Yugoslav Journal of Operations Research
22 (2012), Number 2, 225-246
DOI: 10.2298/YJOR100927020F

TWO GENETIC ALGORITHMS FOR THE BANDWIDTH MULTICOLORING PROBLEM

Jasmina FIJULJANIN

Faculty of Matematics
University of Belgrade, Belgrade, Serbia
fjasmina@yahoo.com

Received: September 2010 / Accepted: August 2012

Abstract: In this paper the Bandwidth Multicoloring Problem (BMCP) and the Bandwidth Coloring Problem (BCP) are considered. The problems are solved by two genetic algorithms (GAs) which use the integer encoding and standard genetic operators adapted to the problems. In both proposed implementations, all individuals are feasible by default, so search is directed into the promising regions. The first proposed method named GA1 is a constructive metaheuristic that construct solution, while the second named GA2 is an improving metaheuristic used to improve an existing solution. Genetic algorithms are tested on the publicly-available GEOM instances from the literature. Proposed GA1 has achieved a much better solution than the calculated upper bound for a given problem, and GA2 has significantly improved the solutions obtained by GA1. The obtained results are also compared with the results of the existing methods for solving BCP and BMCP.

Keywords: Evolutionary computation, graph coloring problem, combinatorial optimization.

MSC: 90C59, 68T20.

1. INTRODUCTION

1.1. Problem definitions

The vertex coloring problem on graphs (VCP) is a well-known NP-complete problem that has been studied extensively. The first coloring algorithms date back to the 1960s [10, 39] and since then, important progress has been made. Nowadays literature contains a great number of heuristic algorithms that belong to three main solution

approaches: sequential construction (very fast methods but not particularly efficient), metaheuristics (tabu search [3, 12, 15, 17], simulated annealing [6, 20], iterated local search [7, 9], variable neighborhood search [2, 18], genetic algorithm [11, 16]) and hybrid approaches [17, 34]).

Given an undirected graph $G(V,E)$ with vertex set V and edge set E, the classical vertex coloring problem on graphs (VCP) is to assign a color to each vertex so that no two adjacent vertices have the same color and the total number of different colors is minimized.

In the bandwidth coloring problem (BCP) distance constraints are imposed between adjacent vertices, replacing the difference constraints. A distance $d(i,j)$ is defined for each edge $\{i,j\} \in E$, and the absolute value of the difference between the colors assigned to i and j must be at least equal to this distance.

In the multicoloring problem (MCP) a positive weight $k(i)$ is defined for each vertex $i \in V$, representing the number of colors that must be assigned to vertex i, so that for each $\{i,j\} \in E$ the intersection of the color sets assigned to vertices i and j is empty. Both $d(i,j)$ and $k(i)$ are positive integers.

The bandwidth multicoloring problem (BMCP) is the combination of the above two problems. In BCP and BMCP graph can contain loops and the absolute value of the difference between each two colors assigned to a vertex i with a loop should not be smaller than $d(i,i)$.

BCP, MCP and BMCP are NP-hard because they generalize VCP. VCP instance is a BMCP instance where all the distances are equal to 1 and each vertex must receive only one color. The BCP where the distances between adjacent vertices are the same, $d(i,j) = T$ for any edge $\{i,j\} \in E$ is also known as *T – Coloring*.

BMCP can be more formally defined in the following way: find the minimal number k such that for each $i \in V$ there exists a subset $S(i) \subset \{1,2,...,k\}$ for which $|S(i)| = k(i)$ (where $|A|$ denotes the cardinality of set A), and for each $\{i,j\} \in E$, each $p \in S(i)$ and each $q \in S(j)$ it follows $|p-q| \geq d(i,j)$. BMCP has special cases: BCP (The Bandwidth Coloring Problem), MCP (The Multicoloring Problem), VCP (The Vertex Coloring Problem).

When $k(i) = 1$ for each $i \in V$ then BMCP is reduced to BCP, i.e. to the problem of finding the minimal number k such that there exists a mapping $r : V \to \{1,2,...,k\}$ such that for each $\{i,j\} \in E$, it follows $|r(i) - r(j)| \geq d(i,j)$.

In MCP, multiple colors can be assigned to each node. In this model, for graph $G(V,E)$ with node weights $k(i)$ for each $i \in V$, find a minimum k and subsets $S(i) \subset \{1,2,...,k\}$ such that $|S(i)| = k(i)$ for each $i \in V$ and $S(i) \cap S(i) = \varnothing$ for each $\{i,j\} \in E$.

In the VCP, one color is assigned to each node in the graph, and the colors for adjacent nodes must be different. For graph $G(V,E)$, find a minimum k, and a mapping $r : V \to \{1,...,k\}$ such that $r(i) \neq r(j)$ for each edge $\{i,j\} \in E$.

The Multi-Coloring Problem can be used to schedule jobs with different time requirements, where the set of colors assigned to a node corresponds to resources assigned to a job. Each node demands a set of colors to be assign to it, ensuring that its neighbors receive disjoint sets (see [28]).

The organization of exams at an university can be seen as the graph coloring problem. Each examination needs a time slot, and the university wants to organize as many examinations in parallel as possible, without exceeding the availability of classrooms, in order to reduce the number of time slots (see [30]).

The next related problem is the frequency assignment problem (FAP). It concerns the allocation of frequencies to transmitters with the aim of avoiding or minimizing interference. FAP can be considered as BMCP. BMCP corresponding to the minimum span frequency assignment problem (MS-FAP), the problem where the span, i.e. the range of frequencies, has to be minimized [28].

1.2. Previous work

From its beginning, the Vertex graph coloring problem is constantly studied and there are many methods for obtaining solutions. In this paper, only the most important methods are listed.

The sequential coloring approaches are the simplest heuristic methods for the VCP. The vertices are sorted and the top vertex is labeled (colored) with number one. The remaining vertices are considered in order. Each vertex is labeled with the first color, which has no adjacent vertices already labeled with this color. Several different schemes have been used for the initial ordering. The first who had success in solving the Graph coloring problem using this method are Welsh and Powell in 1967 [39]. These methods can be easily implemented and are fast, but often produce solutions that are far from optimal. Lim proposed in [25] several methods to adjust the method based on sequential coloring.

Different heuristic approaches were proposed for MS-FAP, like greedy algorithms, local search, tabu search, simulated annealing, constraint programming approaches and genetic algorithm. Representing real MS-FAP cases arising in telecommunications, most of these papers describe algorithms designed to solve instances with a special structure. Those are the well studied Philadelphia instances, firstly proposed by Anderson [1] in 1973. That is the multicoloring version of the problem because in all these instances n is equal to 21, and each transmitter (i.e. vertex) must receive a large number of frequencies (i.e. colors).

Computational Symposium on Graph Coloring and its Generalizations was organized in 2002 in order to promote computational research on these problems [38]. A new set of instances for VCP, BCP, MCP and BMCP were presented during the symposium.

At the computational symposium, Phan and Skiena [33] proposed to solve VCP and BCP by means of a general heuristic, called Discropt (designed for ‘‘black box optimization’’), adapted to the specific coloring problems. Also, to solve generalized graph coloring problems, Prestwich [35] proposed a combination of local search and constraint propagation in a method called FCNS (Forward Checking Neighborhood Search).

