Computing and Informatics, Vol. 29, 2010, 1251–1265

# SOLVING THE GENERALIZED VERTEX COVER PROBLEM BY GENETIC ALGORITHM

Marija Milanović

*Faculty of Mathematics*
*University of Belgrade*
*Studentski trg 16/IV, 11 000 Belgrade, Serbia*
*e-mail:* marija.milanovic@gmail.com

Manuscript received 20 October 2008; revised 4 March 2009
Communicated by Jiří Pospíchal

**Abstract.** In this paper an evolutionary approach to solving the generalized vertex cover problem (GVCP) is presented. Binary representation and standard genetic operators are used along with the appropriate objective function. The experiments were carried out on randomly generated instances with up to 500 vertices and 100 000 edges. Performance of the genetic algorithm (GA) is compared with CPLEX solver and 2-approximation algorithm based on LP relaxation. The genetic algorithm outperformed both CPLEX solver and 2-approximation heuristic.

**Keywords:** Vertex cover, genetic algorithms, evolutionary approach, combinatorial optimization, graph algorithms

## 1 INTRODUCTION

In 1972 Karp [10] showed that 21 diverse problems from graph theory and combinatorics are NP-complete. The vertex cover problem (VCP) was one of them. VCP is defined over an undirected graph $G = (V, E)$ and searches for a set of vertices $S$ such that for each edge $e \in E$ at least one of its endpoints belongs to $S$ and $|S|$ is as small as possible.

Up to now, numerous researchers have studied this problem, mostly from the aspect of approximation. Nevertheless, there is significantly smaller number of researchers who have given experimental results. Some of their recently published, successful methods for solving VCP are given in [7, 12, 30, 31].

There are several papers dealing with various generalizations of the vertex cover problem [2, 8, 9, 29]. In this paper a formulation from [9] was chosen.

Let $G = (V, E)$ be an undirected graph, with three numbers $d_0(e) \geq d_1(e) \geq d_2(e) \geq 0$ for each edge $e \in E$. The solution is a subset $S \subseteq V$ and $d_i(e)$ represents the cost contributed to the solution by the edge $e$ if exactly $i$ of its endpoints are in the solution. The cost of including a vertex $v$ in the solution is $c(v)$. The solution has a cost that is equal to the sum of the vertex costs and the edge costs. The generalized vertex cover problem is to compute a minimum cost set of vertices.

The vertex cover problem has many real-world applications. Examples of the areas where this problem occurs are communications, civil and electrical engineering, and bioinformatics (the vertex cover problem finds applications in the construction of phylogenetic trees, in phenotype identification, and in analysis of microarray data). One of the problems that were a motivation for the generalized vertex cover problem is presented in [23]: given a budget that can be used to upgrade vertices, the goal is to upgrade a vertex set such that in the resulting network the minimum cost spanning tree is minimized.

In [9] Hassin and Levin have studied the complexity of GVCP with the costs $d_0(e) = 1$, $d_1(e) = \alpha$, $d_2(e) = 0$ for every $e \in E$ and $c(v) = \beta$ for every $v \in V$ for all possible values of $\alpha$ and $\beta$. They have also provided 2-approximation algorithms for the general case.

In the special case when $d_0(e) = 1$, $d_1(e) = d_2(e) = 0$ for every $e \in E$ and $c(v) = 1$ for every $v \in V$, GVCP is reduced to VCP. Thus, the generalized vertex cover problem is NP-hard as a generalization of the vertex cover problem which is proved to be NP-hard problem. Hassin and Levin have also proved that there are some cases when GVCP can be solved in polynomial time ([9]).

In the literature, there were no papers offering experimental results for GVCP.

## 2 MATHEMATICAL FORMULATION

Let $G = (V, E)$ be an undirected graph. For every edge $e \in E$ three numbers $d_0(e) \geq d_1(e) \geq d_2(e) \geq 0$ are given and for every vertex $v \in V$ a number $c(v) \geq 0$ is given.

For a subset $S \subseteq V$ denote $\overline{S} = V \backslash S$, $E(S)$ is the set of edges whose both end-vertices are in $S$, $E(S, \overline{S})$ is the set of edges that connect a vertex from $S$ with a vertex from $\overline{S}$, $c(S) = \sum_{v \in S} c(v)$, and for $i = 0, 1, 2$ $d_i(S) = \sum_{e \in E(S)} d_i(e)$ and $d_i(S, \overline{S}) = \sum_{e \in E(S, \overline{S})} d_i(e)$.

The generalized vertex cover problem is to find a vertex set $S \subseteq V$ that minimizes the cost $c(S) + d_2(S) + d_1(S, \overline{S}) + d_0(\overline{S})$. Thus, the value $d_i(e)$ represents the cost of the edge $e$ if exactly $i$ of its endpoints are included in the solution, and the cost of including a vertex $v$ in the solution is $c(v)$.

