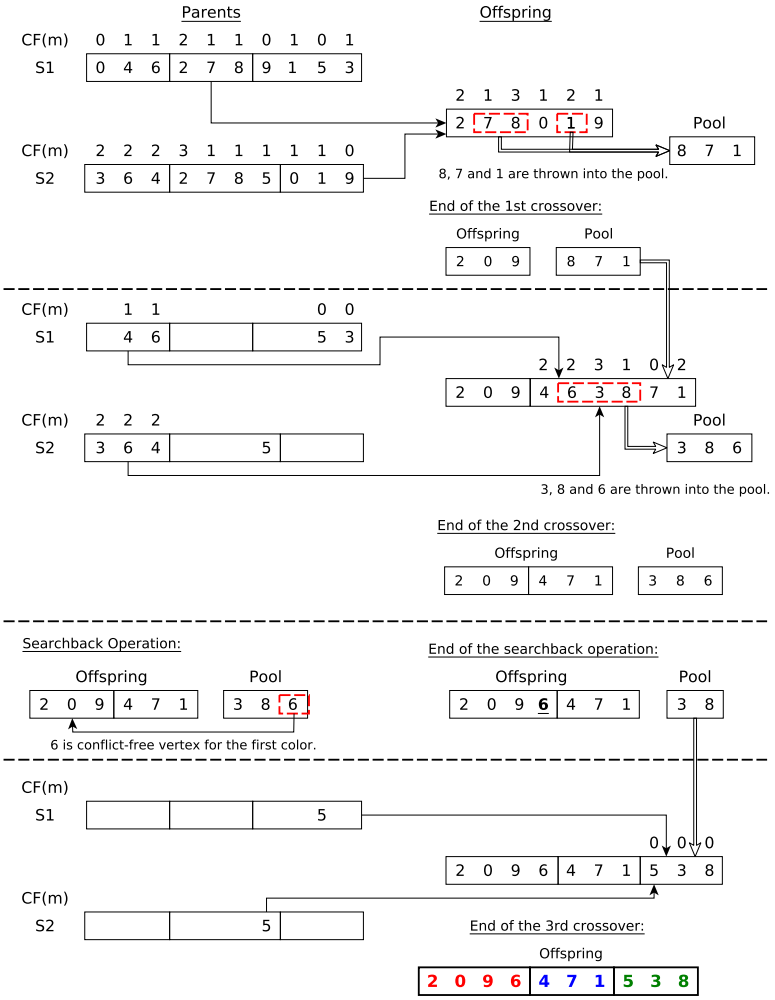


FIGURE 2. InCX operator applied on two parent configurations to the graph given in Fig. 1.

is removed from C_0 since it has a lower weight. C_0 becomes conflict-free and the first crossover iteration is finished.

In the next iteration, the first color classes C_0^1 and C_0^2 from both parents are selected randomly. The *unassigned* vertices 3, 4 and 6 in $(C_0^1 \cup C_0^2)$ are put into the second color class C_1 of S_0 and marked as *assigned*. The vertices 8, 7 and 1 in P are also combined with the vertices in C_1. The conflicts of the vertices are calculated and vertices 3, 8 and 6 are thrown into P to obtain a conflict-free color class C_1. Since the offspring S_0 has a previously generated color class C_0, the unassigned vertices in the pool have a chance to be placed into C_0. The search back operation tries to place vertices in P to C_0. Vertex 3 has conflicts with vertex 0, vertex 8 has conflicts with vertices 2 and 9, so they are kept in P. However, vertex 6 has no conflicts with the vertices in C_0, so it is mapped to C_0. In the final iteration, there are two remaining color classes that have not been selected before as C_2^1 and C_1^2. In both color classes, there is only one *unassigned* vertex which is vertex 5 so it is combined with the vertices 3 and 8 available in P and placed into the third color class C_2 of S_0. According to the given graph in Fig. 1, a conflict free set is obtained for C_2. Since predefined number of k color classes (where $k = 3$) are created for S_0, the crossover operation is finished.

At the end of the integrated crossover operator, an offspring S_0 with 3 color classes and a pool P are obtained. When P contains unassigned vertices, it means the given graph cannot be colored with k colors and the proposed local search technique explained in Section IV-B2 will be applied to S_0 and P. Otherwise, the proposed algorithm has successfully colored all vertices so there is no need to apply the local search technique. An example scenario is given in Fig. 2.

For the example weighted graph given in Fig. 1, the output of our integrated crossover operator (InCX) and three outputs of crossover operators presented in the literature are given in Fig. 3, which are Conflict-free Partition Crossover (CFPX) [44], Cost-oriented Crossover (COPX) [45], Pool Based Crossover (PBC) [47]. The offsprings generated by CFPX, COPX and PBC still have uncolored vertices. CFPX and PBC cannot color vertex 6 whose weight is 3. The COPX

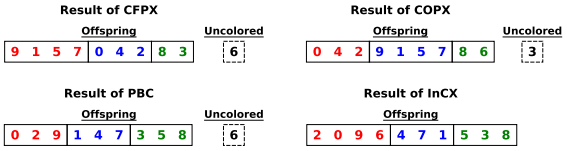


FIGURE 3. The results obtained from four crossover operators which are applied to the same parent configurations.

was not able to color vertex 3 whose weight is 1. However, InCX manages to color all the vertices without applying the local search technique.

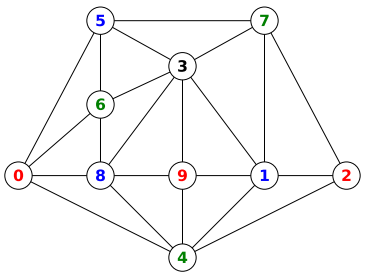
Both CFPX and COPX always select the color classes having maximum conflict-free subsets, whereas PBC and InCX select color classes from both parents in a random manner, so the solutions can be obtained from a variety of combinations. In Fig. 2, PBC and InCX can combine color classes of the parents in 36 different ways and get 36 solutions from these combinations. PBEA can find a conflict-free solution from 50% of these combinations, 22% of the conflict-free solutions are found after PBC and 28% of them are found after its local search operator. Whereas, InCEA can obtain a conflict-free solution from 58% of these combinations and 55% of these conflict-free solutions are found after the InCX operator. These percentages show that our new crossover operator increases the chance of finding a conflict-free solution.

C. LOCAL SEARCH TECHNIQUE

The aim of the InCX operator is to obtain the largest non-conflicting vertex groups in each color class of the offspring. While throwing the conflicting vertices into the pool, the weight values of the vertices are only considered where there is a tie break, i.e. two or more vertices have the same number of conflicts. At the end of our integrated crossover operator, if there are still vertices in the pool, these vertices have conflicts with at least one of the vertices in each color class of the offspring and there may be no conflict free solutions.

The objective of our algorithm is to minimize the total weight of the vertices that cannot be colored. Since the InCX operator does not consider the weights of the vertices, the local search technique targets to decrease the sum of weights of uncolored vertices considering neighborhood solutions. At the end of the crossover operator, the uncolored vertices are present in the pool, so the local search technique tries to place these vertices to the color classes using swap operations.

Inspired by the traditional mutation operator SWAP [48] which exchanges the positions of two selected genes in the order-based gene representation [49], we propose a new local search technique which is called Weighted-Swap (*W-SWAP*). W-SWAP includes searching and swapping the vertices between the pool and the color classes. *If the total weight of the vertex(es) to be swapped from one of the color classes is less than the weight of the vertex in the pool*, then a swap operation is performed. At the end of the local search,



Vertex	Weight
0	4
1	3
2	4
3	1
4	3
5	1
6	3
7	1
8	2
9	2

FIGURE 4. An example weighted graph to be used in the local search technique.

the sum of weights of uncolored vertices would be minimized and there is also a chance to find a conflict-free solution.

At the beginning of the local search technique, the vertices in P are sorted in descending order of their weight values to place the vertex with the highest weight first, if possible. For each v_p in P, the algorithm calculates the sum of weights of the vertex(es) conflicting with v_p in all k classes of S_0 and finds the minimum sum. If this value is less than $w(v_p)$, than a swap operation is performed. The vertex(es) in the color class is thrown to P and v_p is placed to the color class. If no such color class exists, v_p is removed from P and put into a tabu list V_{tabu} that keeps the uncolored vertices for S_0. The local search continues until each vertex v_p in P are processed and P becomes empty. All of the uncolored vertex(es) are present in V_{tabu} so this list can be used in Fitness Calculation of the offspring, which is described in the next subsection. Finally the vertices in V_{tabu} will be assigned to color classes randomly.

Fig. 5 demonstrates the proposed local search technique using the graph given in Fig. 4. Assume that the integrated crossover operator is applied on two parent configurations and an offspring having a non-empty pool is obtained. P has vertex 9 so the local search targets to place this vertex to one of the color classes to minimize the sum of weights of uncolored vertices. In C_0, vertex 9 has a conflict with vertex 3 having a weight of 1 which is less than weight of vertex 9, so vertex 3 is stored in V_{spilled}, *min* value is set as 1. In C_1 the vertices 1 and 8 have conflicts with vertex 9 and their total weight value is 5 which is higher than *min* so C_1 is not suitable for vertex 9. Finally, in C_2 vertex 4 and vertex 9 have conflicts and the weight of vertex 4 is 3 which is again higher than *min* value. All color classes are visited and vertex 9 is removed from P and is mapped to C_0, whereas vertex 3 in V_{spilled} is removed from C_0 and thrown into P. Since P is not empty, the color classes are visited for vertex 3. Vertex 3 has conflicts in all the color classes and it has the minimum weight, so vertex 3 is placed to V_{tabu}. P becomes empty, so W-SWAP is completed.