Method for solving VCP, BCP, MCP and BMCP which combine hill-climbing techniques and squeaky wheel optimization was proposed by Lim in [26].

The sequential method is an adaptation of the well-known DSATUR algorithm [5] for the VCP. In this algorithm vertices are selected at each stage based on its score or saturation degree — the number of distinctly colored adjacent vertices. A vertex with the maximum saturation degree is selected and labeled with the first legal color. Combination of DSATUR and tabu search methodologies was proposed by Malaguti and Toth in [29].

One of the important methods for solving the graph coloring problems is GRASP (Greedy Randomized Adaptive Search Procedure). GRASP is an iterative process where each iteration consists of two phases: construction and local search. The construction phase builds a feasible solution, whose neighborhood is explored until a local optimum is found after the application of the local search phase. The detailed description of GRASP and its properties is out of this paper scope and can be found in [37]. GRASP in combination with tabu search is used by Marti, Gortazar and Duarte in [31] for solving the BCP.

It is known that the graph set T-coloring problem (GSTCP) generalizes the classical graph coloring problem. In [8], they presented an experimental study of local search algorithms for solving general and large size instances of the GSTCP.

In [13], for solving BCP and BMCP an approximation algorithm based on Irregular Cellular Learning Automata is proposed. The multicoloring problem is first simplified as a vertex coloring problem, where each vertex is colored by only one color. Learning automaton is assigned to each vertex of the resultant graph. At each stage, each learning automaton randomly chooses its action according to its action-probability vector. The proposed algorithm is repeated until all cells are rewarded.

In [29], some generalizations of the vertex coloring problem, where distance constraints are imposed between adjacent vertices (Bandwidth Coloring Problem - BCP) and each vertex has to be colored with more than one color (Bandwidth Multicoloring Problem - BMCP) are considered. An evolutionary metaheuristic approach for the first problem, combining an effective tabu search algorithm with population management procedures is proposed. After a simple transformation, the approach can be applied to the second problem.

Generalization of the graph coloring problem assumes that a strictly positive integer weight ω_i is associated with each vertex $i \in V$. An interval coloring of G is a function I that assigns an interval $I(i)$ of ω_i consecutive integers (called colors) to each vertex $i \in V$ so that $I(i) \cap I(i) = \varnothing$ for all edges $\{i, j\} \in E$. The interval coloring problem means to determine an interval coloring that uses as few colors as possible. In [4] Bouchard, Čangalović and Hertz proved that an optimal solution of the interval coloring problem can be obtained by solving a series of bandwidth coloring problems. Computational experiments got demonstrate that such a reduction can help in solving larger instances or obtaining better upper bounds on the optimal solution value of the interval coloring problem.

The paper is organized as follows: Section 2 describes mathematical formulation of considered problem. Section 3 presents a short description of Genetic algorithms. Section 4 describes constructive metaheuristic, named GA1, which construct solution, while in Section 5, an improving metaheuristic named GA2 is given. Computational

experiments on BCP and BMCP instances from the literature are presented in Section 6. Section 7 resumes the results of this work.

2. MATHEMATICAL FORMULATION

In [28] we can see an Integer Linear Programing (ILP) for BCP. In this case, more than n colors may be needed. Let $H = \{1,2,...,t\}$ be the set of available colors (where t represents an upper bound on the value of the maximum color used). Consider the binary variable x_{ih} having value 1 if vertex i is colored with color h, $h \in H$, otherwise $x_{ih} = 0$, and the binary variables y_h having value 1 if color h is used, otherwise $y_h = 0$. Then the model reads:

$$\min k \tag{1}$$

$$k \geq y_h \cdot h \qquad h \in H \tag{2}$$

$$\sum_{h \in H} x_{ih} = 1 \qquad i \in V \tag{3}$$

$$x_{ih} + x_{jl} \leq 1 \quad \{i,j\} \in E,\ h \in H,\ l \in \{h - d(i,j) + 1, ..., h + d(i,j) + 1\} \tag{4}$$

$$x_{ih} \leq y_h \qquad i \in V, h \in H \tag{5}$$

$$x_{ih} \in \{0,1\} \qquad i \in V,\ h \in H \tag{6}$$

$$y_h \in \{0,1\} \qquad h \in H \tag{7}$$

The objective function (1) in conjunction with constraints (2) asks for minimizing the maximum color used. Constraints (3) show that every vertex i in the graph must receive one color. Constraints (4) state that the absolute value of the difference between the colors assigned to vertices i and j must be at least equal to $d(i,j)$. Constraints (5) assure that if a vertex i uses a color h, the color h results as used. The integrality of the variables imposes (6) and (7).

3. GENETIC ALGORITHMS

Genetic algorithms (GAs) are problem-solving metaheuristic methods rooted in mechanisms of evolution and natural genetics. In the last three decades, GAs have emerged as effective, robust optimization and search methods. The main idea was introduced by John Holland [19].

The GA work with a population of individuals, each representing a possible encoded solution to a given problem. The representation is the genetic code of an individual and it is often a binary string, although other alphabets or higher cardinality can be used.

Initial population is either randomly or heuristically generated. The individuals in the population pass through a procedure of simulated “evolution” by means of randomized processes of selection, crossover and mutation. To determine qualities of

individuals from current population, a fitness function is used. Higher chances for survival and reproduction have the individuals with higher fitness value. Best-fitted individuals are selected in different ways. The most often used is tournament selection.

Recombination of genetic material by exchanging portions between the parents, with the chance that good solutions can generate even better ones, provides the crossover operator. Sporadic and random changes that modify individual`s genetic material with some small probability cause mutation. Mutation should prevent premature convergence of the GA to suboptimal solutions.

There are different policies for generation replacement. Certain numbers of individuals (elite individuals) may skip selection or even all genetic operators going directly into the next generation. This approach is named the steady-state generation replacement policy with elitist strategy. It preserves good individuals from the past generations.

Implementing the GA usually involves the following steps: evaluating the fitness of all individuals in a population, selecting the best-fitted individuals and creating a new population by performing crossover and mutation operators although there can be many modifications of the GA.

Experimental results on various optimization problems show that GA often produces high quality solutions in a reasonable time [21, 24, 32].

Algorithm 1: The scheme of the GA implementation

```
Input()
Init()
while not FinishGA() do
    for i := (N_elite + 1) to N_pop do
        if Contain (cache, i) then
            obj_i := From_Cache(i)
        else
            obj_i := Objective_Function(i)
            Put_Into_Cache(i, obj_i)
            if Full(cache) then
                Remove_LRU_Block()
            endif
        endif
    endfor
    Fitness_Function()
    Selection()
    Crossover()
    Mutation()
endwhile
Output()
```

In Algorithm 1, the general outline of GA implementation is given. N_{pop} denotes the overall number of individuals in the population, N_{elite} is a number of elite individuals, and i and obj_i are the individual and its objective value, respectively (also can be seen in [22, 23]). Here, evaluated objective values are stored in a hash-queue data structure, created by the use of Least Recently Used (LRU) caching strategy. When the same code

is obtained again, its objective value is taken from the hash-queue table, instead of recalculating its objective function.

Caching technique investigates whether the Cache memory contains individual. In that case, objective value is directly taken from the Cache memory, otherwise, the objective value is calculated and the pair (individual, objective value) is stored in the Cache memory (**Put_Into_Cache(i, obj$_i$)**). If the Cache memory is full (**if Full(cache)**), in order to make space for the new entry, the last recently used block is removed (**Remove_LRU_Block()**) from cache memory.