An integer programming formulation of the generalized vertex cover problem, introduced in [9], is shown below.

$$\min \sum_{i=1}^{n} c(i)x_i + \sum_{(i,j)\in E} \Big(d_2(i,j)z_{ij} + d_1(i,j)(y_{ij} - z_{ij}) + d_0(i,j)(1 - y_{ij})\Big) \quad (1)$$

subject to:

$$y_{ij} \le x_i + x_j \qquad \text{for every } (i,j) \in E \quad (2)$$

$$z_{ij} \le x_i \qquad \text{for every } i \in V, (i,j) \in E \quad (3)$$

$$z_{ij} \le x_j \qquad \text{for every } j \in V, (i,j) \in E \quad (4)$$

$$x_i, y_{ij}, z_{ij} \in \{0,1\} \quad (5)$$

where $x_i$ is an indicator variable that is equal to 1 if vertex $i$ is included in solution; $y_{ij}$ is an indicator variable that is equal to 1 if at least one of the vertices $i$ and $j$ is included in the solution, $z_{ij}$ is an indicator variable that is equal to 1 if both $i$ and $j$ are included in the solution.

**Example 1.** Let $|V| = 4$ and $|E| = 5$. The costs $c(v)$ of including vertices in the solution are given in Table 1. For every edge its end-points and $d_0$, $d_1$, and $d_2$ costs are given in Table 2.

| v | 1 | 2 | 3 | 4 |
|---|---|---|---|---|
| c(v) | 1 | 2 | 3 | 4 |

Table 1. c(v) costs

| start | end | $d_0$ | $d_1$ | $d_2$ |
|---|---|---|---|---|
| 1 | 2 | 5 | 3 | 2 |
| 1 | 3 | 4 | 4 | 3 |
| 1 | 4 | 5 | 2 | 2 |
| 2 | 3 | 3 | 2 | 1 |
| 3 | 4 | 2 | 2 | 2 |

Table 2. Edges and their $d_0$, $d_1$, $d_2$ costs

The optimal objective value in this example is 15 and the generalized vertex cover consists of only one vertex (vertex 1). The corresponding vertex cost $c(S) = c(1) = 1$ and the edge costs are $d_1(S,\overline{S}) = d_1(1,2) + d_1(1,3) + d_1(1,4) = 3 + 4 + 2 = 9$, $d_0(\overline{S}) = d_0(2,3) + d_0(3,4) = 3 + 2 = 5$ and $d_2(S) = 0$. Optimal solution is obtained by CPLEX solver using integer programming (Equations (1) through (5)).

## 3 PROPOSED GA METHOD

Genetic algorithm (GA) is a heuristic method based on Darwin's theory of evolution and genetic laws. In the first iteration of the algorithm, population usually consists of randomly generated individuals. Each individual represents an encoded solution of a problem and has a value named fitness associated with it, which represents a quality of the individual in the current population. After applying the genetic operators of selection, crossover and mutation to the current population, the next generation is formed. This process is iteratively performed until some finishing criterion is satisfied. In [28], detailed description of GA can be found.

Extensive computational experience on various optimization problems shows that GA often produces high quality solutions in a reasonable time. Some of recent applications are:

- hub location [6, 18, 19, 20, 34, 35];
- facility location [4, 14, 16, 27, 33];
- biconnectivity augmentation of graphs [24, 25, 26];
- metric dimension of graphs [21, 22];
- generalized Euclidean distances [1];
- spanning sets coverage [11];
- binary sequencing [13].
- maximally balanced connected partition of graphs [3];
- index selection [17];
- machine-job assignment [32].

In proposed implementation of GA, the binary encoding of the individuals is used. Each solution is represented by a binary string of length $|V|$. Digit 1 in the genetic code denotes that particular vertex is in corresponding vertex cover $S$, while 0 shows it is not.

**Example 2.** Let genetic code be 1000. This means that $x_1 = 1$, $x_2 = x_3 = x_4 = 0$, implying $S = \{1\}$. According to Example 1, this genetic code represents its optimal solution.

For improving objective value of the best individual in the current population, the local search is used. The local search is performed only when the best individual is changed and not before the 20th generation. As can be seen below, there is no need for local search if there is no change of the best individual. Also, in the first 20 generations, all the individuals have relatively bad solution quality, so using the local search would be a waste of time.

The local search is performed by using add/remove heuristic with first improvement, that usually outperforms best improvement heuristic. In more detail, for each $ch \in \{1, \ldots, |V|\}$, local search tries to complement $S[ch]$ to $1 - S[ch]$. If that brings improvement to the objective value, the change is performed. That process is repeated until there is no improvement in some iteration, and the last objective value is declared as final objective value.