D. FITNESS FUNCTION

The fitness function is calculated as the sum of weights of the uncolored vertices, i.e. the vertex(es) present in V_{tabu} after the local search technique is completed. The best scenario

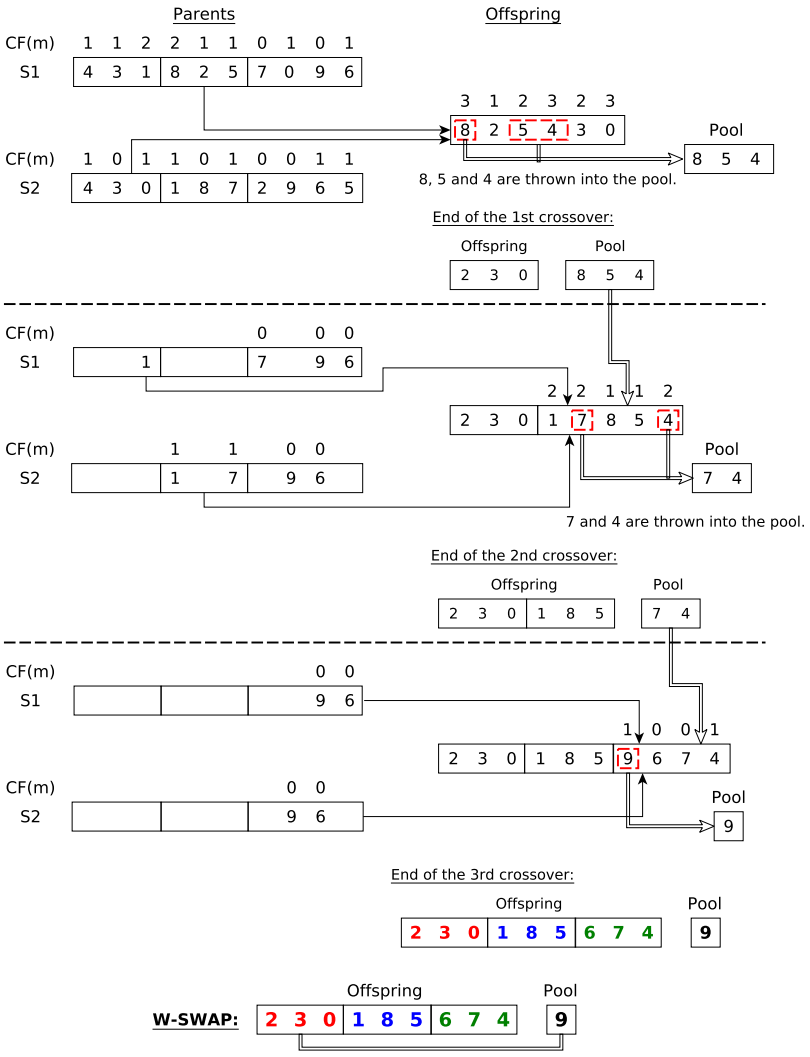


FIGURE 5. InCX operator and W-SWAP technique applied on two parent configurations to the graph given in Fig. 4.

is to have an empty list V_{tabu} which means that all vertices are colored with k colors and the fitness value $f(k)$ is 0. In Fig. 5, vertex 3 becomes *tabu* for all color classes of S_0 and the fitness value of S_0 is assigned to the weight of vertex 3 which is 1.

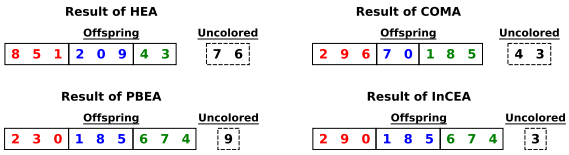


FIGURE 6. The results obtained from four algorithms after their crossover operator and local search technique are applied on the same parent configurations.

Fig. 6 denotes the solutions of the algorithms at the end of the local search operation. Both HEA and COMA cannot assign two vertices having a total weight of 4 to the color classes, so the fitness value of these solutions is 4. PBEA and InCEA have only one uncolored vertex and vertex 3 has the lowest weight, so the proposed work obtains the lowest fitness value of 1, whereas PBEA produces a solution having a fitness value of 2.

If the fitness value of the offspring is better than one or both of the parents, the algorithm chooses the parent with the worst fitness value and swaps the offspring with this parent as a replacement policy.

V. EXPERIMENTAL STUDY

In this section, the performance of the proposed work is compared with HEA [44], COMA [45] and our initial work PBEA [47] using randomly generated graphs and the graphs derived from DIMACS benchmarks. For each graph, the total cost, the number of uncolored vertices and the execution time of the algorithms are the main comparison metrics.

The experiments are performed on a computer with 3.4 GHz Intel Core i7 4770 CPU and 8GB RAM.

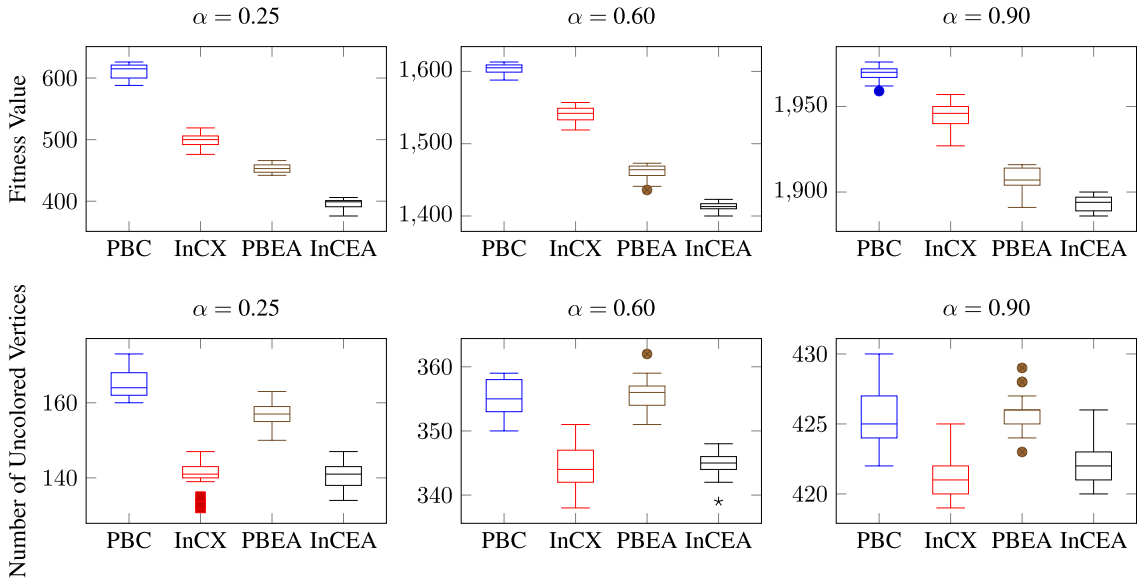


FIGURE 7. Performance comparison of PBC, PBEA, InCX and InCEA using 20 color classes on the randomly generated graphs with 500 vertices and 3 different edge density values.

Implementation of InCEA is in C[1] and is compiled with gcc. Both HEA and COMA were implemented in C++[2] and compiled using g++. The usage of different programming languages for the implementation of the algorithms can be considered as a threat to validity [50] in order to obtain exact execution time of the algorithms.

Both HEA and COMA create one offspring at the end of crossover phase, and try to improve this offspring in the local search phase. So all algorithms generate 1 individual in one iteration. Since all algorithms are evaluated using the same population size and the same number of iterations, the number of fitness evaluations [50] considered by the algorithms is equal except the experiment conducted using a fixed time limit. For this experiment, we have also provided the number of iterations used by all algorithms. For the following experiments, all algorithms take the same parameters and none of them use control parameters or tuners such as REVAC, F-Race or CRS-Tuning.

To evaluate the performance of the operators in our algorithm, the proposed work is compared with our initial study PBEA [47] using three graphs where $n = 500$ and $\alpha = \{0.25, 0.60, 0.90\}$. The performance of the algorithms are measured with or without using their local search techniques. Therefore, PBEA and InCEA executed with their crossover operators only are denoted as PBC and InCX, respectively. The performance of the algorithms are evaluated 20 times on each graph using 20 color classes. Performance comparison of the algorithms with respect to fitness values and number of uncolored vertices is given in Fig. 7. When their fitness values are considered, the proposed crossover operator InCX outperforms PBC with the help of the search back operation but PBEA with its local search technique outperforms InCX. This is expected since InCX does not take into account the weights of the vertices but tries to obtain the largest non-conflicting vertex group. The weights of the uncolored vertices are considered in our proposed local search technique (W-SWAP), which might increase the number of uncolored vertices. On the other hand, with the objective of decreasing the fitness value, our approach has the best fitness value in all of the test cases provided.