4. GA1 IMPLEMENTATION

In GA1, the maximal number of possible colors are fixed during the search process. That number is obtained by a greedy algorithm before search process. Each individual in GA1 is a sequence of integer numbers. The fitness function value is defined as the maximal color used in the corresponding coloring. GA1 implementation will be described only for general problem BMCP, as for all node weights equal to 1 it can be applied to BCP.

4.1. Greedy algorithm for computing upper bound

Greedy algorithm takes a sequence of 'split nodes' and assign colors to them greedily. Colors are assigned one by one for each node. The details of this algorithm are presented in Algorithm 2.

Algorithm 2 Greedy Algorithm which Assigns Colors

```
for i:= 1 to m do
c[i] := −1
end for
for i := 1 to m do
    forbidden_set := ∅
      u := p[i]
      v := get_original_node(u)
      for each node s that adjacent to u do
            if c[s] ≠ −1 then
                t := get original node(s)
                a := Max{0, c[s] − d(v, t) + 1}
                b :=c[s] + d(v, t) − 1
                forbidden_set := forbidden_set ∪[a..b]
            end if
      end for
      c[u] := Min{r, r∈N, r∉ forbidden_set}
end for
```

Here, n is the number of nodes in the original graph; $d(i, j)$ is the weight of edge $\{i, j\}$, m is the number of nodes after splitting; $p[1..m]$ is the priority sequence for

the m nodes ($p[1]=u$ implies node u has the highest priority); $c[1..m]$ records the assigned colors, and the function gets the original node(u) to be returned to the node v in the original graph such that u is split from v. This is a standard greedy procedure, which also can be found in [27].

Example 1: This example shows how to find the maximum number of colors that can be used by the Greedy algorithm in the case of the graph from **Figure 1**.

Greedy algorithm:

I node: two colors on minimum distance 2 between each other: 1, 3

II node: three colors on minimum distance 2 between each other and distance at least 1 from colors of I node: 2, 4, 6

III node: one color on distance at least 2 from colors of I node (1,3) and on distance at least 3 from colors of II node (2,4,6): 9

IV node: two colors on minimum distance 1 between each other, on distance at least 2 from colors of II node (2,4,6) and on distance at least 2 from colors of III node (9): 11, 12.

The maximum number of colors that can be used is 12.

4.2. Representation and objective function

The proposed GA1 first counts the maximum number of colors that can be used, (this is also found in the master thesis in [14]). The number of colors is obtained using the greedy algorithm. Than for each node subsidiary colors are assigned. Instead on the original graph, this algorithm is applied to an auxiliary graph obtained from the original one by replacing each node i, where $k(i)>1$, with the complete subgraph on $k(i)$ nodes and all edge distances equal to $d(i,i)$ (in the case when there is no a loop at node, $d(i,i)=1$).

The encoding scheme is defined in the following way: greedy algorithm getting the maximum number of colors that can be used, let that number be t. The code in GA1 is defined for a given ordering of nodes as a sequence of $\sum_{i\in V} k(i)$ integers from the set $\{1,2,...,t\}$. In such a code, numbers on positions from $k(i-1)+1$ to $k(i-1)+k(i)$ correspond to node i where $k(0)=0$.

'Decoding' technique, which for the given code finds the corresponding feasible coloring of the graph, can be described as follows:

***Step 1*:** for the current node $i\in V$ and each $j\in\{1,2,...,k(i)\}$, the technique finds a sequence of all colors from $\{1,2,...,t\}$ that are "free" to be assigned to vertex i, i.e. the colors which are not in conflict with those already assigned. (A color a of vertex i is not in conflict with already assigned colors if $|a-p|\geq d(i,l)$ for each already colored vertex l adjacent to i and all colors p associated to l. If $i=l$ and there is not a loop at i, then $d(i,i)=1$). Then, the technique considers gene r on position $k(i-1)+j$ in the individual code and assigns the r-th color from the sequence of free colors to vertex i.

If for a current sequence C of colors $r > |C|$, then the technique assigns to vertex i the $r \bmod |C|$-th color from C.

If $|C| = 0$ or in the current C, it is impossible to find the *r*-the color which is not in conflict with already assigned colors, the code is incorrect and the corresponding individual is eliminated from the current generation.

***Step 2*:** The technique updates a set of free colors and goes to *Step 1* to consider the next node from *V* as the current one. Otherwise, if the considered node is the last one, the technique stops.

The fitness function of an individual is the maximal color used in the coloring obtained by the decoding technique applied to this individual.

Example 2: This example shows how the proposed GA1 decodes a given code for BMCP on the graph from Figure 1.
BMCP is a graph coloring problem where to each node a positive number is assigned that represents the number of colors that must be assigned to each node respecting the given distance $d(i, j)$ between colors assigned to adjacent nodes, and respecting the given distance $d(i, i)$ between colors assigned to the same node.

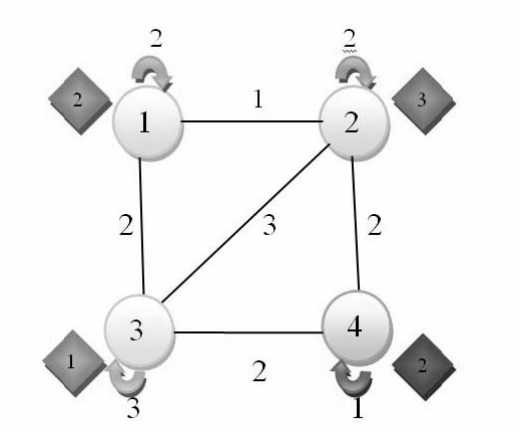

Figure 1: Graph for BMCP

In Example 1, greedy algorithm determinates the maximum number of colors that can be used.

Now, when we know that the number of colors that can be used is 12, for the genetic code: 4, 3, 3, 2, 1, 3, 1, 1, the colors proposed by GA1 will be assigned in the following way:

I node:

I color: all colors are free: 1,2,3,4,5,6,7,8,9,10,11,12, 4-th free color is 4

II color: free colors are: 1,2,3,5,6,7,8,9,10,11,12, 3-rd free color on distance at least 2 from I color is 6

II node:

I color: free colors are: 1,2,3,5,7,8,9,10,11,12, 3-rd free color on distance at least 1 from colors of I node is 3

II color: free colors are: 1,5,7,8,9,10,11,12, 2-nd free color on distance at least 2 from I color and on distance at least 1 from colors of I node is 5

III color: free colors are: 1,7,8,9,10,11,12, 1-st free color on distance at least 2 from first two colors and on distance at least 1 from colors of I node is 1

III node:

I color: free colors are: 8,9,10,11,12, 3-rd free color (on distance at least 2 from colors of I node and on distance at least 3 from colors of II node) is 10

IV node:

I color: free colors are: 7,8,12, 1-st free color (on distance at least 2 from colors II and III nodes) is 7

II color: free colors are: 8,12, 1-sr free is 8.

So, c(1)=4,6; c(2)=3,5,1; c(3)=10; c(4)=7,8 fitness function selects the maximum of these values, that is 10, which means that 10 colors is sufficient for graph coloring.