The proposed GA implementation uses $gvcChange1()$ function, whose code is given in Figure 1, for calculating the objective value of the vertex cover $S$ with $S[ch]$ being complemented to $1 - S[ch]$. In the given code, $val$ represents the objective value of $S$ before the change is performed, $vne$ is array of dimension $|V|$ such that $vne[ch]$ is the number of edges having vertex $ch$ as one of its endpoints, while $E$ and $V$ are matrices of dimension $|V| * |V|$ containing ordinal numbers of those edges and their remaining endpoints, respectively.

```
double gvcChange1(int *S, double val, int ch)
{
   int i;
   double res=val;
   if(S[ch]){
      res-=c[ch];
      for(i=0; i<vne[ch]; i++)
         if(S[V(ch,i)])
            res-=d2[E(ch,i)]-d1[E(ch,i)];
         else
            res-=d1[E(ch,i)]-d0[E(ch,i)];
   }
   else{
      res+=c[ch];
      for(i=0; i<vne[ch]; i++)
         if(S[V(ch,i)])
            res+=d2[E(ch,i)]-d1[E(ch,i)];
         else
            res+=d1[E(ch,i)]-d0[E(ch,i)];
   }
   return res;
}
```

Fig. 1. Code for gvcChange1 function

Initial population of $N_{\text{pop}} = 150$ individuals is randomly generated. This approach provides maximal diversity of genetic material. Fitness of an individual is computed by scaling objective values of all individuals from the population into the interval [0,1] so that the best individual has fitness 1 and the worst one has fitness 0. Explicitly, $f_{\text{ind}} = \frac{\text{obj}_{\text{ind}_{\min}} - \text{obj}_{\text{ind}}}{\text{obj}_{\text{ind}_{\min}} - \text{obj}_{\text{ind}_{\max}}}$. $N_{\text{elite}} = 50$ elite individuals are automatically passed to the next generation. The genetic operators are applied to the rest of the population. Objective value of every elite individual is the same as in the previous generation and is calculated only once, providing significantly better run-time performance of the algorithm.

Individuals with the same genetic code are discarded in every generation in order to avoid premature convergence. Their fitness values are set to zero, except for the first occurrence, so that selection operator avoids them to enter the next generation. Individuals with the same objective value, but different genetic codes, may dominate in some cases in the population. Thus, it is useful to limit the number of their appearance to some constant $N_{\mathrm{rv}}$. In this implementation $N_{\mathrm{rv}} = 40$.

The selection operator chooses the non-elitist individuals which will participate in recombination process and give offspring. In this process, individuals with higher fitness value are favored. As a selection method, the fine grained tournament selection (FGTS), described in [4] is used. This operator uses a real (rational) parameter $F_{tour}$, representing preferable average tournament size. The first type of tournaments is held $k_1$ times and its size is $\lfloor F_{tour} \rfloor$, while the second type is performed $k_2$ times with $\lceil F_{tour} \rceil$ individuals participating, so $F_{tour} \approx \frac{k_1 \cdot \lfloor F_{tour} \rfloor + k_2 \cdot \lceil F_{tour} \rceil}{N_{pop} - N_{elite}}$.

Extensive numerical experiments in [4, 5, 6, 33] performed for different optimization problems indicate that FGTS gives the best results for $F_{tour} = 5.4$. The same value is used in this GA implementation. The running time for FGTS operator is $O((N_{pop} - N_{elite}) \cdot F_{tour})$. In practice $F_{tour}$ and $N_{pop} - N_{elite}$ are considered to be constant that gives a constant running time complexity. For detailed information about FGTS see [5].

After pairs of parents are randomly selected from the set of individuals chosen by FGTS, a crossover operator is applied to them producing two offsprings per each pair of parents. The standard one-point crossover operator is used in the proposed GA. This operator is performed by exchanging segments of two parents' genetic codes starting from a randomly chosen crossover point. The crossover operator is realized with probability $p_{cross} = 0.85$. It means that approximately 85 % pairs of individuals exchange their genetic material.

Modified simple mutation operator is used in this GA implementation. It is performed by changing a randomly selected gene in the genetic code of the individual, with a certain mutation rate. During the GA 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 $2^l$ times smaller and the possibility of a premature convergence increases rapidly. The selection and crossover operators can not change the bit value of any frozen gene and the basic mutation rate is often too small to restore lost subregions of the search space. On the other hand, if the basic mutation rate is increased significantly, a genetic algorithm becomes a random search. For that reason, mutation rate is increased on frozen genes only. In this implementation, the mutation rate for frozen genes is increased 2.5 times $(1.0/n)$, compared to non-frozen ones $(0.4/n)$.

The run-time performance of GA is improved by caching technique. The main idea is to avoid computing the same objective value every time when genetic operators produce individuals with the same genetic code. Evaluated objective values are stored in a hash-queue data structure using the least recently used (LRU) caching technique. When the same code is obtained again, its objective value is taken from the cache memory, that provides time-savings. In this implementation the number of individuals stored in the cache memory is limited to 5 000. For detailed information about caching GA see [15].

A research group the author belongs to has a large experience on genetic algorithms. The choice of GA parameters presented in this paper is based on that experience. In [33] GA parameters were intensively tested and values reported as best are chosen here.

## 4 EXPERIMENTAL RESULTS

All computations were executed on Intel 2.5 GHz PC with 1 GB RAM under Windows XP operating system. Genetic algorithm was coded in C programming language.