When number of uncolored vertices are considered, this number is higher in InCEA as compared to InCX, since the aim of our new local search technique is to decrease the fitness value. Therefore, it can color one uncolored vertex having higher weight by swapping it with more than one vertex having a less total weight. So it may increase the number of uncolored vertices to decrease the fitness value of the solution. InCX successfully obtains the largest non-conflicting vertex group leaving the least number of uncolored vertices. On the other hand, PBC and PBEA spill more vertices because once the color classes are constructed using the crossover operators, their members are fixed which decreases the search space of the solution.

A. RANDOMLY GENERATED GRAPHS

A random graph generator is implemented to produce uniform random graphs using three input parameters which are the number of vertices n, a range of weight values γ for assigning vertex weights, and edge density α to generate the edges connecting the vertices.

The total number of edges in the randomly generated graphs are close to $(n \times (n-1)) \div 2 \times \alpha$. Since the edges are created randomly, some vertices may be isolated for low edge density values, so there is no guarantee to obtain

[1] https://gitlab.com/gizemsungu/incea
[2] http://see.xidian.edu.cn/faculty/jshwu/

TABLE 1. Parameter settings for randomly generated graphs.

Parameters	Description	Values
n	number of vertices	100, 200, 300, 400, 500
α	edge density	0.10, 0.25, 0.45, 0.60, 0.75, 0.90
β	color class density	0.04, 0.08, 0.10, 0.15, 0.20
γ	weight of vertices	[1..10]

TABLE 2. Comparison of the algorithms with respect to fitness values and number of uncolored vertices for 6 different edge density values.

n	α	Fitness Values			# of Uncolored Vertices			Time(s)		
		HEA	COMA	InCEA	HEA	COMA	InCEA	HEA	COMA	InCEA
100	0.10	39.7	39.5	4.9	12.1	12.3	2.2	297	429	2
	0.25	193.6	192.6	35	50.7	51.3	10.6	631	808	2
	0.45	206.8	206.6	90.5	52.9	53.3	26	566	760	3
	0.60	375.3	375.3	136.5	81.5	81.6	37.3	585	863	3
	0.75	374	373.8	198.3	82.1	82.1	51.5	601	793	3
	0.90	404.2	404.2	267.8	88.6	88.6	65.8	465	715	4
200	0.10	50.5	50.1	1.6	16.4	17.1	1	1217	1727	31
	0.25	190	308.1	48.6	55.2	87.4	15.1	2277	3660	33
	0.45	373.5	342.5	144.4	99	94.6	43	3079	3565	37
	0.60	475.9	471.6	231.2	120.9	121.1	66.2	3458	3614	40
	0.75	579.9	758.5	340.4	140	171.2	93.4	3224	2972	41
	0.90	776.2	778.6	480	176.8	177.2	124.5	1951	2687	43
300	0.10	66.5	33	0	21	10.8	0	2898	2358	155
	0.25	450.4	269.5	64.6	120.1	78	19.5	7595	7458	159
	0.45	887.1	818.6	202.1	213	201.7	58	10206	8694	171
	0.60	989.5	943.9	339.2	223.3	216.7	91.8	3672	4688	180
	0.75	1029.1	990.3	498.6	231.9	224.3	131.8	4938	5754	187
	0.90	1055	1044.6	727.1	236.8	235.5	181.3	3863	7640	190
400	0.10	59.5	61.2	0	19.3	20.7	0	3119	4845	470
	0.25	371.9	490.1	75	105.9	140.5	24	5273	11871	479
	0.45	547.3	635.8	242.7	145.4	173	71.7	7830	14583	504
	0.60	796.3	978.1	417	202.9	236.8	116.9	9132	16672	537
	0.75	1191.9	1308.2	627.9	281.4	304.9	170.2	4320	12937	571
	0.90	1356.7	1428.8	924.8	313.4	328	237	4143	11681	589
500	0.10	51	58.3	0	19.1	21.4	0	5827	5474	1208
	0.25	336.4	355.9	79.8	97.8	109.9	28.2	12687	7618	1218
	0.45	776.4	706.1	277	210.8	200.1	85	29993	25367	1268
	0.60	1517.8	1448.4	486.5	356.8	344.8	140.9	27971	24583	1290
	0.75	1954.8	1557.8	745.2	439.9	369.8	207.2	15070	10958	1356
	0.90	1963.4	1797.4	1104.4	443	413.1	291.5	15086	10400	1378

connected graphs. Our graph generator adds an edge between two disconnected vertices that are randomly selected. After the graph is constructed, the weights of the vertices are set randomly using the uniform distribution.

The algorithms are tested using a predefined and fixed number of color classes for the generated graphs. The number of color classes is equal to $(n \times \beta)$, where β is the *color class density*. The range of values for graph generator parameters and color class densities are listed in Table 1.

1) PARAMETER SETTINGS OF THE ALGORITHMS

As part of our experimental study, the performance of the algorithms are measured using various graphs to set the population size of the algorithms and the number of iterations performed by the algorithms. The results denote that the performance of the algorithms does not depend on the initial population size, so it is set to 100 for all algorithms.

TABLE 3. Pairwise comparison of algorithms with respect to α values.

n	α	Pairwise Comparisons						# of Feasible Test Cases		
		InCEA-HEA			InCEA-COMA					
		Better	Equal	Worse	Better	Equal	Worse	HEA	COMA	InCEA
100	0.10	48	74	3	53	70	2	72	69	100
	0.25	120	5	0	121	4	0	5	4	73
	0.45	100	7	18	100	6	19	5	5	25
	0.60	125	0	0	125	0	0	0	0	1
	0.75	125	0	0	125	0	0	0	0	0
	0.90	125	0	0	125	0	0	0	0	0
200	0.10	45	75	5	45	75	5	76	75	100
	0.25	85	30	10	120	5	0	33	5	75
	0.45	104	10	11	100	10	15	10	10	50
	0.60	100	5	20	100	5	20	5	5	25
	0.75	100	0	25	125	0	0	0	0	0
	0.90	125	0	0	125	0	0	0	0	0
300	0.10	35	88	2	16	105	4	90	109	120
	0.25	116	9	0	85	35	5	9	35	100
	0.45	125	0	0	125	0	0	0	0	50
	0.60	115	2	8	125	0	0	2	0	25
	0.75	111	0	14	104	0	21	0	0	0
	0.90	93	0	32	100	0	25	0	0	0
400	0.10	28	97	0	29	96	0	97	96	125
	0.25	89	34	2	107	18	0	34	18	100
	0.45	78	19	28	89	17	19	23	17	50
	0.60	80	9	36	83	8	34	9	8	25
	0.75	101	0	24	105	0	20	0	0	0
	0.90	95	0	30	105	0	20	0	0	0
500	0.10	23	102	0	26	99	0	102	99	125
	0.25	67	54	4	74	46	5	54	46	100
	0.45	92	17	16	84	20	21	17	20	50
	0.60	114	3	8	111	3	11	3	3	25
	0.75	125	0	0	99	0	26	0	0	0
	0.90	125	0	0	115	0	10	0	0	0
Total		2814	640	296	2846	622	282	646	624	1344

The impact of iteration number for HEA, COMA and InCEA is observed on a graph where $n = 500$ and $\alpha = 0.90$. The generated graph is nearly fully-connected, thus the algorithms need the highest number of iterations to obtain their best results. The performance of our algorithm stays steady after 1000^{th} iteration, on the other hand HEA and COMA reach their best solutions after 200^{th} iteration. Hence, we set the iteration number as 1000 for all the algorithms. In the rest of the experiments, the population size and the the number of iterations are set to 100 and 1000 unless stated otherwise.

The performance of algorithms for a given (n, α) pair with variable β values are shown in Table 2 and Table 3. Each row in Table 2 represents the average results obtained from 125 test cases. As the value of α increases, more number of edges between the vertices are added to the graph and this increments both the number of uncolored vertices and the fitness value of the algorithms. This also yields an increase on the execution time, since there is a growth in the search space of the algorithms. This trend can easily be seen using

TABLE 4. Comparison of the algorithms with respect to their fitness values and number of uncolored vertices for 5 different color class densities.

