Computing and Informatics, Vol. 27, 2008, 341–354

SOLVING THE MAXIMALLY BALANCED CONNECTED PARTITION PROBLEM IN GRAPHS BY USING GENETIC ALGORITHM

Brankica Djurić

Faculty of Mathematics
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: mickob@sbb.co.yu

Jozef Kratica

Mathematical Institute
Serbian Academy of Sciences and Arts
Kneza Mihajla 35/I, pp. 367
110 01 Belgrade, Serbia
e-mail: jkratica@mi.sanu.ac.yu

Dušan Tošić, Vladimir Filipović

Faculty of Mathematics
University of Belgrade
Studentski trg 16/IV
11 000 Belgrade, Serbia
e-mail: {dtosic, vladaf}@matf.bg.ac.yu

Manuscript received 10 October 2006; revised 31 May 2007
Communicated by Vladimír Kvasnička

Abstract. This paper exposes a research of the NP-hard Maximally Balanced Connected Partition problem (MBCP). The proposed solution comprises of a genetic algorithm (GA) that uses: binary representation, fine-grained tournament selection, one-point crossover, simple mutation with frozen genes and caching technique. In cases of unconnected partitions, penalty functions are successfully applied in order to obtain the feasible individuals. The effectiveness of presented approach is demonstrated on the grid graph instances and on random instances with up to 300 vertices and 2 000 edges.

Keywords: Balanced partitions, evolutionary computation, metaheuristics, combinatorial optimization.

1 INTRODUCTION

A lot of researchers used genetic algorithms for solving NP-hard optimization problems in the field of combinatorial optimization. Practical orientation and the use of graph problems make them very interesting for research. In most cases it is necessary that involved graphs or subgraphs should be connected.

We considered problem (named the Maximally Balanced Connected Partition Problem – MBCP) and proposed a solution based on genetic algorithm. Our goal was to partition a graph into the two connected subgraphs with minimal misbalance, i.e. having sums of weights that are as much equal as possible.

Solving MBCP problems using GAs can be applied on finding solutions to practical organizational problems in education. As an example, educational course planning usually consists of multiple cases when partitions of single course into two may become necessary. The other example would be partition of a study group into two student subgroups that follow complex assignment criteria.

This problem is considered in [5] and the polynomial-time approximation algorithm, with excellent guaranteed bound 1.072, is proposed. However, that paper contains only theoretical results. It also contains detailed proof of the NP-hardness of the MBCP.

2 PROBLEM DEFINITION

Let $G = (V, E)$ be connected graph where V represents a set of vertices and E is set of edges. The solution of previously described problem is a partition (V_1, V_2) of V, resulting in two non-empty disjoint sets ($V_1, V_2 \subset V$, $V_1 \cup V_2 = V$ and $V_1 \cap V_2 = \emptyset$), which induces subgraphs $G_1 = (V_1, E_1)$, $E_1 \subset E$ and $G_2 = (V_2, E_2)$, $E_2 \subset E$, of G. Those two subgraphs are connected.

We proposed to use $B(V_1, V_2) = |w(V_1) - w(V_2)|$ as objective function, where $w(V_1) = \sum\limits_{i \in V_1} w(i)$ and $w(V_2) = \sum\limits_{i \in V_2} w(i)$ and our goal was to minimize that function. The following example clearly illustrates the definition of our problem.

Example 1. For a graph given in Figure 1, the sum of all weights is $3 + 7 + 1 + 4 + 9 = 24$. The minimum of $|w(V_1) - w(V_2)|$ can be 0 only in case when

$V_1 = \{A, F\}$, $V_2 = \{B, C, D\}$ because $w(V_1) = w(A) + w(F) = 3 + 9 = 12$ and $w(V_2) = w(B)+w(C)+w(D) = 7+1+4 = 12$, are equal. However, both subgraphs G_1, G_2 are unconnected ($(A, F) \notin E$ and $(B, C) \notin E$). Therefore, the optimal solution can not be 0, but at least 2, because the sum of all weights is even. Let $V_1 = \{A, B, C\}$, $V_2 = \{D, F\}$, with $|w(V_1)| = 11$, $|w(V_2)| = 13$. In this case, G_1, G_2 are connected subgraphs ($(A, B) \in E$, $(A, C) \in E$ and $(D, F) \in E$), so $B(V_1, V_2) = |w(V_1) - w(V_2)| = 2$ is an optimal value.