4.3. Genetic Operators

Selection operator: According to individual value of fitness function, the selection operator chooses the individuals that will produce offspring in the next generation. Low fitness-valued individuals have less chance to be selected than high fitness-valued ones. Tournaments are imposed competitions between two or more individuals who will participate in the next generation. The size of a tournament is a given integer parameter, which in same cases can be a limiting factor. Tournament candidates are randomly chosen from the current population. Only the winner of the tournament can participate in the crossover. The selection operator is applied N_{nnel} times on the set of all N_{pop} individuals in the population to choose the N_{nnel} parents for crossover.

In proposed GA1, an improved tournament selection operator is used, known as the fine-grained tournament selection – FGTS. This operator uses a real (rational) parameter F_{tur} which denotes the desired average tournament size. The first type of tournaments is held k_1 times and its size is $[F_{tur}]$, while the second type is performed k_2 times with $[F_{tur}] + 1$ individuals participating, so $F_{tur} \approx \frac{k_1 \cdot [F_{tur}] + k_2 \cdot [F_{tur} + 1]}{N_{nnel}}$.

In the GA1 implementation F_{tur} is set to 5.4 value. FGTS is applied to $N_{nnel} = 50$ non-elitist individual, tournaments are held $k_1 = 30$ and $k_2 = 20$ times with sizes 5 and 6, respectively. Running time for the FGTS operator is $O(N_{nnel} \cdot F_{tur})$. In practice, N_{nnel} and F_{tur} are considered to be constant, not depending on problem size, so overall time complexity of selection operator is constant.

Crossover operator: All non-elitist individuals chosen to produce offspring for the next generation are randomly paired for exchanging genes in $[N_{nnel}/2]$ pairs. Crossover operator is applied to selected parents producing two offspring. In the proposed GA1, the one – point crossover is used. If the length of genetic code of parents is l, then we choose positive number k from the interval [0, l-1]. All genes in parent`s

genetic codes starting from $k+1$ to $l-1$ exchange their positions. Probability of the crossover operator is $p_{cross} = 0.85$. It means that approximately 85% pairs of individuals exchange their genetic material.

Mutation operator: During the GA1 execution, it may happen that all individuals in the population have the same gene on a certain position. This gene is called frozen. If the number of frozen genes is *l*, the search space becomes *l*! time smaller and the possibility of a premature convergence rapidly increases. The selection and crossover operators cannot change the bit value of any frozen gene. The basic mutation rate is often too small to restore lost subregions of the search space. If the basic mutation rate is increased significantly, a genetic algorithm becomes a random search.

Mutation operator is modified so that frozen genes are mutated with one mutation level and non-frozen genes with another. If *n* is the total number of used colors then:

non-frozen genes: first gene is mutated with 0.2/*n*, second with 0.1/*n*, etc.

frozen genes: gene is mutated with 0.5/*n*, second with 0.25/*n*, etc.

So, we can conclude that the level of mutation for frozen genes is 2.5 time higher comparing to non-frozen ones.

4.4. Other GA1 aspects

In the GA1, initial population, which has 150 individuals randomly selected, provides maximal diversity of genetic material. One-third of the population, i.e. 50 individuals, are non-elite and the rest (100 of them) are elite individuals. In GA1 implementation, so called elitist strategy is applied that enables direct passing of the elite individuals into the next generation. The genetic operators are applied to the rest of the population.

In order to avoid premature convergence and to provide the maximal diversity of genetic material, duplicated individuals are removed from the population. Setting their fitness value to zero, they lose the opportunity to appear in the next generation.

If the individuals with the same fitness value that have similar genetic code are dominant in the population, the possibility of a premature convergence rapidly increases. For that reason, it is useful to limit the number of their appearance to some constant $N_{rv.}$ In GA1 implementation, the maximal allowed number of individuals with the same objective value is $N_{rv} = 40$.

4.5. Caching GA

The main purpose of caching is to avoid recalculating objective values of individuals appearing again during the GA run, and to optimize run-time performance of GA1. Evaluated objective values are stored in a hash-queue data structure, created by Least Recently Used (LRU) caching strategy. If the same code is obtained again, its objective value is taken from the hash-queue table, instead of recalculating its objective function.

Caching technique investigates whether the cache memory contains an individual. In that case, objective value is directly taken from the cache memory. Otherwise, the objective value is calculated and the pair (individual, objective value) is

stored in the cache memory. If the cache memory is full, in order to make space for the new entry, the last recently used block is removed from it.

The number of cached objective values in hash-queue table is limited to $N_{cache} = 5000$ in the GA1 implementation ([14]).

5. GA2 IMPLEMENTATION

The experimental results show that GA1 (constructive strategy) is not time consuming, but the solutions are quite different from those, so far, best known in the literature. Therefore, in this section, GA2 (improving strategy) will be presented that improves the existing solutions. Note that GA2 does not need complete solution but only solution value, using it as an upper bound. Therefore, GA2 can be used on any upper bound given in the literature. But for the sake of a fair comparison, in this paper GA2 is used only to improve solutions given by GA1 (or its own solution value).

In GA2, genetic code is of the same length as that of GA1, but the objective function is defined in a totally different way.

5.1. Encoding and objective function

Each code is defined as in GA1, and the only difference is that it contains integers only from $\{1, 2, ..., n_c\}$, where n_c is the total number of the currently used colors. Unlike GA1, where the objective function represents the maximum number of used colors, in GA2 implementation number n_c of used colors is fixed and initialized to a given upper bound minus one, and objective function represents the total level of infeasibility, *infeas*.

There are two types of the infeasibility levels for a vertex: with respect to its adjacent vertices, and with respect to this vertex itself, what will be illustrated in Example 3.

Example 3 Let u, v be nodes; $c(u)$, $c(v)$ represent the colors assigned to the nodes u and v; $d(u,v)$ the minimum distance between colors assigned to nodes u and v

$c(u) = 5$
$c(v) = 7$ => *infeas* = 2 ($c(u)$ and $c(v)$ are on distance 2 and according
$d(u,v) = 4$ $d(u,v) = 4$, they should spread 2, so *infeas* = 2)
$c(u) = 5$
$c(v) = 5$ => *infeas* = 4 ($c(u)$ and $c(v)$ are on distance 0 and according
$d(u,v) = 4$ $d(u,v) = 4$, they should spread 4, so *infeas* = 4)

As GA2 is developed for BMCP, infeasibility can occur with respect to a vertex itself, i.e. if this vertex has the weight greater than 1, some of its colors can be the same. So, if for vertex u its weight $k(u) = 3$ and colors 3, 5, 5 are associated to this vertex, then the infeasibility level of u with respect to itself is equal to 1(*infeas* = 1).

Before the GA2 execution, the number of colors, n_c in use, must be given. It can be done in two ways:

1) n_c takes the value of greedy algorithm`s upper bound minus 1
2) n_c takes the value of GA1 solution minus 1

For gene value r from the genetic code, a corresponding color is taken. If r is at position $k(i-1)+j$ of the code, then r is the j- th color assigned to vertex i in the coloring corresponding to the code. If $r > n_c$, then we modify r to a random number from 1 to n_c, update that gene in the genetic code and assign that color to the node i. For each node, we count the infeasibility level with respect to its adjacent nodes as well as to the node itself. Then the objective function value *infeas* of the genetic code is equal to the sum of infeasibility levels over all the nodes.