n	β	Fitness Values			Uncolored Vertex			Time(s)		
		HEA	COMA	InCEA	HEA	COMA	InCEA	HEA	COMA	InCEA
100	0.04	376.1	375.7	258.7	80.9	81.1	60.8	1609	1071	1
	0.08	297.4	296.9	149.2	67.4	67.6	39.3	729	520	2
	0.1	268.5	268.1	115.1	62.2	62.4	31.9	589	431	2
	0.15	213.3	213	58.7	51.8	52	18.6	399	336	4
	0.2	172.8	173	29.2	44.2	44.5	10.6	314	262	6
200	0.04	647.2	678.7	462.7	147.9	154.5	112.6	7045	5406	8
	0.08	466.3	512.1	254.6	114.2	124.6	70.3	3062	2576	19
	0.1	405.6	452.6	191.7	102.5	112.9	56.6	2404	2101	26
	0.15	295.4	344.4	89.8	79.2	90.8	30.6	1543	1468	51
	0.2	223.9	270.1	39.6	63.3	74.4	15.8	1135	1120	84
300	0.04	1079.9	1012.5	694.5	233.3	220.7	162.9	13680	10297	30
	0.08	854.5	786.5	375.9	194	179.4	99.8	6141	5611	83
	0.1	769.1	706.3	277.1	179.2	165.8	78.4	4827	4857	118
	0.15	590.8	551.6	125.1	146.8	137.5	40.8	3313	4002	237
	0.2	450.1	422.1	53.6	119.1	112.1	20.3	2531	2877	400
400	0.04	1211	1277.3	885.2	277	290.8	213.2	28328	12988	82
	0.08	833.3	924	469.8	203.9	224.7	128.9	12346	5494	239
	0.1	710.5	807.8	340.7	179.1	201.7	100.1	9625	3941	349
	0.15	501.8	606.7	149.1	133.8	158.9	50.1	5917	3060	721
	0.2	356.2	469.4	61.4	98.7	127.2	24.3	4275	2738	1236
500	0.04	1598.2	1518.3	1070.5	358.2	349.3	263.6	32167	33825	183
	0.08	1236.4	1104.8	551.9	288	269	157.1	14233	17373	568
	0.1	1072.9	976.6	393.3	254.1	242.6	120.0	10957	13324	840
	0.15	869.5	745.2	164.4	216.5	194	59.1	7355	13137	1771
	0.2	723	591.7	64.0	189.3	160.9	27.5	5620	11203	3069

TABLE 5. Pairwise comparison of algorithms with respect to β values.

n	β	Pairwise Comparisons						# of Feasible Test Cases		
		InCEA-HEA			InCEA-COMA					
		Better	Equal	Worse	Better	Equal	Worse	HEA	COMA	InCEA
100	0.04	140	3	7	142	2	6	0	0	0
	0.08	136	9	5	136	9	5	9	9	25
	0.1	132	13	5	134	11	5	13	11	48
	0.15	120	26	4	121	24	5	25	24	50
	0.2	115	35	0	116	34	0	35	34	76
200	0.04	126	0	24	135	0	15	1	0	0
	0.08	121	10	19	130	10	10	13	10	25
	0.1	117	20	13	125	15	10	20	15	50
	0.15	100	40	10	115	30	5	40	30	75
	0.2	95	50	5	110	40	0	50	40	100
300	0.04	134	3	13	129	6	15	5	10	20
	0.08	126	15	9	111	29	10	15	29	50
	0.1	121	20	9	110	30	10	20	30	50
	0.15	114	25	11	105	35	10	25	35	75
	0.2	100	36	14	100	40	10	36	40	100
400	0.04	115	7	28	120	7	23	7	7	25
	0.08	101	20	29	110	17	23	20	17	50
	0.1	93	27	30	103	24	23	31	24	50
	0.15	86	43	21	98	36	16	43	36	75
	0.2	76	62	12	87	55	8	62	55	100
500	0.04	129	11	10	118	8	24	11	8	25
	0.08	121	23	6	109	23	18	23	23	50
	0.1	110	31	9	103	31	16	31	31	50
	0.15	97	50	3	96	45	9	50	45	75
	0.2	89	61	0	83	61	6	61	61	100
Total		2814	640	296	2846	622	282	646	624	1344

the results obtained from the performance of the algorithms. The results indicate that the proposed algorithm outperforms the two algorithms from the literature with respect to fitness value, number of uncolored vertices and total execution time.

2) PERFORMANCE EVALUATION OF THE ALGORITHMS

The performance of HEA, COMA and InCEA are evaluated on various randomly generated graphs with different properties. For each combination of n and α given in Table 1, five random graphs are generated. Therefore, a total of 150 different graphs are used to measure the performance of the algorithms with five different β values. For 750 test cases, the algorithms are executed five times in this section.

Table 3 makes a pairwise comparison using two columns, and each column represents the total number of test cases (out of 125) where the first algorithm has a better, equal or worse solution as compared to the second algorithm. When α is equal to 0.10, the algorithms obtain the same performance in at least 60% of the test cases since the graphs are nearly sparse. The proposed algorithm outperforms both HEA and COMA in most of the test cases provided when there is an increase in the value of α. In the third column, the number of test cases where all vertices can be colored is given. The number of colors algorithms use is proportional to n; so for larger values of n, the total number of test cases where the algorithms can color all vertices increases. When $\alpha = \{0.75, 0.90\}$, the graph is nearly fully connected; so in none of the test cases, the algorithms are able to color all vertices of the graphs. Our algorithm outperforms HEA and COMA in 75% and 76% of all of the test cases, respectively. The proposed algorithm can color all vertices in 36% of the test cases, whereas the results for COMA and HEA are both 17%.

The performance of algorithms are also evaluated for (n, β) pair with variable α values. Lower β values imply that the algorithms use less number of colors, so both the fitness value and number of uncolored vertices is high for low β values as can be seen in Table 4. The performance of the algorithms are measured for each (n, β) pair and in each row, the average results (out of 150 test cases) for fitness and the number of uncolored vertices are given. Our proposed work obtains the best results in each row.

In Table 5, a pairwise comparison of the algorithms are given. For larger (β) values, the number of colors used by the algorithms increase, and this results as an assignment of most of the vertices to the given colors. Our algorithm has the highest number of test cases where all vertices are successfully colored.

In our proposed work, the uncolored vertices that minimizes the fitness value are stored in V_{tabu} once the local search technique is completed. So our algorithm does not need any additional calculations to find the best vertex(es) to be

TABLE 6. The best number of colors used by the algorithms for R, queen and myciel graphs with their computation time.

Graph	|V(G)|	|E(G)|	HEA		COMA		InCEA		Δ_1	Δ_2
			k	Time	k	Time	k	Time		
R50_1g.col	50	108	**4**	415	**3**	1361	**3**	1	-1	0
R50_1gb.col	50	108	3	1025	3	1456	3	1	0	0
R50_5g.col	50	612	10	526	10	922	10	12	0	0
R50_5gb.col	50	612	10	454	10	881	10	13	0	0
R50_9g.col	50	1092	**21**	452	**22**	1278	**21**	47	0	-1
R50_9gb.col	50	1092	**21**	354	**22**	1286	**21**	48	0	-1
R75_1g.col	70	251	**5**	1026	**4**	2799	**4**	5	-1	0
R75_1gb.col	70	251	4	2214	4	2605	4	5	0	0
R75_5g.col	75	1407	**14**	1110	**14**	1920	**13**	49	-1	-1
R75_5gb.col	75	1407	**14**	990	**14**	1833	**13**	48	-1	-1
R75_9g.col	75	2513	33	830	33	2820	33	253	0	0
R75_9gb.col	75	2513	33	515	33	2640	33	253	0	0
R100_1g.col	100	509	**6**	2416	**6**	3181	**5**	15	-1	-1
R100_1gb.col	100	509	**6**	2579	**6**	3268	**5**	14	-1	-1
R100_5g.col	100	2456	**17**	1932	**17**	3402	**15**	109	-2	-2
R100_5gb.col	100	2456	**17**	1858	**16**	3787	**15**	105	-2	-1
R100_9g.col	100	4438	**37**	1964	**38**	4600	**36**	510	-1	-2
R100_9gb.col	100	4438	**38**	739	**39**	4257	**36**	527	-2	-3
myciel5g.col	47	236	6	420	6	580	6	4	0	0
myciel5gb.col	47	236	6	424	6	592	6	4	0	0
myciel6g.col	95	755	7	2534	7	3169	7	18	0	0
myciel6gb.col	95	755	7	2655	7	3379	7	18	0	0
myciel7g.col	191	2360	8	21887	8	25215	8	92	0	0
myciel7gb.col	191	2360	8	20727	8	24216	8	92	0	0
queen8_8g.col	64	728	10	857	10	1379	9	19	-1	-1
queen8_8gb.col	64	728	10	756	10	1263	9	19	-1	-1
queen9_9g.col	81	1056	11	1446	11	2172	11	38	0	0
queen9_9gb.col	81	1056	11	1143	11	1951	11	37	0	0
queen10_10g.col	100	1470	**13**	1853	**12**	3650	**13**	76	0	+1
queen10_10gb.col	100	1470	**12**	2117	**13**	2739	**13**	74	+1	0
queen11_11g.col	121	1980	14	2909	14	4453	14	124	0	0
queen11_11gb.col	121	1980	14	2607	14	3924	14	127	0	0
queen12_12g.col	144	2596	15	4083	15	6513	15	201	0	0
queen12_12gb.col	144	2596	15	3490	15	5422	15	201	0	0
#Better			1/34		1/34		10/34		16	16
#Equal			21/34		21/34		22/34		21	21
#Worse			12/34		11/34		2/34		1	1

TABLE 7. The best number of colors used by the algorithms for GEOM and DSJC graphs with their computation time.