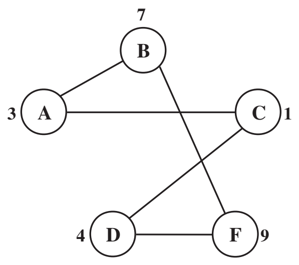

Fig. 1. Graph used in Example 1

3 GA IMPLEMENTATION FOR SOLVING THE MBCP

The main idea of genetic algorithms (GAs) was introduced by Holland [8], and in the last three decades GAs have emerged as effective, robust optimization and search methods. The GAs operate on the population of individuals, each representing a possible solution to a given problem. The encoding of individual entities along with their fitness function definition are important aspects of each GA efficiency. In most cases, successful application of a GA requires a binary encoding (where possible), which assigns a code of predefined length consisting of elements $\{0, 1\}$. Some examples of successful applications based on binary encoding can be found in [6, 12, 13, 20, 21].

Each individual has an assigned fitness value according to the quality of the corresponding solution, comparing the value of the individual to those of its peers in the population. Actual comparison becomes possible only after individual objective values have been calculated, as only all of them become available. The individual fitness values represent their relative impacts on the whole population. The population evolves towards better solutions by means of randomized processes known as selection, crossover, and mutation. Through the selection mechanism, the individuals with better fitness values are favored to reproduce more often than the worse ones. The crossover allows combining of parental information when it is passed to their descendants. The result of crossover is a structured, randomized exchange of genetic material between individuals, with the possibility that good solutions can generate better ones. The mutation involves modification of a individual gene value with some small probability p_{mut}. Typical role of mutation in GAs is to restore lost or unexplored genetic material in the population. The mutation can be used to prevent premature convergence of GA to suboptimal solutions. The initial population is usually randomly initialized. For general information about GAs see [1, 2, 7, 12, 16, 20].

In the past, GAs have been often successfully used to solve NP-hard combinatorial optimization problems. Some of these effective applications are presented in [2, 9, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22].

The sketch of GA proposed to solve maximally balanced connected partition problem is given in Figure 2.

```
Input_Data();
Population_Init();
while not Finish() do
   for ind:= ($N_{elite}$ + 1) to $N_{pop}$ do
      if (Exists_in_Cache(ind)) then
         $ov_{ind}$:= Get_Value_From_Cache(ind);
      else
         if (Correct_Individual(ind)) then
            $ov_{ind}$:= Objective_Function(ind);
         else
            $ov_{ind}$:= Objective_Function(ind) + Penalty_Function(ind);
            Put_Into_the_Cache_Memory(ind, $ov_{ind}$);
            if (Full_Cache_Memory)) then
               Remove_LRU_Block_From_Cache_Memory();
            endif
         endif
      endif
   endfor
   Fitness_Function();
   Selection();
   Crossover();
   Mutation();
endwhile
Output_Data();
```

Fig. 2. The basic scheme of this GA implementation

In the described algorithm N_{pop} denotes the overall number of individuals in a population, N_{elite} represents the number of elite individuals (described in detail in Section 3.3) and *ind* and ov_{ind} are the individual and its objective value. At the beginning, the call to Input_Data() function reads problem instance as well as GA parameters. After that, Population_Init() function is used to randomly generate the initial population. Within the **for** loop, a check of each population member genetic code (*ind*) and value (ov_{ind}) is performed in the following manner: 1) a call to the Boolean function Exists_in_Cache(*ind*) decides whether the currently processed individual had been already stored in cache memory, 2) if it is cached, the individual information is retrieved from the cache, 3) otherwise, the objective value of correct individual *ind* is calculated using the Objective_Function(*ind*), while the objective value of incorrect individual *ind* is the sum of Objective_Function(*ind*) and Penalty_Function(*ind*). Function Put_Into_the_Cache_Memory(ind,ov_{ind}) is used to cache current individual information. If cache memory consumption becomes too large, a call to Remove_LRU_Block_From_Cache_Memory() function deallocates least recently used (LRU) block from cache memory. In that way, current individual can be stored into the previously deallocated block. In each generation, the fitness values for all individuals are calculated in the above mentioned way, and three GA operators are subsequently applied to them by invoking Selection(), Crossover(), and Mutation(). In the continuation, we presented detailed description of all these GA aspects.