Since there were no instances for this problem, the author randomly generated instances using the following algorithm:

- input data: $|V|$, $|E|$, random_seed;
- $|E|$ out of all possible $\frac{|V|*(|V|-1)}{2}$ edges are randomly generated;
- for each edge $e \in E$, $d_0(e)$ is a random real number from interval [0,100], $d_2(e)$ is a random real number from interval [0, $d_0(e)$], $d_1(e)$ is a random real number from interval [$d_2(e)$, $\frac{d_0(e)+d_2(e)}{2}$];
- following pseudo code describes generation of $c(v)$ for all $v \in V$:

```
for(i=0; i<|V|; i++){
   avg_gain=0;
   for(j=0;j<100;j++) {
      for(k=0; k<|V|; k++) S[k] = random(0,1);
      avg_gain += calculate_gain_of_d(i);
   }
   avg_gain /= 100;
   c[i] = avg_gain;
}
```

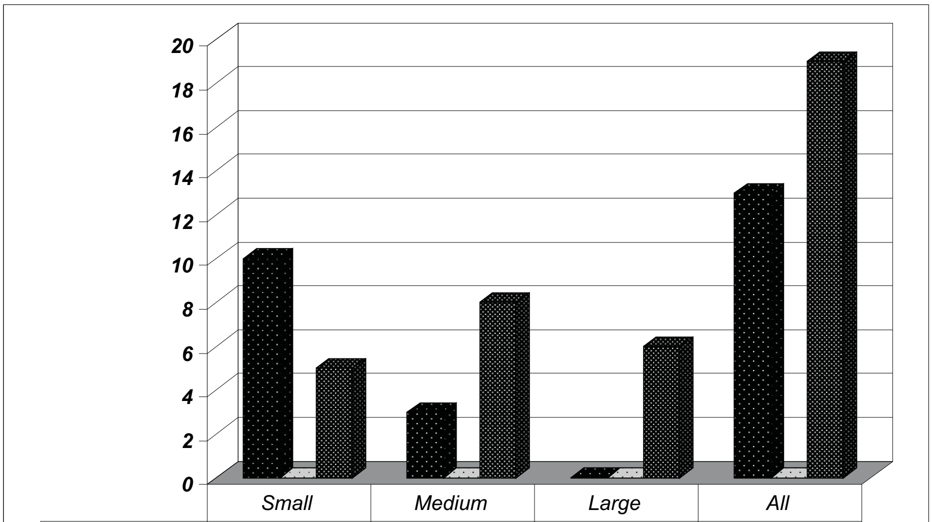
Supposing $S$ is the current vertex cover, procedure *calculate_gain_of_d*() returns the value obtained as the edge costs gain of adding vertex $v$ to the cover $S$. The previous pseudo code calculates the cost $c(v)$ of including vertex $v$ in the solution as the average edge costs gain of including vertex $v$ in 100 random solutions. All of the modern exact and heuristic methods have a preprocessing part which removes all "useless" variables, i.e. the ones which clearly can not participate in any good solution. Taking the previously described average gain as $c(v)$, generated instances have a small number of "useless" vertices.

Moreover, previous procedure for generating GVCP instances effectively prevents occurrence of "easy" instances that can be solvable in polynomial time. Detailed information about cases when GVCP is polynomially solvable can be found in [9]. Thus, the calculation of right endpoints of intervals from which coefficients $d_1$ were chosen during the generation of instances is based on results from [9].

| Instance name | $Opt_{sol}$ | *CPLEX (2-hour)* | | *2-appr* | |
|---|---|---|---|---|---|
| | | Sol | t (sec) | Sol | $t$ (sec) |
| gvc-30-50 | 2 227 | opt | 0.031 | 2 543 | 0.015 |
| gvc-30-100 | 4 163 | opt | 0.125 | 4 685 | 0.031 |
| gvc-30-200 | 9 687 | opt | 1.031 | 10 703 | 0.062 |
| gvc-30-400 | 18 553 | opt | 18.750 | 20 146 | 0.125 |
| gvc-50-100 | 4 326 | opt | 0.109 | 4 930 | 0.046 |
| gvc-50-200 | 8 853 | opt | 0.546 | 9 899 | 0.296 |
| gvc-50-500 | 23 072 | opt | 245.296 | 25 384 | 0.156 |
| gvc-50-1000 | – | 46 729 | 7 200 | 50 518 | 0.640 |
| gvc-100-200 | 8 430 | opt | 0.343 | 9 468 | 0.046 |
| gvc-100-500 | 22 334 | opt | 157.796 | 24 867 | 0.109 |
| gvc-100-1000 | – | 44 922 | 7 200 | 49 283 | 0.343 |
| gvc-100-4000 | – | 185 359 | 7 200 | 199 840 | 6.656 |
| gvc-200-500 | 22 510 | opt | 10.453 | 25 553 | 0.234 |
| gvc-200-2000 | – | 90 486 | 7 200 | 99 198 | 2.515 |
| gvc-200-5000 | – | 231 586 | 7 200 | 249 837 | 6.093 |
| gvc-200-15000 | – | 707 176 | 7 200 | 755 170 | 171.140 |
| gvc-300-1000 | – | 44 319 | 7 200 | 49 826 | 0.421 |
| gvc-300-5000 | – | 231 947 | 7 200 | 253 128 | 8.453 |
| gvc-300-20000 | – | 934 713 | 7 200 | 999 471 | 195.656 |
| gvc-300-40000 | – | 1 865 107 | 7 202 | 1 992 350 | 1760 |
| gvc-400-1200 | – | 53 953 | 3 903.* | 60 483 | 1.218 |
| gvc-400-5000 | – | 227 290 | 7 200 | 247 458 | 6.937 |
| gvc-400-20000 | – | 934 176 | 7 200 | 997 593 | 189.125 |
| gvc-400-70000 | – | 3 280 984 | 7 201 | 3 497 365 | 6 004 |
| gvc-500-1500 | – | 67 628 | 6 110.* | 75 631 | 0.859 |
| gvc-500-5000 | – | 228 311 | 7 200 | 249 300 | 6.828 |
| gvc-500-30000 | – | 1 403 360 | 7 200 | 1 499 411 | 352.234 |
| gvc-500-100000 | – | 4 725 438 | 7 202 | 4 993 478 | 4 420 |