More formally, let $S(i),\ i \in V$, be sets of colors assigned to vertices i in the coloring defined by an individual code. Then, the infeasibility level $infeas(i)$ of vertex i is the sum of values $\max\{0, d(i,j)-|p-q|\}$ over all vertices j adjacent to i, $i \neq j$, and all $p \in S(i)$ and all $q \in S(j)$, and, in case when $k(i) > 1$, of values $\max\{0, d(i,i)-|p-q|\}$ over all $p, q \in S(i)$, where $d(i,i) = 1$ if there is not a loop at vertex i. Let us mention that, for $k(i) > 1$, it is supposed that some of $k(i)$ members in $S(i)$ can be equal. Now, the objective function value for the individual code could be defined as $\sum_{i \in V} infeas(i)$.

When, in the population, the objective value of an individual becomes zero, it means that a new solution is obtained. This new solution is, by default, better than the previous, so it should be saved. Then the number of used colors n_c is subtracted by $1 (n_c = n_c - 1)$.

Now, when n_c is subtracted by 1 for all individuals in the population (elite and non-elite) objective value must be computed again. From these reasons, in GA2, caching technique is not used.

5.2. Local search

In order to reduce the level of infeasibility, local search is applied. For each node, we try to change one of the assigned colors. The objective value for particular individual is remembered, and the level of infeasibility subtracted from that value only for that node. This is done in such a way that for particular node we are looking adjacent nodes.

If in the coloring defined by an individual code color, $p \in S(i)$, already assigned to a vertex i, is replaced with the color $k \in \{1, 2, ..., n_c\}$, $k \neq p$, then the new objective function value $infeas(i,k)$ is equal to

$$infeas(i,k) = s + \sum_{\substack{j:\{i,j\} \in V \\ i \neq j}} \sum_{q \in S(j)} \max\{0, d(i,j) - |k-q|\} + \sum_{q \in S(i) \setminus \{p\}} \max\{0, d(i,i) - |k-q|\}$$

where

$$s = infeas(i) - \sum_{\substack{j:\{i,j\}\in V \\ i\neq j}} \sum_{q\in S(j)} \max\left\{0, d(i,j) - |p-q|\right\} - \sum_{q\in S(i)\setminus\{p\}} \max\left\{0, d(i,i) - |p-q|\right\}$$

If $infeas(i,k^*) = \min\limits_{k\in\{1,\ldots,n_c\}} infeas(i,k)$ and $infeas(i,k^*) < infeas(i)$, then the objective function value of the individual code can be improved by replacing color p of vertex i with color k^*.

For given vertex i and its color p, value $infeas(i,k^*)$ can be obtained by a procedure which determines a sequence $z(k), k = 1,2,\ldots,n_c$, as follows:

Initially, $z(k) = s$ for each $k \in \{1,2,\ldots,n_c\}$.

For each vertex j adjacent to vertex i, $j \neq i$, and each $q \in S(j)$, values of $z(k)$ are transformed in the following way:

For $\max\{1, q - d(i,j) + 1\} \le k \le \min\{q + d(i,j) - 1, n_c\}$,

$z(k) = z(k) + \max\{0, d(i,j) - |k-q|\}$, while for other k value, $z(k)$ is not changed.

If $k(i) > 1$, then for each $q \in S(i)$, values of $z(k)$ are additionally transformed such as for $\max\{1, q - d(i,i) + 1\} \le k \le \min\{q + d(i,i) - 1, n_c\}$

$z(k) = z(k) + \max\{0, d(i,i) - |k-q|\}$, while for other k, value $z(k)$ is not changed.

The final value of $z(k)$ obtained by this procedure is equal to $infeas(i,k)$ for each $k \in \{1,2,\ldots,n_c\}$. Therefore, $infeas(i,k^*) = z(k^*) = \min\limits_{k\in\{1,\ldots,n_c\}} z(k)$.

If there is no improvement when color p of vertex i is replaced with color k^*, the local search procedure continues with a new vertex.

The resulting value entered in genetic code and for the following similar individual, the value doesn`t have to be counted again. In this way, we are saving time. In the second case, we see that by changing one color, we do not get a better solution (ie. this is a local minimum).

Example 4: Let $c(2) = 5, c(3) = 4, d(2,3) = 3$ $ov = 15$, $infeas(2,3)=2$

For node 3 without color 5, the total level of infeasibility is 13, candidates for colors of node 2 are:

1, 2, 3, 4, 5, 6, 7, 8, …, 20.

At the beginning, members of the series z are all the same.

z: 13, 13, 13, 13, 13, 13, 13, 13, …

Series z for the given values is changing in the following way:

z: 13, 14, 15, 16, 15, 14, 13, 13, …

Now, consider all edges where node 2 takes part. Let, for example, $c(5)=2$, $d(2,5)=4$. For these values, we have

z: 16, 18, 18, 18, 16, 14, **13**, **13**, …

Now, we are looking for minimum of the series z. That is 13, so we take color 7 or color 8.

5.3. Genetic Operators

The FGTS selection operator was also used in GA2 method, with the same value of F_{tour} parameter ($F_{tur} = 5.4$).

GA2 uses one – point crossover like GA1 with the same probability $p_{cross} = 0.85$. Mutation operator is the same as for the GA1, but the mutation rates are different. Note that, unlike GA1, mutation rates are constant. If n is the total number of used colors, then:

non-frozen genes: genes are mutated with $0.4/n$
frozen genes: genes are mutated with $1.0/n$

5.4. Other GA2 aspects

Other aspects of GA2 are the same as for GA1:

- The population size is 150 individuals
- In GA2 is also applied elitist strategy
- One hundred elite individuals and 50 non-elite ones
- Duplicated individuals are removed from population
- Limiting the number of individuals with the same objective value ($N_{rv} = 40$)

In GA2 the caching is not applied, because the objective function depends on the number of used colors.

6. EXPERIMENTAL RESULTS

This section presents the computational results for BCP and BMCP. The GAs tests were performed on an Intel Core 2 Duo 3.0 GHz with 4 GB memory, under Windows 7 operating system. The stopping criterion was the maximum number of generations equal to 2000, or at most 5000 generations without improvement of the objective value. Both GAs have been run 20 times for each instance, and the results are summarized in Table 1(for BCP) and Table 2 (for BMCP).

The algorithms were coded in C programming language. The GAs are tested on GEOM instances (presented in [38]) available in the literature. A characteristic of these instances is that the number of graphs nodes are contained in their name. Note that the optimal solutions for these problems are not known in literature.

The tables (Table 1 and Table 2) contain the following data, by columns:

- the first three columns contain the test instance name, the number of nodes (n) and edges (m), respectively;
- the fourth column *Greedy* contains solution obtained by greedy algorithm

- the fifth column *GA1* shows solution obtained by proposed GA1 ;
- the average running time *t* is given in the sixth column ;
- the seventh column *GA2* shows solution obtained by proposed GA2 ;
- the average execution time *t* is given in the eighth column ;
- GA2 is applied once again, using solution values from the seventh column minus one as upper bounds. In the last two columns, *GA2a* and *t* is presented given solution values and corresponding running times.