3.1 Representation

One of the key factors affecting the success and efficiency of a genetic algorithm is the appropriate representation of the problem space. Therefore, it is difficult to find a relatively short GA encoding dealing with specific constraints that usually direct search to unfeasible solutions.

A candidate solution is represented by the vector x of $|V|$ binary genes $x_i \in \{0,1\}$, $i = 1, \ldots, |V|$. Each gene x_i is associated to a vertex from V and indicates whether the vertex i is included in the subset V_1 ($x_i = 1$) or in the subset V_2 ($x_i = 0$). After that, the values of $w(V_1)$ and $w(V_2)$ are computed by summing of all weights in the corresponding subsets, and the objective value is $B(V_1, V_2) = |w(V_1) - w(V_2)|$.

For each new chromosome, the previous procedure gives one partition of graph G. This GA representation may generate unconnected partitions. Such solution is incorrect and GA has to be oriented towards correct solution space. In order to perform this task, the unconnected partitions are penalized. Properly regulated penalty will increase probability for appearance of correct individuals in the population, through the next generations. For every subgraph, a check of the number of connected components is executed. This task is performed on both subgraphs of the current partition via the well known breadth-first search algorithm. If both subgraphs have only one connected component, it means that subgraphs are connected. Such individual is obviously correct and the penalty function will not be applied.

Nevertheless, dealing with unconnected subgraphs (the number of connected components is more than one), it is convenient to explicitly incorporate that number into the penalty function. An explicit formula for the penalty function is given in (1).

$$Penalty\_Function(ind) = (ncc_2(ind) - 1) * max_1 + (ncc_1(ind) - 1) * max_2 \quad (1)$$

The value calculated would be added to the objective value in case of incorrect individual *ind*. Here $ncc_1(ind)$ and $nnc_2(ind)$ denote the numbers of connected components in the first and second subgraphs. The formula for maximal weights of vertices in the first subgraph is $max_1 = \max\{w(j)|j \in V_1\}$, and for the second subgraph it is $max_2 = \max\{w(j)|j \in V_2\}$.

The overall time complexity of the objective function is $O(|V| + |E|)$. Its estimation is based on the following facts: 1) time complexity for obtaining vector x from genetic code is equal to $O(|V|)$; 2) retrieval of connected components for subgraphs $G_1 = (V_1, E_1)$ and $G_2 = (V_2, E_2)$ is based of the Breadth First Search (BFS) algorithm whose time complexity is equal to $O(|V_1| + |E_1|)$ (for $G_1 = (V_1, E_1)$) and $O(|V_2| + |E_2|)$ (for $G_2 = (V_2, E_2)$), or implies $O(|V| + |E|)$ for searching both subgraphs; 3) time complexity for summing of all weights in the corresponding subsets is $O(|V|)$; 4) time complexity for penalty function is $O(1)$, because max1, max2, ncc1 and ncc2 are computed in 2).