Graph	|V(G)|	|E(G)|	HEA		COMA		InCEA		Δ_1	Δ_2
			k	Time(s)	k	Time(s)	k	Time(s)		
GEOM20.col	20	20	5	75	5	103	5	0,1	0	0
GEOM20a.col	20	37	5	82	5	110	5	0,1	0	0
GEOM20b.col	20	32	3	77	3	117	3	0,1	0	0
GEOM30.col	30	50	6	156	6	220	6	1	0	0
GEOM30a.col	30	81	6	172	6	247	6	1	0	0
GEOM30b.col	30	81	5	155	5	235	5	1	0	0
GEOM40.col	40	78	6	286	6	396	6	2	0	0
GEOM40a.col	40	146	7	293	7	436	7	4	0	0
GEOM40b.col	40	157	7	298	7	430	7	4	0	0
GEOM50.col	50	127	6	438	6	596	6	4	0	0
GEOM50a.col	50	238	9	469	9	717	9	9	0	0
GEOM50b.col	50	249	8	445	8	674	8	8	0	0
GEOM60.col	60	185	6	673	6	918	6	6	0	0
GEOM60a.col	60	339	10	653	10	1011	10	16	0	0
GEOM60b.col	60	366	9	669	9	1030	9	14	0	0
GEOM70.col	70	267	8	868	8	1220	8	13	0	0
GEOM70a.col	70	459	11	913	11	1377	11	26	0	0
GEOM70b.col	70	488	10	1004	10	1543	10	23	0	0
GEOM80.col	80	349	8	1206	8	1777	8	19	0	0
GEOM80a.col	80	612	12	1106	12	1749	12	41	0	0
GEOM80b.col	80	663	**12**	1279	**13**	1837	**12**	38	0	-1
GEOM90.col	90	441	8	1892	8	2580	8	22	0	0
GEOM90a.col	90	789	13	1544	13	2236	13	61	0	0
GEOM90b.col	90	860	15	1445	15	2352	15	74	0	0
GEOM100.col	100	547	9	2211	9	3101	9	34	0	0
GEOM100a.col	100	992	14	1915	14	2875	14	86	0	0
GEOM100b.col	100	1050	15	1846	15	2895	15	96	0	0
GEOM110.col	110	638	9	2790	9	3411	9	42	0	0
GEOM110a.col	110	1207	15	2051	15	3224	15	118	0	0
GEOM110b.col	110	1256	16	1875	16	3293	16	128	0	0
GEOM120.col	120	773	11	2796	11	3967	11	70	0	0
GEOM120a.col	120	1434	**16**	2517	**16**	3967	**17**	174	+1	+1
GEOM120b.col	120	1491	17	2735	17	4428	17	171	0	0
DSJC125.1g.col	125	736	6	5669	6	7606	6	27	0	0
DSJC125.1gb.col	125	736	6	5685	6	6703	6	28	0	0
DSJC125.5g.col	125	3891	**21**	2562	**21**	4655	**20**	277	-1	-1
DSJC125.5gb.col	125	3891	**20**	2807	**21**	4528	**20**	280	0	-1
DSJC125.9g.col	125	6961	**48**	2768	**49**	6560	**46**	1315	-2	-3
DSJC125.9gb.col	125	6961	**48**	1429	**48**	6474	**45**	1233	-3	-3
#Better			1/39		1/39		3/39		6	9
#Equal			35/39		33/39		35/39		35	33
#Worse			3/39		5/39		1/39		1	1

uncolored yielding the lowest fitness value. But in both HEA and COMA, there is an exhaustive search in the local search technique and in the selection of uncolored vertices for fitness calculation, that's why the execution time of the proposed algorithm is significantly better than HEA and COMA as can be clearly seen in Tables 2 and 4.

B. DIMACS BENCHMARKS

There are 73 DIMACS instances [51] with five different types for the vertex-weighted graph coloring problem. The graphs that are created for the register allocation problem are denoted as "Rn_αg{b}*.col". The GEOM graphs are geometric graphs [52] and they are denoted as "GEOM{a,b}n.col" where a and b correspond to the density percentage. The queen graphs are built using nxn chessboard problem [53], which are represented as "queenn_n.col". The DSJC graphs are random graphs [54] for simulated annealing problem [55] and are shown as "DSJCn_αg{b}*.col". The myciel graphs are constructed based on the Mycielski transformation [56] and are denoted as "myciel{5, 6, 7}g{b}*.col". In R, myciel, queen and DSJC graph names, the weight range of the vertices are represented with either "g" ($\gamma = [1..5]$) or "gb" ($\gamma = [1..20]$).

For the experiments regarding DIMACS benchmarks, first the minimum k value for each algorithm to obtain legal k-coloring is found. The experiments start with a sufficiently large k value and k is decreased until an illegal solution is found where some vertex(es) of the graphs are uncolored. The minimum k found by the algorithms is referred as the

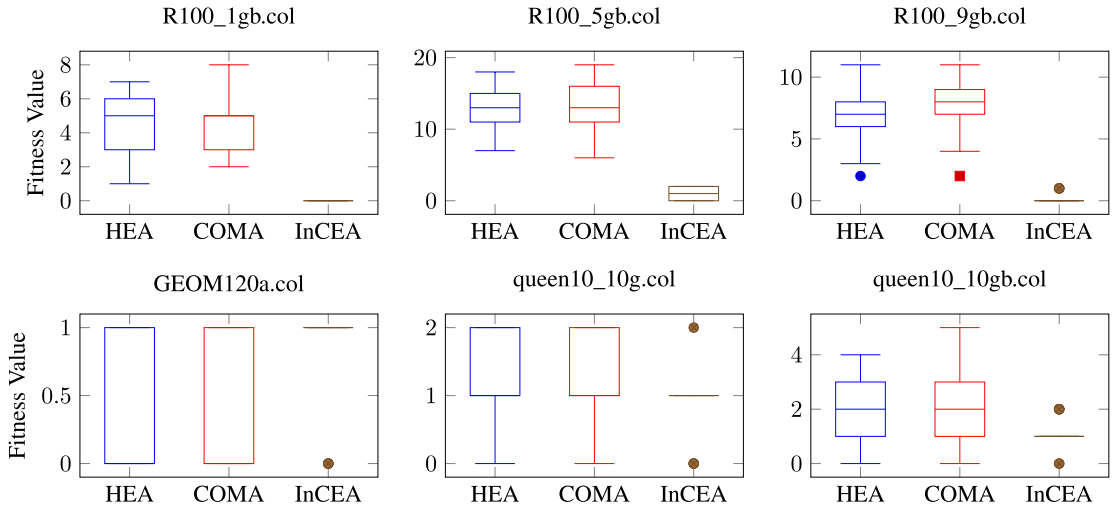


FIGURE 8. Fitness values of HEA, COMA and InCEA on a set of DIMACS instances (R100_1gb.col, R100_5gb.col, R100_9gb.col, GEOM120a.col, queen10_10g.col and queen10_10gb.col) using 5, 15, 36, 16, 12 and 12 color classes, respectively.

minimum number of colors used by the algorithms. Once the best k value (k_{best}) is found for each instance, the performance of the algorithms is measured using different scenarios.

Table 6 and Table 7 present performance comparison of the algorithms on 73 DIMACS instances, where each algorithm is executed five times to obtain the best result using 30000 iterations. The graph name, number of vertices and number of edges in the graphs are given in the first three columns of the tables. For each algorithm, the minimum number of colors used and the execution time are listed in two separate columns. On the other hand, the column denoted by Δ_1 in tables presents the difference between the number of colors used by InCEA and HEA algorithms, $k_{\text{InCEA}} - k_{\text{HEA}}$. Similarly Δ_2 column presents $k_{\text{InCEA}} - k_{\text{COMA}}$ for graphs considered in our experiments.

InCEA outperforms HEA and COMA in 30% and 33% of the instances, whereas in three of them: namely queen10_10g.col, queen10_10gb.col and GEOM120a.col; InCEA has used one additional color. For these three instances, we further investigate the behavior of the algorithms. We also add three instances R100_1gb.col, R100_5gb.col and R100_9gb.col where InCEA obtains better results. For these six instances, the algorithms are executed 20 times using k_{best}, the fitness values and the number of uncolored vertices are shown in Fig. 8 and Fig. 9, respectively. For example, the minimum number of colors used by HEA and COMA to color R100_1gb.col is 6, whereas InCEA successfully colored this instance using 5 colors. So k_{best} to obtain a legal k-coloring for this instance is 5, and all algorithms are executed using 5 number of color classes. The fitness value is 0 for a feasible solution, whereas if an infeasible solution is found, then the fitness value of the algorithm is calculated as the total weight of the uncolored vertex(es).

When the fitness values of the algorithms are considered, the results of our proposed work is more condensed for the first three instances, R100_1gb.col, R100_5gb.col and R100_9gb.col. In GEOM120a.col, InCEA can also obtain the best fitness value of 0, but it is an outlier for our algorithm because in 80% of the runs our algorithm obtains 1, whereas HEA and COMA obtains the best fitness value of 0 in 70% and 65% of the runs, respectively. For both queen10_10g.col and queen10_10gb.col, InCEA can obtain the best fitness value in 15% and 10% of the runs and this is considered as an outlier because our algorithm varies less as compared to HEA and COMA and produces the fitness value of 1 in majority of the cases so it is more consistent. In Fig. 9 for the first three instances, InCEA has the minimum number of vertices uncolored. Even the minimum values observed by HEA and COMA are larger than the outlier or Q3 value of InCEA. As for GEOM120a.col, queen10_10g.col and queen10_10gb.col, the behavior of the algorithms are very similar to that of the fitness values.