Table 1 Results for BCP

Inst	*n*	*m*	*Greedy*	*GA1*	*T* (sec)	*GA2*	*t* (sec)	*GA2a*	*t* (sec)
GEOM20	20	40	25	21	0.682	21	0.05	21	0.01
GEOM20a	20	57	28	21	1.196	20	0.03	20	0.11
GEOM20b	20	52	17	14	0.604	13	2.62	13	1.36
GEOM30	30	80	34	32	0.916	28	0.13	28	0.06
GEOM30a	30	111	32	28	2.106	27	0.20	27	0.31
GEOM30b	30	111	28	26	1.249	26	26.24	26	2.27
GEOM40	40	118	34	30	1.66	28	0.09	28	0.08
GEOM40a	40	186	49	40	3.572	37	12.12	37	0.83
GEOM40b	40	197	41	36	2.131	33	72.18	33	8.78
GEOM50	50	177	34	30	4.382	28	1.26	28	1.28
GEOM50a	50	288	60	53	5.796	50	2.79	50	7.36
GEOM50b	50	299	55	43	3.434	37	120.64	35	14.33
GEOM60	60	245	41	34	5.189	33	0.91	33	3.09
GEOM60a	60	339	61	55	7.24	51	123.79	50	17.20
GEOM60b	60	426	64	51	5.346	43	192.18	43	21.34
GEOM70	70	337	47	42	6.413	38	125.82	38	15.42
GEOM70a	70	529	73	65	9.424	63	5.32	62	6.38
GEOM70b	70	558	77	60	7.478	50	232.91	49	55.36
GEOM80	80	429	52	45	8.768	41	0.51	41	18.67
GEOM80a	80	692	80	69	13.983	66	273.72	65	47.33
GEOM80b	80	743	99	73	10.938	65	479.78	65	40.05
GEOM90	90	531	54	49	9.683	46	1.70	46	5.25
GEOM90a	90	879	81	75	17.239	68	305.19	67	56.30
GEOM90b	90	950	110	86	16.493	75	863.86	75	57.25
GEOM100	100	645	58	55	12.501	52	2.57	50	4.97
GEOM100a	100	1092	97	83	20.564	74	370.76	72	48.77
GEOM100b	100	1150	122	93	17.312	78	588.82	77	119.77
GEOM110	110	748	58	57	13.444	52	259.95	50	45.84
GEOM110a	110	1317	98	88	28.91	80	629.47	77	140.05
GEOM110b	110	1366	122	98	25.194	85	848.37	85	94.39
GEOM120	120	893	73	66	17.115	60	31.59	60	36.47
GEOM120a	120	1554	108	100	29.815	88	244.67	87	152.63
GEOM120b	120	1611	124	106	27.951	92	1706.23	91	166.95

From the results presented in Table 1 and Table 2, it can be seen that GA1 has achieved a much better solution than the calculated upper bound for a given problem. Executing of GA1 was relatively short, where even the large-scaled problem instances worked less than half an hour.

Table 2 Results for BMCP

Inst	*n*	*m*	*Greedy*	*GA1*	*t* (sec)	*GA2*	*t* (sec)	*GA2a*	*t* (sec)
GEOM20	20	40	195	150	14.074	149	93.06	149	123.86
GEOM20a	20	57	201	178	18.459	171	132.66	170	157.88
GEOM20b	20	52	47	45	1.708	44	10.17	44	10.11
GEOM30	30	80	214	168	27.519	161	248.31	160	153.66
GEOM30a	30	111	292	234	70.929	214	470.17	212	429.24
GEOM30b	30	111	88	80	5.87	77	35.53	77	34.45
GEOM40	40	118	226	180	60.739	170	518.30	168	392.48
GEOM40a	40	186	297	246	109.84	223	1148.09	215	994.56
GEOM40b	40	197	121	82	8.754	75	82.92	75	96.44
GEOM50	50	177	271	243	113.607	227	617.63	226	823.41
GEOM50a	50	288	440	370	309.143	323	2392.80	323	1739.73
GEOM50b	50	299	126	99	14.258	88	129.22	88	129.78
GEOM60	60	245	312	268	166.331	262	818.02	258	902.33
GEOM60a	60	339	460	399	417.723	368	3276.33	365	3542.70
GEOM60b	60	426	161	134	26.998	121	286.94	120	196.25
GEOM70	70	337	374	304	231.641	275	2823.52	273	1784.50
GEOM70a	70	529	577	508	601.959	480	2646.17	476	3438.47
GEOM70b	70	558	176	142	36.051	126	377.66	126	301.05
GEOM80	80	429	513	421	528.526	391	2721.73	390	1857.09
GEOM80a	80	692	461	411	620.676	374	3269.56	374	3064.03
GEOM80b	80	743	179	162	50.136	141	454.75	140	946.39
GEOM90	90	531	418	363	569.227	335	2669.81	335	2635.58
GEOM90a	90	879	485	430	845.052	380	3796.58	380	4154.56
GEOM90b	90	950	210	180	80.652	158	529.51	155	514.05
GEOM100	100	645	503	449	870.236	425	3070.92	421	3261.30
GEOM10a	100	1092	626	541	1399.933	460	5856.17	458	9502.08
GEOM100b	100	1150	243	198	88.188	169	1045.89	168	603.34
GEOM110	110	748	483	434	957.523	397	4365.64	387	4465.73
GEOM110a	110	1317	662	585	2485.472	502	12105.00	501	10054.41
GEOM110b	110	1366	297	240	132.244	215	1257.47	209	1232.95
GEOM120	120	893	515	458	1174.291	420	5543.78	412	6226.78
GEOM120a	120	1554	702	649	2711.486	561	16376.59	559	19254.63
GEOM120b	120	1611	268	225	153.825	196	906.84	196	857.13

GA2 improve the solutions obtained by GA1 for almost all instances. Note that the time required to obtain the result using GA2 is much longer then for GA1. The reason is relatively time consuming procedure of local search in GA2. A very interesting feature of applying GA2 once again is the improvement of many solution values. The explanation can be that because GA2 uses only upper bound value, not complete solution, so the initial solution in the next GA2 running is usually completely different from the last GA2 final solution.

Although the optimal solution for these instances is not known so far, from experimental results presented in Tables 1 – 4, it can be concluded that the proposed approaches are very successful in solving the large-scale problem instance for graphs up to 120 vertices and up to 1611 edges.