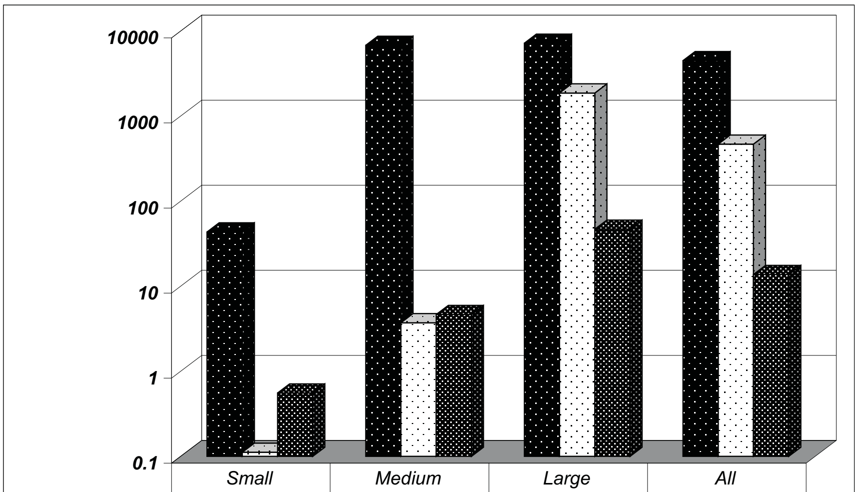
Table 3. CPLEX and 2-appr results

Integer programming formulation (Equations (1) through (5)) is implemented and tested by CPLEX 10.1.0 solver in order to obtain optimal solutions. Time limitation was set to 7 200 seconds per execution preventing very long running time. According to this limitation, in some cases optimal solutions were not reached, so best found solution was reported.




| | Small | Medium | Large | All |
|---|---|---|---|---|
| CPLEX 2-hour | 10 | 3 | 0 | 13 |
| 2-appr | 0 | 0 | 0 | 0 |
| GA | 5 | 8 | 6 | 19 |

Fig. 2. Number of optimal/best solutions of GA, CPLEX 2-hour, and 2-appr methods




| | Small | Medium | Large | All |
|---|---|---|---|---|
| CPLEX 2-hour | 43.448 | 6801 | 7201 | 4488 |
| 2-appr | 0.112 | 3.724 | 1870 | 469.079 |
| GA | 0.556 | 4.657 | 46.876 | 13.747 |