3.2 Genetic Operators

GA selection operator performs selection of population individuals that are eligible for creation of the new generation. Selection operation is based on values returned by the fitness function. So far, various selection operators have been reported (see for example [1, 2, 12]): roulette wheel selection, ranking selection, tournament selection, uniform selection, etc. In this paper, an improvement of the classical tournament selection, fine-grained tournament selection – FGTS (as proposed in [6]) is adopted as a reproduction operator. Such selection scheme proved to be successful in the cases when it is desirable that the size of tournament group has rational instead of integer values. This operator uses real (rational) parameter F_{tour} denoting desired average tournament size. The first type of tournaments is held k_1 times and its size is $\lfloor F_{tour} \rfloor$. As the second type is held k_2 times with $\lceil F_{tour} \rceil$ individuals participated, the value of F_{tour} is calculated as $F_{tour} \approx \frac{k_1 \cdot \lfloor F_{tour} \rfloor + k_2 \cdot \lceil F_{tour} \rceil}{N_{nnel}}$. In this implementation of FGTS, the value of $F_{tour} = 5.4$ is used, which corresponds to values $k_1 = 30$ and $k_2 = 20$ for $N_{nnel} = N_{pop} - N_{elite} = 50$ non-elitist individuals. The particular value of 5.4 is chosen because it gave very acceptable results in solving similar problems (see [6, 7, 13, 14]). The time complexity of FGTS operator is $O(N_{nnel} \cdot F_{tour})$. In practice, F_{tour} and N_{nnel} are considered constant (not depending on problem dimension), and that effectively results in constant running time complexity. For detailed information about FGTS see [6, 7].

After selecting pairs of parents, the crossover operator is applied to them in order to produce offsprings. Provision of an effective penalty function for the unfeasible individuals implied that standard one-point crossover is used. This operator is performed with probability of $p_{cross} = 0.85$, meaning that approximately 85 % pairs of individuals exchange their genetic material.

Finally, a modified simple mutation operator is used to perform randomly selected gene changes in the genetic code of the individual, with certain mutation rate. During the GA execution it may happen that (almost) all individuals in the population have the same gene on certain position. These genes are called frozen. If the number of frozen genes is l, the search space becomes 2^l times smaller and the possibility of the premature convergence increases rapidly. Selection and crossover operators can not change the bit value of any frozen gene and basic mutation rate is often insufficiently small to restore the lost search space subregions. If the basic mutation rate is increased significantly, a genetic algorithm becomes random search. For this reason, the mutation rate is increased on frozen genes only. Therefore, in this implementation, mutation rate for frozen genes is 2.5 times higher ($1.0/|V|$) when compared to non-frozen ones ($0.4/|V|$).

3.3 Other GA Aspects

The population size is $N_{pop} = 150$ individuals. The steady-state generation replacement with elitist strategy is used. Initial population is randomly generated, providing maximal diversity of genetic material.

Two thirds of the population are directly passing to the next generation, i.e. the number of elite individuals is $N_{elite} = 100$. Genetic operators are applied on the rest of population, so that only one third of the population ($N_{pop} - N_{elite}$ individuals) is replaced in every generation. The objective value of every elite individual is calculated only once, providing significant time saving.

Duplicated individuals are removed from each GA generation. Their fitness values are set to zero so the selection operator prevents them to enter next generation. This method is very effective for keeping the desired level of genetic material diversity and keeping the algorithm away from premature convergence. Individuals with the same objective values, but different genetic codes may, in some cases, dominate in the population. If their codes are similar, GA can lead to local optimum. For that reason, it is useful to limit their appearance to some constant value N_{rv} (specifically, it was set to 40 in this GA application).

In order to prevent undeserved domination of elite individuals over the population, their fitness are decreased by applying the following formula:

$$f_{ind} = \begin{cases} f_{ind} - \overline{f}, & f_{ind} > \overline{f} \\ 0, & f_{ind} \le \overline{f} \end{cases}; \qquad 1 \le i \le N_{elite}; \quad \overline{f} = \frac{1}{N_{pop}} \sum_{i=1}^{N_{pop}} f_{ind}. \tag{2}$$

In this way, even non-elite individuals preserve their chance to survive to the next generation.

During the running of GA, some solutions are often generated repeatedly in subsequent generations. The classical GA evaluates each solution irrespectively of its repetition. To increase the efficiency, we used a caching technique [11, 12], memorizing all newly generated solutions with their objective values. In the case of a subsequent occurrence of a solution, the objective value can be quickly retrieved instead of performing a new evaluation, as long as the solution resides in the cache. The caching technique used applies a least-recently-used (LRU) strategy with a hash-queue data structure [12]. In this implementation the number of individuals stored in caching table is limited to constant $N_{cache} = 5\,000$.