In order to show the performance of the algorithms for variable number of color classes, the algorithms are executed 10 times for color class values less than or equal to k_{best}. The average fitness values are given in Fig. 10 for the same six instances. HEA and COMA follows nearly the same pattern with slight differences whereas InCEA reaches the minimum fitness value using minimum number of colors for the first three instances. InCEA has the worst performance and uses the maximum number of colors for coloring GEOM120a.col. All algorithms have similar performance for queen10_10g.col and queen10_10gb.col.

The performance of the algorithms are also measured for a fixed time limit, which is set to 500 seconds. The same six instances and their k_{best} values are used in these experiments. Fig. 11 shows the average fitness values obtained using five runs. For the first three instances, R100_1gb.col, R100_5gb.col and R100_9gb.col, our algorithm successfully obtains k-legal coloring and outperforms both HEA and COMA. In the previous tests, the number of iterations

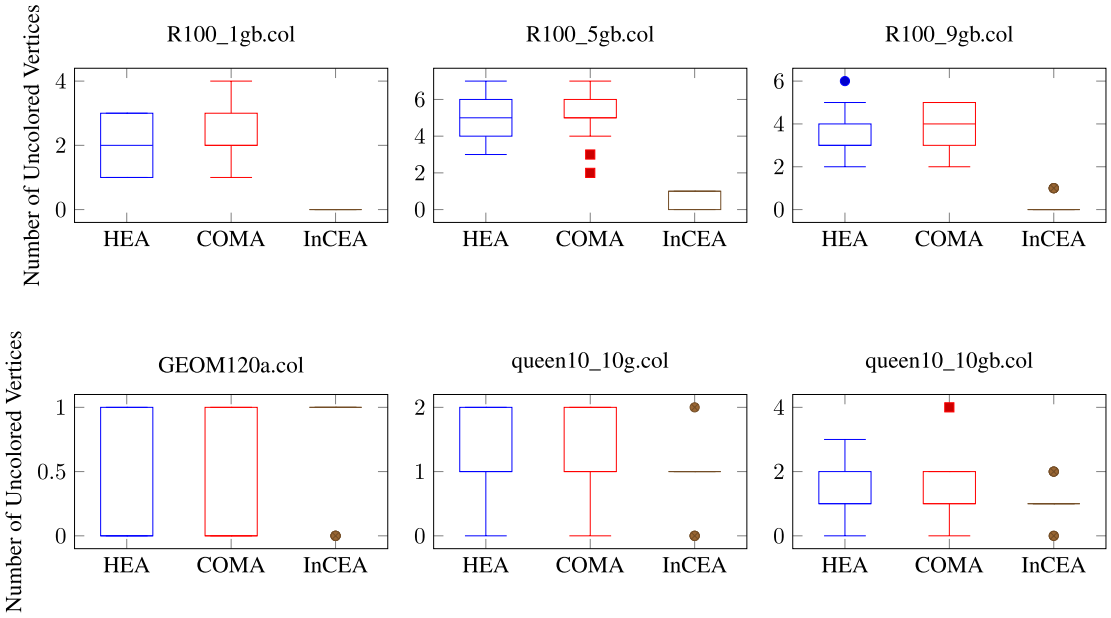


FIGURE 9. Number of uncolored vertices of HEA, COMA and InCEA on a set of DIMACS instances (R100_1gb.col, R100_5gb.col, R100_9gb.col, GEOM120a.col, queen10_10g.col and queen10_10gb.col) using 5, 15, 36, 16, 12 and 12 color classes, respectively.

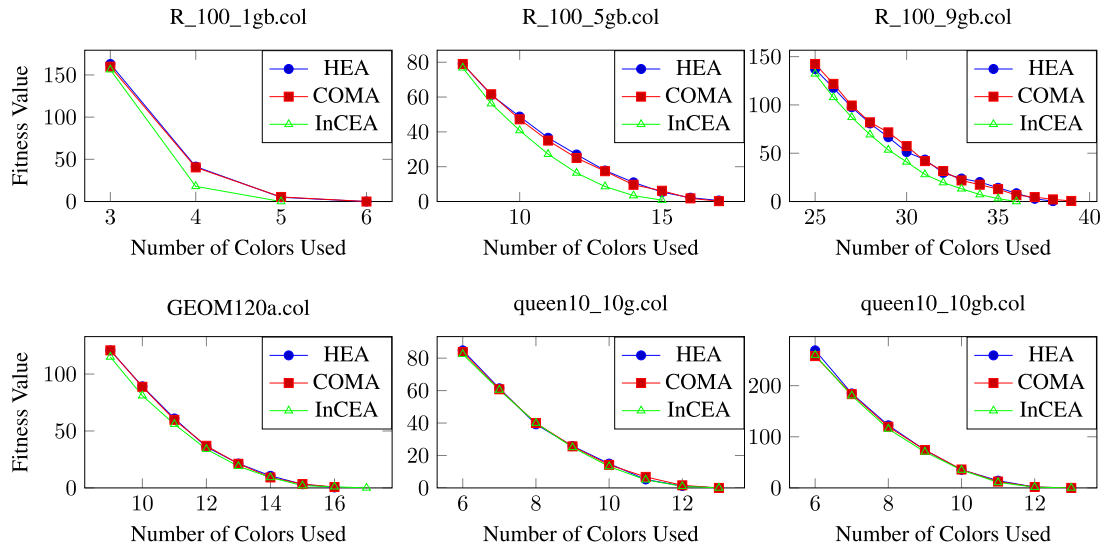


FIGURE 10. The fitness values of HEA, COMA and InCEA on a set of DIMACS instances using variable number of color classes.

is the main parameter which is fixed to 30000 and our algorithm was not successful in coloring GEOM120a.col, queen10_10g.col and queen10_10gb.col in most of the cases. It successfully colors these three instances using higher number of iterations. When the average fitness values are considered, COMA reaches the best result in a very short amount of time for GEOM120a.col. For the same instance, InCEA also finds a feasible k-coloring in most of the test cases. InCEA obtains the best performance in queen10_10g.col and queen10_10gb.col in terms of average fitness values.

Since HEA and COMA make their decisions based on some smart metrics in all phases of the algorithms, they converge to a solution that can no longer improve. Our algorithm also includes intelligent mechanisms such as search back operation or weighted-swap operation but it selects the color classes randomly in crossover phase. So it continues to explore the search space if it is allowed to use large number of iterations.

For the same test settings, the average distribution of computation time (in percentage) of the algorithms and the average number of iterations they use are measured. As can be clearly seen from Table 8, both HEA and COMA spend most of their time in the local search phase, whereas due to the usage of the pool and tabu list, InCEA spends less than 5%

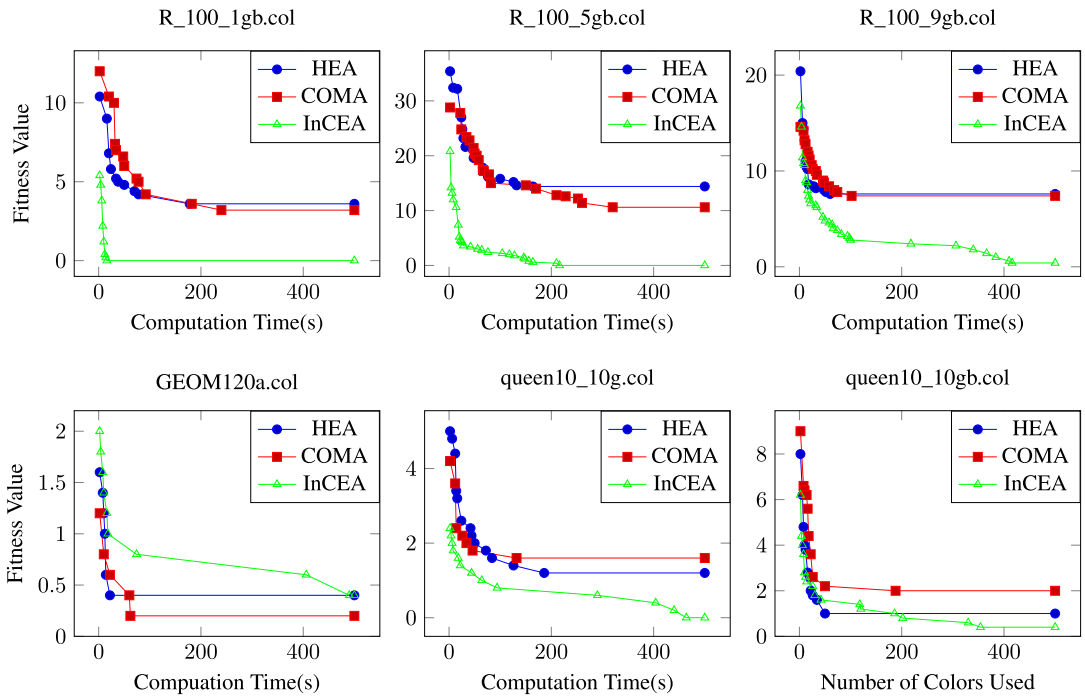


FIGURE 11. The fitness values of HEA, COMA and InCEA on a set of DIMACS graphs (R100_1gb.col, R100_5gb.col, R100_9gb.col, GEOM120a.col, queen10_10g.col and queen10_10gb.col) using 5, 15, 36, 16, 12 and 12 color classes, respectively when the computation time limit is set to 500 seconds.