Fig. 3. Average running times of GA, CPLEX 2-hour, and 2-appr methods

| Instance name | $Best_{sol}$ | GA | | | | |
|---|---|---|---|---|---|---|
| | | Sol | t (sec) | Gen | Eval | Cache (%) |
| gvc-30-50 | 2 227 | opt | 0.259 | 2 210 | 19 546 | 82 |
| gvc-30-100 | 4 163 | opt | 0.252 | 2 225 | 15 550 | 86 |
| gvc-30-200 | 9 687 | opt | 0.274 | 2 261 | 19 980 | 82 |
| gvc-30-400 | 18 553 | opt | 0.269 | 2 047 | 14 966 | 85 |
| gvc-50-100 | 4 326 | 4 327 | 0.376 | 2 447 | 29 342 | 76 |
| gvc-50-200 | 8 853 | 8 880 | 0.412 | 2 576 | 32 276 | 75 |
| gvc-50-500 | 23 072 | opt | 0.457 | 2 264 | 28 466 | 75 |
| gvc-50-1000 | 46 665 | best | 0.607 | 2 292 | 30 992 | 73 |
| gvc-100-200 | 8 430 | 8 438 | 0.666 | 2 787 | 53 850 | 61 |
| gvc-100-500 | 22 334 | 22 355 | 0.832 | 2 825 | 56 465 | 60 |
| gvc-100-1000 | 44 815 | best | 1.359 | 3 372 | 70 298 | 58 |
| gvc-100-4000 | 184 327 | best | 3.074 | 3 007 | 64 612 | 58 |
| gvc-200-500 | 22 510 | 22 579 | 1.767 | 4 154 | 106 005 | 49 |
| gvc-200-2000 | 90 294 | best | 2.668 | 3 260 | 83 994 | 49 |
| gvc-200-5000 | 229 782 | best | 6.728 | 4 171 | 111 989 | 46 |
| gvc-200-15000 | 701 164 | best | 16.471 | 3 931 | 107 367 | 46 |
| gvc-300-1000 | 44 319 | 44 414 | 3.284 | 4 796 | 135 735 | 43 |
| gvc-300-5000 | 230 810 | best | 8.028 | 4 419 | 127 094 | 42 |
| gvc-300-20000 | 925 711 | best | 20.974 | 3 460 | 101 868 | 41 |
| gvc-300-40000 | 1 853 084 | best | 37.974 | 3 209 | 95 163 | 41 |
| gvc-400-1200 | 53 953 | 54 033 | 3.816 | 4 434 | 131 280 | 41 |
| gvc-400-5000 | 226 097 | best | 8.103 | 4 081 | 122 797 | 40 |
| gvc-400-20000 | 922 492 | best | 19.615 | 3 083 | 93 384 | 39 |
| gvc-400-70000 | 3 257 512 | best | 80.215 | 3 474 | 106 971 | 39 |
| gvc-500-1500 | 67 628 | 67 758 | 4.870 | 4 482 | 137 535 | 39 |
| gvc-500-5000 | 227 207 | best | 8.692 | 4 070 | 125 570 | 38 |
| gvc-500-30000 | 1 387 633 | best | 36.475 | 3 744 | 117 362 | 37 |
| gvc-500-100000 | 4 651 995 | best | 116.410 | 3 236 | 103 695 | 36 |

Table 4. GA results

Since there were no papers containing experimental results for GVCP, for comparing with performances of the GA, 2-approximation algorithm from [9] was implemented and tested. 2-approximation algorithm is based on LP relaxation of (1)–(5) integer program, fixing all relaxed binary variables with values greater than or equal to $\frac{1}{2}$ to 1. Other variables (with relaxed value less than $\frac{1}{2}$) were fixed to 0. For solving this LP relaxation, CPLEX 10.1.0 solver is also used.

Table 3 summarizes CPLEX and 2-approximation algorithm results on generated instances. In the first column the test instance name is given. The instance's name carries information about the number of vertices and the number of edges, respectively. For example, the instance *gvc*_100_1000 is created by using the above algorithm given 100 as a number of vertices, and 1 000 as a number of edges.

The second column contains optimal solution on the current instance, if it is known (obtained by CPLEX solver), otherwise sign − is written. The third and fourth columns contain solution and running time of CPLEX solver while solution and running time of 2-approximation heuristic are presented in the fifth and sixth columns of the table. Mark *opt* is given if optimal solution is reached. Sign $*$ in the column $t$ is written for the cases when CPLEX solver run out of memory.

The finishing criterion of GA is the maximal number of generations $N_{gen} = 5\,000$. The algorithm also stops if the best individual or best objective value remains unchanged through $N_{rep} = 2\,000$ successive generations. Since the results of GA are nondeterministic, the GA was run 20 times on each problem instance.

In Table 4 results of GA are presented. The first column also contains the test instance name. The second column contains the best known solution on the current instance, i.e. the best solution among the solutions given by CPLEX (2-hour), 2-approximation, and GA. The best GA value $GA_{best}$ is given in the following column, with mark *opt* in cases when GA reached optimal solution (obtained by CPLEX solver). If the GA reached the best-known solution, which is not proved to be optimal, the mark *best* is written. The next column $t$ contains the average running time (in seconds) used to reach the final GA solution. The average number of generations for finishing GA is presented in column *Gen*. In the last two columns, *Eval* represents the average number of the objective function evaluations, while *Cache* displays savings (in percent) achieved by using cache technique.

In both tables, time values are shown without decimal places if they are greater than 1 000, and with three decimal places otherwise. Also, all other values are presented without decimal places.

The results shown in Table 3 and Table 4 are put together and illustrated in a graphical form in Figures 2 and 3. Instances are divided into three groups: small (up to 500 edges), medium (between 1 000 and 5 000 edges) and large (more than 5 000 edges). In Figure 2 for every of three compared methods (CPLEX, 2-approximation, and GA) and every group of instances, the number of the group's instances on which the method has reached optimal/best solution is reported. Also, the corresponding number for group of all 28 instances is presented. In Figure 3 for every group of instances and every method the average running time of that method on the instances of that group is presented. Also, the corresponding average running time is presented for group of all 28 instances. A logarithmic scale is used in Figure 3.

As can be seen from Table 3, 2-approximation algorithm has not reached either optimal or best solution. In five cases ($gvc-30-50$, $gvc-30-100$, $gvc-30-200$, $gvc-30-400$, $gvc-50-500$) both GA and CPLEX solver have reached optimal solution. On other five instances ($gvc-50-100$, $gvc-50-200$, $gvc-100-200$, $gvc-100-500$, $gvc-200-500$) CPLEX has reached optimal solution, but GA has not. CPLEX has not reached optimal solution in 2 hours, but produced better solution than GA in 3 more cases: $gvc-300-1000$, $gvc-400-1200$, $gvc-500-1500$. For remaining 15 instances GA produced better solutions than CPLEX. Note that CPLEX is more suitable for solving GVCP on smaller dimension sparse graphs, while GA has advantages on larger and/or dense graphs.

