Introduction to Genetic Algorithms and Some Applications

Jozef Kratica
Serbian Academy of Sciences and Arts,
Mathematical Institute
Kneza Mihajla 35/I, pp. 367
11 001 Belgrade, Yugoslavia
E-mail: jkratica@mi.sanu.ac.yu

Dušan Tošić
University of Belgrade
Faculty of Mathematics
Studentski trg 16
11000 Belgrade, Yugoslavia
E-mail: dtosic@matf.bg.ac.yu

Abstract

Genetic Algorithms (GAs) are inspired mainly from nature and became robust and effective methods for solving combinatorial and global optimization problems. In this paper the basic principles of GAs are presented. Also, the important parts of a GA implementation and its effectiveness illustrated by results on several real-life problems is described. This implementation includes several new methods for the improving of previous genetic algorithm implementation.

Keywords: Genetic algorithms, caching GA, location problems, biconnectivity augmentation, network design.

1. Introduction

GAs were inspired by the processes of natural evolution in early 1970s and were rigorously stated by Holland [6]. In next 25 years GAs were developed in many directions. After that, a lot of papers were published, but the basic statements of Holland's work are still correct.

GAs directly use analogy with natural evolution working with the population of individuals (usually 10-500). Each individual represents a one possible solution in the searching space for particular problem. The individuals are represented by genetic code, over a some finite alphabet. Every gene in genetic code represents a particular solution property of the problem. Usually each gene corresponds to one variable in solution.

The fitness value for each individual is computed (or assigned). This value indicates quality of the chosen individual. The average fitness of entire population may be a good measure of solution quality during the generations. GAs have to ensure the continually improvement of average fitness from generation to generation using the genetic operators. That also means the improvement of solution quality toward the optimal value. The basic genetic operators are: selection, crossover and mutation. In some situations may be used other operators such as: inversion, local search, etc.

The selection mechanism favorises highly fitted individuals and its highly fitted genes. These individuals have better chances for reproduction into the next generation than other ones. The least fitted members of population (also least fitted genes) have reduced chances for reproduction and successively wiped out from population.

The crossover operator divides population into the pairs of individuals named parents. For each pair of parents with some probability p_{cross}, the recombination of their genes is performed. The result of this operator is an undeterministic exchange of genes in population. In that way the good parents usually produce better offsprings. This mechanism gives the real chances for lower fitted individuals to keep in population and to improve by mutual recombination of good genes.

Multiple usage of selection and crossover (without mutation) results in loosing of genes variety and some regions of search space are not reachable. This usually causes the premature convergence in local optimum far from global optimal value. The mutation operator can help to avoid this shortcoming by randomly change of gene with small probability p_{mut}. The mutation is basic mechanism for restoring lost genes into the population. This increases the diversibility of genetic material and previously not reachable regions of search space may be reachable again.

The initial population usually is randomly generated, although in some situation, the population may be fully or partially generated by some initial heuristic. The second approach usually has problems with reduced diversibility of genetic material. It can produce better solutions in several starting generations, but later it gives worse results.

The basic description of GA is given in Figure 1; the number of individuals is denoted by N_{pop} and p_i is objective value of i-th individual.

```
Input_Data();
Population_Init();
while(! Finish_GA())
    {
        for(i=0; i<Npop; i++) pi = Objective_Function();
        Compute_Fitnesses();
        Selection();
        Crossover();
        Mutation();
    }
Output_Results();
```

Figure 1. Basic form of GA

2. The simple genetic algorithm

The basic variant of GA described in Holland's book [6] contains simple roulette selection scheme, one-point crossover and simple mutation. In literature (see: [1] and [5]) it is well known as Simple Genetic Algorithm (SGA).

2.1. Simple selection

Example 1. Let the individuals are represented by 2-bit strings with the objective values given in the second column:

00	0.2
01	0.3
10	0.5
11	0.7

If population contains the following individuals **00, 01, 01, 10, 11**, sum of their objective values is: 0.2 + 0.3 + 0.3 + 0.5 + 0.7 = 2.0. Their chances for reproduction into the next generation, respectively, are: 5 * 0.2 / 2.0 = 0.5; 5 * 0.3 / 2.0 = 0.75; 5 * 0.3 / 2.0 = 0.75; 5 * 0.5 / 2.0 = 1.25; 5 * 0.7 / 2.0 = 1.75 . The individuals go to the roulette with chances given by these values.

Example 2. Let us keep the individuals from previous example, but with different objective values:

00	0.2
01	0.3
10	8.0
11	9.0

In this case, for population **00, 00, 01, 01, 10**, the results are quite different than previous. The sum of their objective values is: 0.2 + 0.2 + 0.3 + 0.3 + 8.0 = 9.0 and their chances are, respectively, equal to 5 * 0.2 / 9.0 = 0.11; 5 * 0.2 / 9.0 = 0.11; 5 * 0.3 / 9.0 = 0.17; 5 * 0.3 / 9.0 = 0.17; 5 * 8.0 / 9.0 = 4.44. Note that the last individual has chance 7 times greater than overall chances for other individuals. In that case, the individual **10** dominates over the population, although it is not globally optimum. Still, it discards all other individuals from population and GA prematurely converges in local optimum.