Table 3 Comparison with the most effective heuristic algorithms for the BCP

Inst	*best*	*GA1*	*GA2*	*GA2a*	*Lim03* [26]	*Phan-Skiena*[33]	*Prest 05* [36]	*Malaguti 08* [29]	*Greedy*
GEOM20	20	21	21	21	21	20	21	21	25
GEOM20a	20	21	20	20	22	20	20	20	28
GEOM20b	13	14	13	13	14	13	13	13	17
GEOM30	27	32	28	28	29	27	28	28	34
GEOM30a	27	28	27	27	32	27	27	27	32
GEOM30b	26	26	26	26	26	26	26	26	28
GEOM40	27	30	28	28	28	27	28	28	34
GEOM40a	37	40	37	37	38	38	37	37	49
GEOM40b	33	36	33	33	34	36	33	33	41
GEOM50	28	30	28	28	28	29	28	28	34
GEOM50a	50	53	50	50	52	54	50	50	60
GEOM50b	35	43	37	35	38	40	35	35	55
GEOM60	33	34	33	33	34	34	33	33	41
GEOM60a	50	55	51	50	53	54	50	50	61
GEOM60b	41	51	43	43	46	47	43	41	64
GEOM70	38	42	38	38	38	40	38	38	47
GEOM70a	61	65	63	62	63	64	62	61	73
GEOM70b	48	60	50	49	54	54	48	48	77
GEOM80	41	45	41	41	42	44	41	41	52
GEOM80a	63	69	66	65	66	69	63	63	80
GEOM80b	60	73	65	65	65	70	61	60	99
GEOM90	46	49	46	46	46	48	46	46	54
GEOM90a	63	75	68	67	69	74	64	63	81
GEOM90b	70	86	75	75	77	83	72	70	110
GEOM100	50	55	52	50	51	55	50	50	58
GEOM100a	68	83	74	72	76	84	68	68	97
GEOM100b	73	93	78	77	83	87	73	73	122
GEOM110	50	57	52	50	53	59	50	50	58
GEOM110a	72	88	80	77	82	88	73	72	98
GEOM110b	78	98	85	85	88	87	79	78	122
GEOM120	59	66	60	60	62	67	60	59	73
GEOM120a	84	100	88	87	92	101	84	84	108
GEOM120b	84	106	92	91	98	103	86	84	124

The experiments performed with the evolutionary algorithms (proposed in Sections 3 and 4) are described and compared with the most effective algorithms proposed in the literature on the BCP and BMCP instances. The corresponding computational results are reported, in Tables 3 and 4, respectively. The first two columns of the tables report the instance name and the corresponding best published solution value (‘‘best’’). The next three columns represent solution obtained by proposed GA1 (“*GA1*”), GA2 (“*GA2*”) and solutions obtained by executing GA2 several times (“*GA2a*”). The sixth column (“*Lim03*”) represents the solutions obtained by Lim [26], where he combined hill-climbing techniques and squeaky wheel optimization. The algorithm works in optimization version (i.e. it does not require as input the maximum color k to be used).

Table 4 Comparison with the most effective heuristic algorithms for the BMCP

Inst	*best*	*GA1*	*GA2*	*GA2a*	*Lim03* [26]	*Lim05*[25]	*Prest 05* [36]	*Malaguti 08* [29]	*Chard* [8]	*Greedy*
GEOM20	149	150	149	149	149	149	149	149	-	195
GEOM20a	169	178	171	170	169	169	170	169	-	201
GEOM20b	44	45	44	44	44	44	44	44	44	47
GEOM30	160	168	161	160	160	160	160	160	-	214
GEOM30a	209	234	214	212	211	209	214	211	209	292
GEOM30b	77	80	77	77	77	77	77	77	77	88
GEOM40	167	180	170	168	167	167	167	167	-	226
GEOM40a	213	246	223	215	214	213	217	215	214	297
GEOM40b	74	82	75	75	76	74	74	74	74	121
GEOM50	224	243	227	226	224	224	224	225	-	271
GEOM50a	315	370	323	323	326	318	323	320	315	440
GEOM50b	83	99	88	88	87	87	86	83	84	126
GEOM60	258	268	262	258	258	258	258	258	258	312
GEOM60a	356	399	368	365	368	358	373	363	356	460
GEOM60b	114	134	121	120	119	116	116	114	115	161
GEOM70	267	304	275	273	279	273	277	270	267	374
GEOM70a	469	508	480	476	478	469	482	473	478	577
GEOM70b	117	142	126	126	124	121	119	119	119	176
GEOM80	382	421	391	390	394	383	398	388	382	513
GEOM80a	360	411	374	374	379	379	380	370	360	461
GEOM80b	139	162	141	140	145	141	141	141	139	179
GEOM90	332	363	335	335	335	332	339	334	333	418
GEOM90a	377	430	380	380	382	377	382	384	377	485
GEOM90b	144	180	158	155	157	157	147	146	147	210
GEOM100	404	449	425	421	413	404	424	412	404	503
GEOM100a	437	541	460	458	462	459	461	452	437	626
GEOM100b	156	198	169	168	172	170	159	160	159	243
GEOM110	376	434	397	387	389	383	392	382	376	483
GEOM110a	490	585	502	501	501	494	500	492	490	662
GEOM110b	206	240	215	209	210	206	208	207	206	297
GEOM120	396	458	420	412	409	402	417	405	397	515
GEOM120a	549	649	561	559	564	556	565	559	549	702
GEOM120b	191	225	196	196	201	199	196	195	191	268

The seventh column of Table 3 ("*Phan-Skiena*") reports the solution values obtained by Phan and Skiena [33] by means of the Discropt general heuristic in the case of BCP (they report no results for BMCP). The algorithm works in optimization version.

The solution values ("*Lim05*") by Lim [25], with an algorithm combining squeaky wheel optimization with tabu search, are reported in the seventh column of Table 4 in the case of BMCP (results for BCP are not competitive and were not reported in detail in [25]).

The eighth column ("*Prest 05*") in tables represents the solutions obtained by Prestwich [36] with an algorithm which hybridizes local search and constraint programming.

The solution values ("*Malaguti 08*") by Malaguti and Toth [29], with an algorithm combining an effective tabu search algorithm with population management procedures are reported in the ninth column of tables.

The tenth column ("*Chard*") in Table 4 contains solution obtained by Chardiani and Stützle [8]. For instances GEOM20, GEOM20a, GEOM30, GEOM40 and GEOM50 in [8], the solutions were not given, and therefore, in Table 4 in the places provided for them is a sign "-".

The last column in tables ("*Greedy*") contains solution obtained by greedy algorithm.

Although the genetic approaches presented in this paper did not improve the previous best known solution values, GA2 results are highly comparable with other methods, as can be seen from Tables 3 and 4.

7. CONCLUSIONS

In this paper, evolutionary heuristic algorithms for two generalizations of the well known vertex coloring problem (VCP), namely the bandwidth coloring (BCP) and the bandwidth multicoloring (BMCP) problems, are presented.

GAs use integer encoding, one – point crossover and the fine-grained tournament selection (FGTS). The idea of frozen genes is used in both algorithms to increase the diversity of genetic material. The initial population is generated to be feasible. Genetic operators preserve the feasibility of solutions, so incorrect individuals do not appear throughout all generations. The caching technique additionally improves the computational performance of GA1.

Proposed GA1 has achieved much better solution than the calculated upper bound for a given problem, and GA2 has significantly improved the solutions obtained using GA1. Computational experiments on GEOM instances demonstrate the robustness of the proposed algorithms with respect to the solution quality and running time. Comparisons with the results from the literature show the appropriateness of applying the proposed algorithm components.

This work can be extended in several ways. Based on the results, it seems that the proposed GAs have potential to be useful metaheuristics for solving other similar problems. The second extension can be a parallelization of GA2 and testing on more powerful multiprocessor computer systems.

REFERENCES

[1] Anderson, L.G., "A simulation study of some dynamic channel assignment algorithms in a high capacity mobile telecommunications system", *IEEE Transactions on Communications*, 21 (1973) 1294–1301.

[2] Avanthay, C., Hertz, A., and Zufferey, N., "A variable neighborhood search for graph coloring", *European Journal of Operational Research*, 151 (2) (2003) 379–388.