4 EXPERIMENTAL RESULTS

The computational experiments were carried out on a PC based on an AMD Sempron CPU working at 1 597 MHz, and with 256 MB memory. The OS was Knoppix 3.7 Linux. The algorithm was implemented in C programming language.

To demonstrate the efficiency of the proposed genetic algorithm, two families of graph instances were used. Since there are no standard instances for MBCP, the instances for the k-cardinality tree problem (KCTP) that contains vertex-weights appropriate for our problem were generated by grid graph instance's generator, mentioned in [3].

The interior vertices of the grid graph have 4 neighbors, the vertices along the sides have 3 neighbors and only four of them have 2 neighbors (extreme vertices of the rectangle). Hence, the grid graphs are very sparse. An example of such graph is presented in Figure 3. According to [3, 4], these graphs represent the hardest test instances for KCTP. This fact does not necessarily mean that the mentioned grid graph instances are (too) difficult for our problem (MBCP).

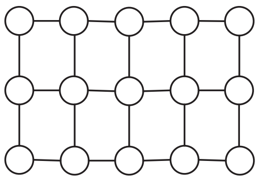

Fig. 3. An example of grid graph

We have constructed 16 instances for each grid graph with $n = 05 \times 05$ up to 15×15 vertices. In the first case (instance name is ended with letter a), the vertex weights were integers, chosen from a uniform distribution in the interval $[1, 100]$. In the second case (instance name is ended with letter b), random interval chosen was $[1, 500]$. The results of applying the proposed GA implementation on those instances are summarized in Table 1.

Our research was especially focused at the behavior of GA implementation on instances with weights defined as real numbers (non-integers). Since grid graphs have symmetric structure, it was interesting to test GA on the other (non-symmetric) random graph instances. So, we have generated connected random graphs with real vertex weights, chosen from a uniform distribution in the interval $(0, 100]$. Specified generation parameters are the number of vertices and the number of edges. The edges were generated by choosing among the non-existing ones available at the moment of generation. Once the graph is generated, a check if it is connected is run, and if it is not, the procedure is repeated until a connected graph has been created. Obviously, such process can be more time consuming than the only solution finding. Additionally, it is very hard to provide a graphic representation of such graphs due to their total randomness, i.e. effectively lacking any typical characteristics and inner structure specific to grid graphs. The results related to sparse graphs of various number of edges used in our experiments can be seen in Table 2.

As mentioned earlier, the parameters tending to be adequate and to demonstrate GA robustness for this problem were used. The maximal number of generations is $N_{gen} = 5\,000$. The algorithm stops if the best individual or the best objective value remains unchanged through $N_{rep} = 2\,000$ successive generations, respectively. Previous criterion allowed the convergence of GA to optimal solutions in all cases of grid graph instances. On random graph instances, GA also converged to very good solutions. Only minor improvements in the quality of final solutions can be expected when prolonging the runs, that can be seen from the t_{tot} and t columns of Tables 1–2. All experiments were repeated 20 times using different random number seeds.

Also, comparison between our GA implementation and balance implementation was made. The only method for creation of the latter was found in [5]. Block-articulation tree was done using the considerably efficient block-cut-tree implementation. Description of this method can be found in [10, 15], and its efficiency became obvious as per results presented in the $balance_t$ column. Balance method was then applied on the block-articulation tree generated in the above mentioned manner.

In first three columns, the instance's name, the number of vertices and edges are given, respectively. The next column (GA_{best}) contains the best solution of GA on the current instance. Average time needed to detect the best value is given in t column, while t_{tot} represents the total time (in seconds) needed to meet finishing criterion. On average, GA finished its execution in *gen* generations. The next two columns are related to the caching: *eval* represents the average number of evaluations, while *cache* displays savings (in percents) achieved by using caching technique. The results, got by testing the mentioned balance implementation, are given in the last two columns. $balance_{sol}$ represents the solution of balance algorithm and the column $balance_t$ contains time (in seconds) needed for executing of balance algorithm.