TABLE 8. Distribution of computation time of algorithms (in percentage scale), and their number of iterations in 500 seconds for 6 DIMACS instances.

Instance	Crossover (%)			Local Search (%)			Fitness Calculation (%)			Number of Iterations		
	HEA	COMA	InCEA	HEA	COMA	InCEA	HEA	COMA	InCEA	HEA	COMA	InCEA
R100_1gb.col	0.2	0.4	94.8	97.6	97.2	4.6	2.2	2.4	0.6	1120	760	595703
R100_5gb.col	3.2	1.9	97.4	94.9	96.4	2.2	1.8	1.7	0.4	1060	870	115649
R100_9gb.col	17.6	6.4	99.4	80	92.6	0.5	2.4	0.9	0.1	3740	520	18160
queen10_10g.col	2.5	1.5	97.7	95.9	97.3	2.2	1.6	1.2	0.2	1525	1600	195163
queen10_10gb.col	2.1	1.8	97.7	96.3	96.3	2	1.6	1.9	0.3	2710	1690	110366
GEOM120a.col	3.4	2.1	99.2	95.4	97.2	0.7	1.2	0.7	0.1	1265	1150	57875

and 1% of its computation time in the local search phase and the fitness calculation phase, respectively. This experiment reveals the algorithmic differences of the algorithms and the reasons behind obtaining different iteration numbers for a fixed time limit.

VI. CONCLUSION AND FUTURE WORK

In this paper, we propose a novel evolutionary algorithm for the vertex-weighted k-coloring problem. The algorithm has two objectives: (1) maximize the number of non-conflicting vertices that can be assigned to the same color using the integrated crossover operator, (2) minimize the total weight of the uncolored vertices by applying weighted-swap as a local search technique. The integrated crossover operator (InCX) uses a pool and a search back operation. The pool holds the vertices that has at least one conflict with the vertices available in the color classes of the offspring, so it separates the feasible (vertices assigned to the color classes of the offspring) and infeasible (vertices available in the pool) parts of the solution. The search back operation tries to decrease the number of vertices in the pool by assigning them to a color class in the offspring if possible. The local search technique targets to decrease sum of weights of the uncolored vertices considering neighborhood solutions. Once the local search technique is finished, V_{tabu} contains the uncolored vertices, so without the need of an exhaustive search, the fitness value can be calculated very fast.

We have performed an experimental evaluation using synthetic benchmarks and DIMACS graphs, synthetic benchmarks contain 150 graphs and 3750 test cases, whereas DIMACS has 73 well-known graphs. The proposed work is compared with a hybrid evolutionary algorithm and a memetic algorithm that targets to solve the same problem. The experimental study indicates that the proposed algorithm

generated successful results outperforming both algorithms from the literature with respect to total number of colors used, fitness value and execution time.

The graph coloring problem plays an important role for the solution of many theoretical and practical problems such as resource allocation problem and register allocation problem. As a future work the proposed algorithm can be applied to many real world problems to obtain successful results within a reasonable amount of time. Our algorithm can be a good match for solving the dynamic graph coloring problem as well, because it can easily adapt to the dynamic changes in the graph. Also the proposed integrated crossover operator can be used for a wide variety of grouping or clustering problems where the items in the problem have constraints.

ACKNOWLEDGMENT

This article [47] was presented in part at the Genetic and Evolutionary Computation Conference (GECCO'15), Madrid, Spain, in 2015.

REFERENCES

[1] H. Furmanczyk, "Equitable coloring of graphs," *Amer. Math. Soc.*, vol. 352, pp. 35–53, Jun. 2004.
[2] E. Malaguti, "The vertex coloring problem and its generalizations," *4OR-Q J. Oper. Res.*, vol. 7, no. 1, pp. 101–104, Mar. 2009, doi: 10.1007/s10288-008-0071-y.
[3] T. R. Jensen and B. Toft, *Graph Coloring Problems*. New York, NY, USA: Wiley, 1995.
[4] R. M. R. Lewis, *A Guide to Graph Colouring*, vol. 7. Berlin, Germany: Springer, 2015, doi: 10.1007/978-3-319-25730-3.
[5] R. W. Quong and S.-C. Chen, "Register allocation via weighted graph coloring (technical summary)," Purdue Univ. Lib., West Lafayette, IN, USA, ECE Tech. Rep. TR-EE 93-23 (232), Jun. 1993.
[6] M. R. Garey and D. S. Johnson, *Computers and Intractability: A Guide to the Theory of NP-Completeness*. New York, NY, USA: Freeman, 1990.
[7] I. Blöchliger and N. Zufferey, "A graph coloring heuristic using partial solutions and a reactive tabu scheme," *Comput. Oper. Res.*, vol. 35, no. 3, pp. 960–975, Mar. 2008.
[8] P. Galinier and A. Hertz, "A survey of local search methods for graph coloring," *Comput. Oper. Res.*, vol. 33, no. 9, pp. 2547–2562, Sep. 2006.
[9] Z. Lü and J.-K. Hao, "A memetic algorithm for graph coloring," *Eur. J. Oper. Res.*, vol. 203, no. 1, pp. 241–250, May 2010.
[10] L. Moalic and A. Gondran, "The new memetic algorithm HEAD for graph coloring: An easy way for managing diversity," in *Proc. Eur. Conf. Evol. Comput. Combinat. Optim.*, Copenhagen, Denmark, Apr. 2015, pp. 173–183, doi: 10.1007/978-3-319-16468-7_15.
[11] Y. Zhou, B. Duval, and J.-K. Hao, "Improving probability learning based local search for graph coloring," *Appl. Soft Comput.*, vol. 65, pp. 542–553, Apr. 2018, doi: 10.1016/j.asoc.2018.01.027.
[12] W. Wang, J.-K. Hao, and Q. Wu, "Tabu search with feasible and infeasible searches for equitable coloring," *Eng. Appl. Artif. Intell.*, vol. 71, pp. 1–14, May 2018, doi: 10.1016/j.engappai.2018.01.012.
[13] W. Sun, J.-K. Hao, W. Wang, and Q. Wu, "Memetic search for the equitable coloring problem," *Knowl.-Based Syst.*, vol. 188, Jan. 2020, Art. no. 105000, doi: 10.1016/j.knosys.2019.105000.
[14] P. Galinier, A. Hertz, and N. Zufferey, "An adaptive memory algorithm for the k-colouring problem," *Discrete Appl. Math.*, vol. 156, no. 2, pp. 267–279, 2008.
[15] L. Moalic and A. Gondran, "Variations on memetic algorithms for graph coloring problems," *J. Heuristics*, vol. 24, no. 1, pp. 1–24, Feb. 2018, doi: 10.1007/s10732-017-9354-9.
[16] M. Mrad, O. Harrabi, J. C. Siala, and A. Gharbi, "A column generation-based lower bound for the minimum sum coloring problem," *IEEE Access*, vol. 8, pp. 57891–57904, 2020, doi: 10.1109/ACCESS.2020.2973122.
[17] W. Sun, J.-K. Hao, X. Lai, and Q. Wu, "Adaptive feasible and infeasible tabu search for weighted vertex coloring," *Inf. Sci.*, vol. 466, pp. 203–219, Oct. 2018, doi: 10.1016/j.ins.2018.07.037.
[18] A. E. Eiben and J. E. Smith, *Introduction to Evolutionary Computing*, vol. 53. Berlin, Germany: Springer, 2003, doi: 10.1007/978-3-662-44874-8.
[19] G. J. Chaitin, "Register allocation & spilling via graph coloring," *ACM SIGPLAN Notices*, vol. 17, no. 6, pp. 98–101, Jun. 1982, doi: 10.1145/872726.806984.
[20] S. Shamizi and S. Lotfi, "Register allocation via graph coloring using an evolutionary algorithm," in *Proc. SEMCCO*, Visakhapatnam, India, 2011, pp. 1–8.
[21] A. Mishra, S. Banerjee, and W. Arbaugh, "Weighted coloring based channel assignment for WLANs," *ACM SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 9, no. 3, pp. 19–31, Jul. 2005, doi: 10.1145/1094549.1094554.
[22] L. Zhao, H. Wang, and X. Zhong, "Interference graph based channel assignment algorithm for D2D cellular networks," *IEEE Access*, vol. 6, pp. 3270–3279, 2018, doi: 10.1109/ACCESS.2018.2789423.
[23] S. Uygungelen, G. Auer, and Z. Bharucha, "Graph-based dynamic frequency reuse in femtocell networks," in *Proc. IEEE 73rd Veh. Technol. Conf. (VTC Spring)*, Yokohama, Japan, May 2011, pp. 1–6, doi: 10.1109/VETECS.2011.5956438.
[24] L. Tan, Z. Feng, W. Li, Z. Jing, and T. A. Gulliver, "Graph coloring based spectrum allocation for femtocell downlink interference mitigation," in *Proc. IEEE Wireless Commun. Netw. Conf.*, Cancun, Mexico, Mar. 2011, pp. 1248–1252, doi: 10.1109/WCNC.2011.5779338.
[25] M. Miri, K. Mohamedpour, Y. Darmani, M. Sarkar, and R. L. Tummala, "An efficient resource allocation algorithm based on vertex coloring to mitigate interference among coexisting WBANs," *Comput. Netw.*, vol. 151, pp. 132–146, Mar. 2019, doi: 10.1016/j.comnet.2019.01.014.
[26] E. K. Burke, B. McCollum, A. Meisels, S. Petrovic, and R. Qu, "A graph-based hyper-heuristic for educational timetabling problems," *Eur. J. Oper. Res.*, vol. 176, no. 1, pp. 177–192, Jan. 2007, doi: 10.1016/j.ejor.2005.08.012.
[27] M. Boudhar and G. Finke, "Scheduling on a batch machine with job compatibilities," *Belg. J. Oper. Res. Statist. Comput. Sci.*, vol. 40, nos. 1–2, pp. 69–80, 2000.
[28] C. A. Glass, "Bag rationalisation for a food manufacturer," *J. Oper. Res. Soc.*, vol. 53, no. 5, pp. 544–551, May 2002.
[29] N. Zufferey, P. Amstutz, and P. Giaccari, "Graph colouring approaches for a satellite range scheduling problem," *J. Scheduling*, vol. 11, no. 4, pp. 263–277, Aug. 2008.
[30] M. Gamache, A. Hertz, and J. O. Ouellet, "A graph coloring model for a feasibility problem in monthly crew scheduling with preferential bidding," *Comput. Oper. Res.*, vol. 34, no. 8, pp. 2384–2395, Aug. 2007, doi: 10.1016/j.cor.2005.09.010.
[31] P. B. Myszkowski, "Solving scheduling problems by evolutionary algorithms for graph coloring problem," in *Metaheuristics for Scheduling in Industrial and Manufacturing Applications* (Studies in Computational Intelligence), vol. 128. Berlin, Germany: Springer, 2008, pp. 145–167.
[32] S. Gualandi and F. Malucelli, "Exact solution of graph coloring problems via constraint programming and column generation," *INFORMS J. Comput.*, vol. 24, no. 1, pp. 81–100, Feb. 2012.
[33] Z. Zhou, C.-M. Li, C. Huang, and R. Xu, "An exact algorithm with learning for the graph coloring problem," *Comput. Oper. Res.*, vol. 51, pp. 282–301, Nov. 2014, doi: 10.1016/j.cor.2014.05.017.
[34] J. Guo, L. Moalic, J.-N. Martin, and A. Caminada, "Exact graph coloring algorithms of getting partial and all best solutions," in *Proc. Int. Symp. Artif. Intell. Math. (ISAIM)*, Fort Lauderdale, FL, USA, Jan. 2018, pp. 102–109.
[35] D. Brélaz, "New methods to color the vertices of a graph," *Commun. ACM*, vol. 22, no. 4, pp. 251–256, Apr. 1979, doi: 10.1145/359094.359101.
[36] C. L. Mumford, "New order-based crossovers for the graph coloring problem," in *Parallel Problem Solving from Nature*. Reykjavík, Iceland: Springer, 2006, pp. 880–889.
[37] R. Dorne, and J.-K. Hao, "A new genetic local search algorithm for graph coloring," in *Parallel Problem Solving from Nature* (Lecture Notes in Computer Science), vol. 1498. Amsterdam, The Netherlands: Springer, 1998, pp. 745–754.
[38] D. C. Porumbel, J.-K. Hao, and P. Kuntz, "Diversity control and multi-parent recombination for evolutionary graph coloring algorithms," in *Proc. Eur. Conf. Evol. Comput. Combinat. Optim.*, Tübingen, Germany, Apr. 2009, pp. 121–132.
[39] B. Escoffier, J. Monnot, and V. T. Paschos, "Weighted coloring: Further complexity and approximability results," in *Proc. Italian Conf. Theor. Comput. Sci.* Siena, Italy: Springer, 2005, pp. 205–214.