2.2. One-point crossover

As we mentioned previous, the crossover operator divides population into the pairs of individuals and performs recombination of their genes with the probability p_{cross}. For every pair of individuals one position in individual genetic code is chosen. All genes after that position are exchanged among individuals as it can be seen in Figure 2.

before crossover	***after crossover***
XXX | XXXXX	XXX | **YYYYY**
YYY | **YYYYY**	**YYY** | XXXXX

Figure 2. Scheme of one-point crossover

Example 3. Let p_{cross} = 0.85 and population contains 5 individuals: **01011, 10001, 00100, 10100, 11001**. Suppose that the (first, fifth) and (second, fourth) individuals are chosen and crossover is performed for both of them. If for the first pair position 2 is chosen and for the second pair position 4. Then the results are given as follows:

first pair (*first, fifth*)		***second pair (*second, fourth*)***	
before ------> *after*		before ------> *after*	
0 1 | 0 1 1	0 1 0 0 0	1 0 0 0 | 1	1 0 0 0 0
1 1 | 0 0 0	1 1 0 1 1	1 0 1 1 | 0	1 0 1 1 1

After crossover population will be: **01000, 10000, 00100, 10111, 11011**

2.3. Simple mutation

The simple mutation operator processes every gene and muted it with small probability p_{mut}. In this approach probability p_{mut} is equal for all individuals.

Example 4. Let be p_{mut} = 0.2 and binary genetic code of some individual: **010111000101**. The mutation process in this case is presented below, where character * marks position for mutation.

individual (genetic code) before mutation

```
      *               *
0 1 0 1 1 1 0 0 0 1 0 1
```

individual after mutation

```
0 1 1 1 1 1 0 0 0 0 0 1
```

3. Advanced techniques

In previous section SGA is presented and some of its deficiencies are mentioned. Two most known of deficiencies are: prematurely convergence and lost of genetic material. They imply unsatisfactory quality of obtained solution. In this section some methods for avoiding of deficiencies will be presented.

3.1. Generation replacement policy

In every generation of SGA entire population is replaced with new individuals through selection, crossover and mutation. This variant is named generational GA and ensures maximal gradient in genetic search. Unfortunately, it has not an absolute mechanism for preserving the excellent individuals from unlucky applying of some genetic operator. If the good solution is destroyed, it has to be reexplored again by genetic search, but the running time is wasted.

Therefore some number of individuals may skip selection or even all genetic operators going directly into the next generation. These approaches are named steady-state generation replacement policy and elitist strategy. They provide smaller gradient in genetic search, but preserve good individuals from past generations becoming more powerful than generational GA.

In the case of steady-state replacement with elitist strategy a part of population skips all genetic operators and their objective values are evaluated only in the first generation. In all subsequent generations they are directly proceeded and reevaluation is not necessary. Since the objective value function is usually most computationally expensive part of GA, the elite individuals are obtained very cheaply. In the other side, important individuals or genes can be preserved by this policy.

3.2. Selection

The rank-based selection scheme is a mechanism for avoiding of simple roulette selection shortcomings. Before the GA starting, the rank value for each individual in population is defined. Afterward, the population is arranged in decreasing order of objective values and the ranks become their fitness values in every generation. This approach distributes fitness onto the population only using the order of individuals not its real objective values. Using the rank-based selection effectively prevents possibility of premature convergence.

Example 5. Assume that individuals are same as in Example 2. with ranks 2.5, 2, 1.8, 1.5, 1.2. After arranging the population in decreasing order of objective values, the individuals: 00, 00, 01, 01, 10 have fitness, respectively: 1.5, 1.2, 2, 1.8, 2.5. In next generation the population has at least 3 individuals (expected number is 5*(1.5+1.2+2+1.8) / (1.5+1.2+2+1.8+2.5) = 5*6.5/9 =3.6) from set {00,01}. This leaves the real chances for possible crossover of individuals 10 and 01 on the first position producing the optimal individual 11. The mentioned approach favourises best individual in population, but prevents its domination and other individuals have real chances for their reproduction.

The other successful selection method for avoiding premature convergence is the tournament selection. The tournament with N_{tourn} individuals is performed for every free position that forming population in next generation. The winner of tournament is best-valued (fitted) individual and it is selected into the next generation. All other individuals are losers on that tournament. The tournament members are randomly generated and this approach gives chances to all individuals in population, except the worst N_{tourn}-1.

Example 6. For population from Example 2., N_{tourn} = 2 and tournament members {00,01}; {00,00}; {01,10}; {01,00}; {10,10}, the population in next generation will be: 01, 00, 10, 01, 10. This is significantly diverse compared to simple roulette selection from Example 2.