Except for instances: $gvc-30-50$, $gvc-30-100$, $gvc-50-100$, $gvc-50-200$, $gvc-100-200$ where running time of both CPLEX and GA is less than 1 second, on all other instances GA has much smaller running time. Note that GA running time did not exceed 117 seconds for all instances. On the other hand, CPLEX has different behavior, in cases of smaller dimension sparse graphs where optimal solution was reached in less than 246 seconds, in other cases, even after 2 hours of execution quality of solution was rather bad.

## 5 CONCLUSIONS

In this paper an efficient evolutionary metaheuristic for solving the generalized vertex cover problem is presented. The binary representation, the mutation with frozen genes, limited number of different individuals with the same objective value and the caching technique were used. Solution quality is improved by using the local search heuristic that is efficiently implemented in GA.

As can be seen from experimental results, this approach seems to be a good candidate for solving GVCP. Computational experiments demonstrate the robustness of the proposed algorithm with respect to the solution quality and running times. Comparisons with the results of the CPLEX and 2-approximation heuristic show the appropriateness of applying the proposed algorithm components.

Future research will be directed to parallelization of the presented GA, incorporation in exact methods and application for solving similar problems.

### Acknowledgement

This research was partially supported by the Serbian Ministry of Science and Ecology under project 144007. The author is grateful to Jozef Kratica, Jelena Kojić and Ivana Ljubić for their useful suggestions and comments.

## REFERENCES

[1] Aler, R. M.—Valls, J.—Fernandez, O.: Evolving Generalized Euclidean Distances for Training RBNN. Computing and Informatics, Vol. 26, 2007, No. 1, pp. 33–43.

[2] Broersma, H. J.—Paulusma, D.—Smit, G. J. M.—Vlaardingerbroek, F.—Woeginger, G. J.: The Computational Complexity of the Minimum Weight Processor Assignment Problem. Lecture Notes in Computer Science, Vol. 3353, 2004, pp. 189–200.

[3] Djurić, B.—Kratica, J.—Tošić, D.—Filipović, V.: Solving the Maximally Balanced Connected Partition Problem in Graphs by Using Genetic Algorithm. Computing and Informatics, Vol. 27, 2008, No. 3, pp. 341–354.

[4] Filipović, V.—Kratica, J.—Tošić, D.—Ljubić, I.: Fine Grained Tournament Selection for the Simple Plant Location Problem. Proceedings of the 5th Online World

Conference on Soft Computing Methods in Industrial Applications – WSC5, September 2000, pp. 152–158.

[5] Filipović, V.: Fine-Grained Tournament Selection Operator in Genetic Algorithms. Computing and Informatics, Vol. 22, 2003, pp. 143–161.

[6] Filipović, V.: Operatori Selekcije i Migracije i Web Servisi Kod Paralelnih Evolutivnih Algoritama. Ph. D. thesis, University of Belgrade, Faculty of Mathematics, 2006.

[7] Gilmour, S.—Dras, M.: Kernelization as Heuristic Structure for the Vertex Cover Problem. Lecture Notes in Computer Science, Vol. 4150, 2006, pp. 452–459.

[8] Guo, J.—Niedermeier, R.—Wernicke, S.: Parameterized Complexity of Generalized Vertex Cover Problems. Lecture Notes In Computer Science, Vol. 3608, 2005, pp. 36–48.

[9] Hassin, R.—Levin, A.: The Minimum Generalized Vertex Cover Problem. ACM Transactions on Algorithms, Vol. 2, 2006, pp. 66–78.

[10] Karp, R. M.: Reducibility Among Combinatorial Problems. Complexity of Computer Computations, Plenum Press, 1972, pp. 85–103.

[11] Khamis, A. M.—Girgis, M. R.—Ghiduk, A. S.: Automatic Software Test Data Generation for Spanning Sets Coverage Using Genetic Algorithms. Computing and Informatics, Vol. 26, 2007, No. 4, pp. 383–401.

[12] Kotecha, K.—Gambhava, N.: A Hybrid Genetic Algorithm for Minimum Vertex Cover Problem. Proceedings of the First Indian International Conference on Artificial Intelligence, 2003, pp. 904–913.

[13] Kovačević, J.: Hybrid Genetic Algorithm for Solving the Low-Autocorrelation Binary Sequence Problem. Yugoslav Journal of Operations Research (to appear).

[14] Kratica, J.: Improvement of Simple Genetic Algorithm for Solving the Uncapacitated Warehouse Location Problem. Advances in Soft Computing – Engineering Design and Manufacturing, R. Roy, T. Furuhashi and P. K. Chawdhry (Eds.), Springer-Verlag London Limited, 1998, pp. 390–402.