[3] Blöchliger, I., and Zufferey, N., "A graph coloring heuristic using partial solutions and a reactive tabu scheme", *Computers and Operations Research*, 35 (3) (2008) 960–975.

[4] Bouchard, M., Čangalović, M., and Hertz, A., "On a reduction of the interval coloring problem to a series of bandwidth coloring problems", *Journal of Scheduling*, 13 (6) (2010) 583-595.

[5] Brelaz, D., „New methods to color the vertices of a graph", *Communications Of The ACM*, 22 (4) (1979) 251–256.

[6] Chams, M., Hertz, A., and De Werra, D., “Some experiments with simulated annealing for coloring graphs”, *European Journal of Operational Research*, 32 (2) (1987) 260–266.
[7] Chiarandini, M., and Stützle, T., An application of iterated local search to graph coloring, in: D. S. Johnson (ed.), *Computational Symposium on Graph Coloring and its Generalizations*, 2002, 112–125.
[8] Chiarandini, M., and Stützle, T., "Stochastic local search algorithms for graph set T-colouring and frequency assignment", *Springer Science + Business Media*, 12 (2007) 371–403.
[9] Chiarandini, M., Dumitrescu, I., and Stützle, T., “Stochastic local search algorithms for the graph colouring problem”, in T.F. Gonzalez (ed.), *Handbook of Approximation Algorithms and Metaheuristics*, Chapman & Hall/CRC, Boca Raton,FL,USA, 2007, 1-17.
[10] Christofides, N., “An algorithm for the chromatic number of a graph”, *The Computer Journal*, 14 (1) (1971) 38–39.
[11] Dorne, R., and Hao, JK., "A new genetic local search algorithm for graph coloring", *PPSN 98*, *Lecture Notes In Computer Science*, Berlin: Springer, 1498 (1998) 745-760.
[12] Dorne, R., and Hao, J.K., “Tabu search for graph coloring, t-colorings and set t-colorings” in: S. Voss (ed.), *Meta-Heuristics Advances and Trends in Local Search Paradigms for Optimization*, Kluwer, 1998, 77-92.
[13] Eraghi, A.E., Torkestani, J.A., and Meybodi, M.R., „Solving the bandwidth multicoloring problem: a cellular learning automata approach“, *Proceedings Of the 2009 International Conference On Machine Learning And Computing*, Perth,Australia, 2009.
[14] Fijuljanin, J., "Genetic algorithm for the bandwidth coloring problem and its application in teacing", Master thesis (in Serbian), Faculty of Matematics, University of Belgrade, 2010.
[15] Fleurent, C., and Ferland, J.A., “Object-oriented implementation of heuristic search methods for graph coloring, maximum clique, and satisfiability”, *Cliques, Coloring, and Satisfiability: 2nd DIMACS Implementation Challange*, 1993, *DIMACS Series in Discrete Mathematics and Theoretical Computer Science. American Mathematical Society*, (26) (1996) 619–652.
[16] Galinier, P., and Hao, J.K., “Hybrid evolutionary algorithms for graph coloring”, *Journal of Combinatorial Optimization*, 3 (4) (1999) 379–397.
[17] Hertz A., and De Werra D., “Using tabu search techniques for graph coloring”, *Computing*, 39 (4) (1987) 345–351.
[18] Hertz, A., Plumettaz, A., and Zufferey, N., “Variable space search for graph coloring”, *Discrete Applied Mathematics*, 156 (13) (2008) 2551–2560.
[19] Holland, J., *Adaptation in Natural and Artificial Systems*, The University of Michigan Press (1975).
[20] Johnson, D.S., Aragon, C.R., McGeoch, L.A., and Schevon, C., “Optimization by simulated annealing an experimental evaluation; part II, graph coloring and number partitioning” *Operations Research*, 39 (3) (1991) 378–406.
[21] Kratica, J., Čangalović, M., and Kovačević-Vujčić, V., "Computing minimal doubly resolving sets of graphs", *Computers & Operations Research*, 36 (7) (2009) 2149-2159.
[22] Kratica, J., Kovačević-Vujčić, V., and Čangalović, M., "Computing strong metric dimension of some special classes of graphs by genetic algorithms", *Yugoslav Journal of Operations Research*, 18 (2) (2008) 143-151.
[23] Kratica, J., Kovačević-Vujčić, V., and Čangalović, M., "Computing the metric dimension of graphs by genetic algorithms", *Computational Optimization and Applications*, 44 (2) (2009) 343-361.
[24] Kratica, J., Milanović, M., Stanimirović, Z., and Tošić, D., "An evolutionary based approach for solving a capacitated hub location problem", *Applied Soft Computing*, DOI: 10.1016/j.asoc.2010.05.035.
[25] Lim, A., Lou, Q., Rodrigue,s B., and Zhu, Y., “Heuristic methods for graph coloring problems”, in: *Proceedings of the 2005 ACM Symposium on Applied Computing,* Santa Fe, NM, 2005, 933–939.

[26] Lim, A., Zhang, X., and Zhu, Y., “A hybrid methods for the graph coloring and its related problems”, in: *Proceedings of MIC2003: The Fifth Metaheuristic International Conference*, Kyoto, Japan, 2003.
[27] Lü, Z., and Hao, J.K., "A memetic algorithm for graph coloring", *European Journal of Operational Research*, 203 (1) (2010) 241–250.
[28] Malaguti, E., “The vertex coloring problem and its generalizations“, Dottorato di Ricerca in Automatica e Ricerca Operativa, Universita degli studi di Bologna, Bologna, 2009.
[29] Malaguti, E., and Toth, P., "An evolutionary approach for bandwidth multicoloring problems", *European Journal of Operational Research*, 189 (2008) 638–651.
[30] Malaguti, E., Monaci, M., and Toth, P., "A metaheuristic approach for the vertex coloring problem", *INFORMS Journal on Computing*, 20 (2) (2008) 302-320.
[31] Marti, R., Gortazar, F., and Duarte, A., “Heuristics for the bandwidth coloring problem”, 2009.
[32] Matić, D., "A genetic algorithm for composing music", *Yugoslav Journal of Operations Research*, 20 (1) (2010) 157-177.
[33] Phan, V., and Skiena, S., “Coloring graphs with a general heuristic search engine”, in: [38], 2002, 92–99.
[34] Porumbel, C.D., Hao, J.K., and Kuntz, P., “A search space “cartography” for guiding graph coloring heuristics”, *Computers and Operations Research*, 37 (4) (2010) 769–778.
[35] Prestwich, S., “Constrained bandwidth multicoloration neighborhoods”, in: [38], 2002, 126–133.
[36] Prestwich, S., “Generalized graph colouring by a hybrid of local search and constraint programming”, Technical Report, Cork Constraint Computation Center, Ireland, 2005.
[37] Resende, M.G.C., and Ribeiro, C.C., “Greedy randomized adaptive search procedures”, *Handbook of Metaheuristic,* Kluwer Academic Publishers, 2003, 219–249.
[38] Trick, M.A., *Computational Symposium: Graph Coloring and its Generalization*, Cornell University, Ithaca,NY, 2002. <http://mat.gsia.cmu.edu/COLOR02/>.
[39] Welsh, D.J.A., and Powell, M.B., “An upper bound for the chromatic number of a graph and its application to timetabling problems”, *The Computer Journal*, 10 (1) (1967) 85–86.