GA concept cannot prove optimality as adequate exit criteria. Therefore, as column t_{tot} in Tables 1-2 shows, our algorithm runs through additional $t_{tot} - t$ time (until finishing criteria is satisfied), although it already reached optimal/best solution.

As can be seen from Table 1, GA reaches optimal solutions in all cases on the grid graph instances, because the value of objective function $B(V_1, V_2) = |w(V_1) - w(V_2)| \geq 0$ and weights are integers. The obtained value of 0 is verified to be optimal in case of $w(V) = \sum_{i \in V} w(i)$ is even and value of 1 is verified to be optimal if $w(V)$ is odd.

The optimality of GA solutions, presented in Table 2, can not be verified, because GA is not an exact method but only metaheuristic. Therefore, these results seem to be promising because weights are chosen uniformly from interval $(0, 100]$ and balance is less than 0.02 in almost all cases.

Inst.	$\|V\|$	$\|E\|$	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	*eval*	*cache* (%)	$balance_{sol}$	$balance_t$ (sec)
05x05a	25	40	1	0.01	0.36	2 039	23 945	76.5	75	< 0.01
05x05b	25	40	1	0.02	0.37	2 113	22 949	78.4	215	< 0.01
05x06a	30	49	0	0.03	0.41	2 121	31 384	70.6	30	< 0.01
05x06b	30	49	1	0.05	0.43	2 223	31 289	71.7	241	< 0.01
05x10a	50	85	1	0.10	0.72	2 284	46 321	59.6	59	< 0.01
05x10b	50	85	0	0.30	0.91	2 894	59 114	59.2	36	< 0.01
05x20a	100	175	0	0.52	1.73	2 845	77 895	45.3	22	< 0.01
05x20b	100	175	1	0.62	1.77	2 939	79 153	45.9	103	0.01
07x07a	49	84	0	0.185	0.804	2 575	54 770	57.6	20	< 0.01
07x07b	49	84	1	0.135	0.743	2 425	49 687	58.9	209	< 0.01
07x10a	70	123	1	0.376	1.249	2 839	68 848	51.3	7	< 0.01
07x10b	70	123	0	0.306	1.186	2 676	64 921	51.5	28	< 0.01
10x10a	100	180	1	0.598	1.809	2 890	81 512	43.6	73	< 0.01
10x10b	100	180	1	0.426	1.633	2 678	72 829	45.7	213	< 0.01
15x15a	225	420	0	1.637	4.439	3 083	102 896	33.2	84	0.08
15x15b	225	420	0	2.132	4.669	3 349	108 240	35.3	290	0.07

Table 1. GA and balance results on grid-graph instances

Inst.	$\|V\|$	$\|E\|$	GA_{best}	t (sec)	t_{tot} (sec)	*gen*	*eval*	*cache* (%)	$balance_{sol}$	$balance_t$
rnd01	20	30	1.14274	0.19	0.52	3 079	37 560	75.6	298.89	< 0.01
rnd02	20	50	0.01965	0.12	0.49	2 609	43 500	66.7	283.79	< 0.01
rnd03	20	100	0.00626	0.25	0.68	3 155	55 751	64.7	338.45	0.01
rnd04	30	50	0.00915	0.17	0.58	2 841	47 652	66.5	332.88	< 0.01
rnd05	30	70	0.01036	0.20	0.65	2 879	55 696	61.6	133.94	< 0.01
rnd06	30	200	0.00029	0.32	0.93	2 962	66 923	54.8	35.64	< 0.01
rnd07	50	70	0.01608	0.31	0.92	3 010	62 175	58.9	2 172.22	< 0.01
rnd08	50	100	0.00531	0.39	1.10	3 131	70 911	54.9	3.22	< 0.01
rnd09	50	400	0.00024	0.88	1.92	3 410	90 822	46.3	272.12	0.01
rnd10	70	100	0.01328	0.59	1.40	3 298	77 341	53.2	1 347.74	< 0.01
rnd11	70	200	0.00023	0.784	1.737	3 438	87 888	48.9	228.12	0.01
rnd12	70	600	0.00232	1.275	2.910	3 325	102 261	38.1	331.02	0.02
rnd13	100	150	0.00859	0.954	2.032	3 405	90 886	46.8	2 378.97	< 0.01
rnd14	100	300	0.00137	0.924	2.355	3 216	91 851	42.9	125.77	0.02
rnd15	100	800	0.00155	1.472	3.812	3 169	103 405	35.1	176.28	0.05
rnd16	200	300	0.00574	3.216	4.972	4 232	133 738	36.9	3 821.85	0.04
rnd17	200	600	0.00058	2.567	5.255	3 535	116 925	34.4	40.58	0.16
rnd18	200	1500	0.00011	3.318	7.380	3 449	108 022	37.5	23 158.79	0.43
rnd19	300	500	0.00039	5.048	7.747	4 179	140 576	32.8	455.49	0.18
rnd20	300	1 000	0.00025	3.035	7.560	3 218	112 420	30.2	255.35	0.60
rnd21	300	2 000	0.00008	6.088	11.614	3 864	12 4274	35.5	51.25	1.21