[40] T. Karthick, F. Maffray, and L. Pastor, "Polynomial cases for the vertex coloring problem," *Algorithmica*, vol. 81, no. 3, pp. 1053–1074, Mar. 2019, doi: 10.1007/s00453-018-0457-y.
[41] M. Kubale and B. Jackowski, "A generalized implicit enumeration algorithm for graph coloring," *Commun. ACM*, vol. 28, no. 4, pp. 412–418, Apr. 1985, doi: 10.1145/3341.3350.
[42] E. Malaguti, M. Monaci, and P. Toth, "An exact approach for the vertex coloring problem," *Discrete Optim.*, vol. 8, no. 2, pp. 174–190, May 2011, doi: 10.1016/j.disopt.2010.07.005.
[43] A. E. Eiben, J. K. Van Der Hauw, and J. I. van Hemert, "Graph coloring with adaptive evolutionary algorithms," *J. Heuristics*, vol. 4, no. 1, pp. 25–46, 1998.
[44] H. R. Topcuoglu, B. Demiroz, and M. Kandemir, "Solving the register allocation problem for embedded systems using a hybrid evolutionary algorithm," *IEEE Trans. Evol. Comput.*, vol. 11, no. 5, pp. 620–634, Oct. 2007, doi: 10.1109/TEVC.2007.892766.
[45] J. Wu, Z. Chang, L. Yuan, Y. Hou, and M. Gong, "A memetic algorithm for resource allocation problem based on node-weighted graphs [application notes]," *IEEE Comput. Intell. Mag.*, vol. 9, no. 2, pp. 58–69, May 2014, doi: 10.1109/MCI.2014.2307231.
[46] P. Galinier and J.-K. Hao, "Hybrid evolutionary algorithms for graph coloring," *J. Combinat. Optim.*, vol. 3, no. 4, pp. 379–397, Dec. 1999, doi: 10.1023/A:1009823419804.
[47] G. Sungu and B. Boz, "An evolutionary algorithm for weighted graph coloring problem," in *Proc. Companion Publication Genet. Evol. Comput. Conf. (GECCO Companion)*, Madrid, Spain, 2015, pp. 1233–1236.
[48] J. K. van der Hauw, "Evaluating and improving steady state evolutionary algorithms on constraint satisfaction problems," M.S. thesis, Dept. Comput. Sci., Leiden Univ., Leiden, The Netherlands, 1996.
[49] M. Rocha, C. Vilela, and J. Neves, "A study of order based genetic and evolutionary algorithms in combinatorial optimization problems," in *Proc. 13th IEA/AIE*, in Lecture Notes in Computer Science. New Orleans, LA, USA: Springer, 2000, pp. 601–611.
[50] M. Črepinšek, S.-H. Liu, and M. Mernik, "Replication and comparison of computational experiments in applied evolutionary computing: Common pitfalls and guidelines to avoid them," *Appl. Soft Comput.*, vol. 19, pp. 161–170, Jun. 2014, doi: 10.1016/j.asoc.2014.02.009.
[51] D. S. Johnson and M. A. Trick, *Cliques, Coloring, and Satisfiability: Second DIMACS Implementation Challenge*. Providence, RI, USA: American Mathematical Society, 1993.
[52] M. B. Dillencourt, D. Eppstein, and M. T. Goodrich, "Choosing colors for geometric graphs via color space embeddings," in *Proc. Int. Symp. Graph Drawing*. Karlsruhe, Germany: Springer, Sep. 2006, pp. 294–305.
[53] M. Vasquez, "New results on the queens_n^2 graph coloring problem," *J. Heuristics*, vol. 10, no. 4, pp. 407–413, 2004, doi: 10.1023/B:HEUR.0000034713.28244.e1.
[54] D. S. Johnson, C. R. Aragon, L. A. McGeoch, and C. Schevon, "Optimization by simulated annealing: An experimental evaluation; part II, graph coloring and number partitioning," *Oper. Res.*, vol. 39, no. 3, pp. 378–406, Jun. 1991.
[55] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[56] P. C. B. Lam, W. Lin, G. Gu, and Z. Song, "Circular chromatic number and a generalization of the construction of Mycielski," *J. Combinat. Theory B*, vol. 89, no. 2, pp. 195–205, 2003, doi: 10.1016/S0095-8956(03)00070-4.

BETÜL BOZ (Member, IEEE) was born in Ankara, Turkey, in 1980. She received the B.Sc. and M.Sc. degrees in computer engineering from Marmara University, Istanbul, Turkey, in 2002 and 2004, respectively, and the Ph.D. degree in computer engineering from Bogazici University, Istanbul, in 2011.

She is currently an Assistant Professor with the Computer Engineering Department, Marmara University. Her research interests include design of efficient operators and techniques for real-world problems using evolutionary algorithms, computational and artificial intelligence, parallel processing, and computer architecture.

GIZEM SÜNGÜ was born in Istanbul, Turkey, in 1993. She received the B.S. and M.S. degrees in computer engineering from Marmara University, Turkey, in 2015 and 2018, respectively. She is currently pursuing the Ph.D. degree in computer engineering with Gebze Technical University, Turkey.

Since 2017, she has been a Research Assistant with the Institute of Information Technologies, Gebze Technical University. Her research interests include optimization problems, such as graph coloring and path planning, evolutionary algorithms, and integer partition analysis.

Ms. Süngü was a recipient of the ACM-Woman Scholarship to present her paper at the Genetic and Evolutionary Computation Conference (GECCO), Madrid, Spain, in 2015.