[15] Kratica, J.: Improving Performances of the Genetic Algorithm by Caching. Computers and Artificial Intelligence, Vol. 18, 1999, pp. 271–283.

[16] Kratica, J.—Tošić, D.—Filipović, V.—Ljubić, I.: Solving the Simple Plant Location Problem by Genetic Algorithms. RAIRO – Operations Research, Vol. 35, 2001, pp. 127–142.

[17] Kratica, J.—Ljubić, I.—Tošić, D.: A Genetic Algorithm for the Index Selection Problem. Lecture Notes in Computer Science, Vol. 2611, 2003, pp. 281–291.

[18] Kratica, J.—Stanimirović, Z.—Tošić, D., Filipović, V.: Genetic Algorithm for Solving Uncapacitated Multiple Allocation Hub Location Problem. Computing and Informatics, Vol. 24, 2005, pp. 415–426.

[19] Kratica J.—Stanimirović, Z.: Solving the Uncapacitated Multiple Allocation P-Hub Center Problem by Genetic Algorithm. Asia-Pacific Journal of Operational Research, Vol. 24, 2006, No. 4, pp. 425–437.

[20] Kratica, J.—Stanimirović, Z.—Tošić, D.—Filipović, V.: Two Genetic Algorithms for Solving the Uncapacitated Single Allocation P-Hub Median Problem. European Journal of Operational Research, Vol. 182, 2007, No. 1, pp. 15–28.

[21] Kratica, J.—Kovačević-Vujčić, V.—Čangalović, M.: Computing the Metric Dimension of Graphs by Genetic Algorithms. Computational Optimization and Applications, DOI 10.1007/s10589-007-9154-5 (to appear).

[22] Kratica, J.—Kovačević-Vujčić, V.—Čangalović, M.: Computing Strong Metric Dimension of Some Special Classes of Graphs by Genetic Algorithms. Yugoslav Journal of Operations Research, Vol. 18, 2008, No. 2, pp. 143–151.

[23] Krumke, S. O.—Marathe, M. V.—Noltemeier, H.—Ravi, R.— Ravi, S. S.—Sundaram, R.—Wirth, H. C.: Improving Minimum Cost Spanning Trees by Upgrading Nodes. Journal of Algorithms, Vol. 33, 1999, pp. 92–111.

[24] Ljubić, I.—Raidl, G. R.—Kratica, J.: A Hybrid GA for the Edge-Biconnectivity Augmentation Problem. Lecture Notes in Computer Science, Vol. 1917, 2000, pp. 641–650.

[25] Ljubić, I.—Raidl, G. R.: A Memetic Algorithm for Minimum-Cost Vertex-Biconnectivity Augmentation of Graphs. Journal of Heuristics, Vol. 9, 2003, pp. 401–427.

[26] Ljubić, I.: Exact and Memetic Algorithms for Two Network Design Problems. Ph. D. thesis, Institute of Computer Graphics, Vienna University of Technology, 2004.

[27] Marić, M.: An Efficient Genetic Algorithm for Solving the Multi-Level Uncapacitated Facility Location Problem. Computing and Informatics, Vol. 29, 2010, No. 2, pp. 183–201.

[28] Mitchell, M.: Introduction to Genetic Algorithms. MIT Press, Cambridge, Massachusetts, 1999.

[29] Moser, H.: Exact Algorithms for Generalizations of Vertex Cover. M. Sc. thesis, Friedrich-Schiller-University Jena, Faculty of Mathematics and Informatics, 2005.

[30] Pelikan, M.—Kalapala, R.—Hartmann, A. K.: Hybrid Evolutionary Algorithms on Minimum Vertex Cover for Random Graphs. Proceedings of the Genetic and Evolutionary Computation Conference – GECCO 2007, pp. 547–554.

[31] Richter, S.—Helmert, M.—Gretton, C.: A Stochastic Local Search Approach to Vertex Cover. Lecture Notes in Computer Science, Vol. 4667, 2007, pp. 412–426.

[32] Savić, A.: An Genetic Algorithm Approach for Solving the Machine-Job Assignment With Controllable Processing Times. Computing and Informatics (to appear).

[33] Stanimirović, Z.—Kratica, J.—Dugošija, Dj.: Genetic Algorithms for Solving the Discrete Ordered Median Problem. European Journal of Operational Research, Vol. 182, 2007, No. 3, pp. 983–1001.

[34] Stanimirović, Z.: Genetic Algorithms for Solving Some NP-Hard Hub Location Problems. Ph. D. thesis, University of Belgrade, Faculty of Mathematics, 2007.

[35] Stanimirović, Z.: A Genetic Algorithm Approach for the Capacitated Single Allocation P-Hub Median Problem. Computing and Informatics, Vol. 29, 2010, No. 1, pp. 117–132.

**Marija Milanović** received her B. Sc. degree in mathematics (2007) from University of Belgrade, Faculty of Mathematics. Since 2007 she works as a teaching assistant and since 2008 she is a Ph. D. student at the Faculty of Mathematics. Her research interests include genetic algorithms, combinatorial optimization and algorithms on graphs.