Table 2. GA and balance results on random graph instances

The presented GA results clearly indicate very good quality of obtained solutions. Also, the overall running times (t_{tot}) are relatively short (not exceeding 12 seconds in the worst case). Although the optimality can not be proved in the case of random instances, we believe that our GA obtained high-quality solutions and that it represents significant contribution for solving MBCP.

Comparison of GA_{best} and $balance_{sol}$ shows that the latter implementation provides weaker results. Such outcome is expected since GA does not have gap upper bound ($gap = GA_{best} - opt$, where opt denotes optimal solution) as the balance method. Also, the value of gap upper bound in the balance implementation was very good. It is taken from [5] as well as the very quick heuristics.

5 CONCLUSION

In this paper we have described a GA approach to MBCP solving. This heuristic seems to work well for problems with up to 300 vertices and 2000 edges. The proposed GA obtained the optimal solutions for all available grid graph instances. For random instances, the obtained results are very promising, but we are not able to prove their optimality by GA. Computational experiments demonstrate the robustness of our GA implementation with respect not only to the quality of obtained solutions, but also to the running times.

Our work can be extended in several ways. Firstly, the investigation of the possibility of hybridization with some exact methods could be done. The genetic algorithm described in this paper can also be used to solve similar problems within the acceptable time limits.

Acknowledgement

This research was partially supported by the Serbian Ministry of Science and Ecology under project 144007. The authors are grateful to Christian Blum for grid graph instance's generator, Janka Chlebikova for information about the balance algorithm and Ivana Ljubic for the efficient block-cut-tree implementation. We also thank Igor Djurić, Zorica Stanimirović and anonymous referees for their useful comments on this paper.

REFERENCES

[1] Bäck, T.—Fogel, D. B.—Michalewicz, Z.: Evolutionary Computation 1: Basic Algorithms and Operators. Institute of Physics Publishing, Bristol-Philadelphia, 2000.

[2] Bäck, T.—Fogel, D. B.—Michalewicz, Z.: Evolutionary Computation 2: Advanced Algorithms and Operators. Institute of Physics Publishing, Bristol-Philadelphia, 2000.

[3] Blum, C.—Ehrgott, M.: Local Search Algorithms for the k-Cardinality Tree Problem. Discrete Applied Mathematics, Vol. 128, 2003, No. 2–3, pp. 511–540.

[4] Brimberg, J.—Urošević, D.—Mladenović, N.: Variable Neighborhood Search for the Vertex Weighted k-Cardinality Tree. European Journal of Operational Research, Vol. 171, 2006, pp. 74–84.

[5] Chlebíková, J.: Approximating the Maximally Balanced Connected Partition Problem in Graphs. Information Processing Letters, Vol. 60, 1996, pp. 225, pp. 230.

[6] Filipović, V.: Fine-Grained Tournament Selection Operator in Genetic Algorithms. Computing and Informatics, Vol. 22, 2003, No. 2, pp. 143–161.

[7] Filipović, V.: Selection and Migration Operators and Web Services in Parallel Evolutionary Algorithms (in Serbian). Ph. D. thesis, Faculty of Mathematics, University of Belgrade, 2006.

[8] Holland, J. H.: Adaptation in Natural and Artificial Systems. The University of Michigan Press, Ann Arbor, 1975.

[9] Juhos, I.—van Hemert, J. I.: Improving Graph Colouring Algorithms and Heuristics Using a Novel Representation. Lecture Notes in Computer Science, Vol. 3906, 2006, pp. 123–134.

[10] Kersting, S.—Raidl, G. R.—Ljubic, I.: A Memetic Algorithm for Vertex-Biconnectivity Augmentation. Applications of Evolutionary Computing: EvoWorkshops 2002, Vol. 2279, 2002, pp. 102–111.

[11] Kratica, J.: Improving Performances of the Genetic Algorithm by Caching. Computers and Artificial Intelligence, Vol. 18, 1999, No. 3, pp. 271–283.

[12] Kratica, J.: Parallelization of Genetic Algorithms for Solving Some NP-complete Problems (in Serbian). Ph. D. thesis, Faculty of Mathematics, University of Belgrade, 2000.

[13] 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.

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

[15] 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.

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

[17] Ljubić, I.—Weiskircher, R.—Pferschy, U.—Klau, G. W.—Mutzel, P.—Fischetti, M.: An Algorithmic Framework for the Exact Solution of the Prize-Collecting Steiner Tree Problem. Mathematical Programming, Series B, Vol. 105, No. 2–3, 2006, pp. 427-449.

[18] Puchinger, J.—Raidl, G. R.—Pferschy, U.: The Core Concept for the Multidimensional Knapsack Problem. Lecture Notes in Computer Science, Vol. 3906, 2006, pp. 195–208.

[19] Raidl, G. R.—Gottlieb, J.: Empirical Analysis of Locality, Heritability and Heuristic Bias in Evolutionary Algorithms: A Case Study for the Multidimensional Knapsack Problem. Evolutionary Computation, Vol. 13, 2005, No. 4, pp. 441–475.

[20] Stanimirović, Z.: Solving Some Discrete Location Problems by Using Genetic Algorithms (in Serbian). Master's thesis, Faculty of Mathematics, University of Belgrade, 2004.

[21] Stanimirović, Z.: An Efficient Genetic Algorithm for the Uncapacitated Multiple Allocation p-hub Median Problem. submited for Control and Cybernetics.

[22] Stanimirović, Z.: A Genetic Algorithm Approach for the Capacitated Single Allocation p-Hub Median Problem. Computing and Informatics, Vol. 27, 2008.

Brankica Djurić received her B. Sc. degree in mathematics (1999) from University of Belgrade, Faculty of Mathematics. Since 2000 she works as a research assistant at the Faculty of Mathematics. She is about to complete her M. Sc. studies in teaching mathematics and computer science, with thesis concerned with graph problems. Her research interests also include genetic algorithms and combinatorial optimization.

Jozef Kratica received his B. Sc. degrees in mathematics (1988) and computer science (1988), M. Sc. in mathematics (1994) and Ph. D. in computer science (2000) from University of Belgrade, Faculty of Mathematics. In 2002 he joined Mathematical Institute as a researcher. As a delegation leader, he participated in the International Olympiads in Informatics (IOI '90 Minsk – Belarus, IOI '93 Mendoza – Argentina). His research interests include genetic algorithms (evolutionary computation), parallel and distributed computing and location problems.

Dušan Tošić received his B. Sc. degree in mathematics (1972), M. Sc. in mathematics (1977) and Ph. D. in mathematics (1984) from the University of Belgrade, Faculty of Mathematics. Since 1985 he has been professor of computer science at the Faculty of Mathematics. His research interests include parallel algorithms, optimization and evolutionary computation, numerical solving of the differential equations and teaching computer science.

Vladimir Filipović received his B. Sc. degree (1993), M. Sc. (1998) and Ph. D. (2006) in computer science from University of Belgrade, Faculty of Mathematics. In 2006, he became assistant professor of computer science at the Faculty of Mathematics. His research interests include genetic algorithms, parallel algorithms and operational research.