The fine-grained tournament selection represents improvement of basic tournament selection scheme when integer value of N_{tourn} is not adequate. For example if N_{tourn} = 2 is small and N_{tourn} = 3 is large, compromise will be the number $N_{tourn} \in (2, 3)$. This is impossible for particular tournament because N_{tourn} must be integer, but if some tournaments have int(N_{tourn}) and other int(N_{tourn})+1 members, average value N_{tourn} is rational. This selection scheme is explained in details in [3].

3.3. Crossover

In two-point crossover, the individuals of population are also grouped into the pairs. Moreover, two positions are randomly chosen, instead of only one position in one-point crossover. After that, individuals exchange genes between these positions as can be seen from Figure 3.

before crossover	***after crossover***
XXX | XXX | XX	XXX | **YYY** | XX
YYY | **YYY** | **YY**	**YYY** | XXX | **YY**

Figure 3. Scheme of two-point crossover

Example 7. Two-point crossover applied on population from Example 3, with positions 2,4 for first pair and 3,4 for second pair, gives results as follows:

first pair (*first, fifth*)		***second pair (*second, fourth*)***	
before ------> *after*		before ------> *after*	
0 1 | 0 1 | 1	0 1 0 0 1	1 0 0 | 0 | 1	1 0 0 1 1
1 1 | 0 0 | 0	1 1 0 1 0	1 0 1 | 1 | 0	1 0 1 0 0

After crossover population will be: **01001, 10011, 00100, 10100, 11010**

Differently from previous crossover operators with collective gene exchanging, the uniform crossover individually resolves whether particular gene position will be exchanged or not. The binary mask is firstly generated; if this mask has value 1 on some position, corresponding genes stay, otherwise they are exchanged between individuals as can be seen from Figure 4.

mask
101101001001
before crossover
XXXXXXXXXXXX
YYYYYYYYYYYY
after crossover
X**Y**XX**Y**X**YY**X**YY**X
YX**YY**X**Y**XX**Y**XX**Y**

Figure 4. Scheme of uniform crossover

Example 8. Using the population from Example 3 and performing the uniform crossover with $p_{cross} = 0.45$ and probability of exchanging genes $p_{unif} = 0.6$ (in case that crossover is performed only to first pair with mask 11001) we get:

first pair (*first, fifth*)		***second pair (*second, fourth*)***
mask		(*no crossover*)
1 1 0 0 1		
before	------> *after*	
0 1 0 1 1	**0 1 0 0 1**	
1 1 0 0 0	**1 1 0 1 0**	

After crossover, the population will be: **01001, 10001, 00100, 10100, 11010**

3.4. Caching

The idea of caching is used for avoiding the attempt to compute the same objective value repeatedly many times. Instead of that, the objective values are remembered and reused. If the program has computed objective value for a particular item string, and the same string appears again, the cached values are used to avoid computing twice. The simple, but effective Least Recently Used (LRU) strategy is used for caching GA. Basic description of caching GA technique is given in Figure 5. Here the program segment replaces *Objective_Function*() from Figure 1.

```
if  Belong(individual, cache_memory) Set_Value(individual, cache_block);
else {
        Objective_Function();
        if Full(cache_memory) Remove(cache_memory, oldest_cache_block);
        Put(cache_memory, individual);
     }
newest_cache_block_position = Position(individual);
```

Figure 5. Caching GA technique

Caching is implemented by hash-queue data structure. More information about caching GA can be found in [10] and [12].

4. Some GA applications

4.1. Simple plant location problem

The simple plant location problem (SPLP) is a well-known NP-hard combinatorial optimization problem and basic member in family of location problems. The problem is also known as uncapacitated warehouse location problem or uncapacitated facility location problem.

Consider a set $I = \{1,...., m\}$ of candidate sites for facility location, and a set $J=\{1,...., n\}$ of customers. Each facility $i \in I$ has a fixed cost f_i . Every customer $j \in J$ has a demand b_j, and c_{ij} is the unit transportation cost from facility i to customer j.

It has to be decided:

- facilities to be established and
- quantities to be supplied from facility i to customer j

such that the total cost (including fixed and variable costs) is minimized.

Mathematically, the SPLP is formulated by (1)-(3):

$$\min \left(\sum_{i=1}^{m} \sum_{j=1}^{n} c_{ij} x_{ij} + \sum_{i=1}^{m} f_i y_i \right) \qquad (1)$$

subject to:

$$\sum_{i=1}^{m} x_{ij} = 1, \qquad \text{for every } j \in J; \qquad (2)$$

$$0 \le x_{ij} \le y_i \text{ and } y_i \in \{0,1\}, \qquad \text{for every } i \in I \text{ and every } j \in J; \qquad (3)$$

where:

x_{ij} represents the quantity supplied from facility i to customer j,

y_i indicates whether facility i is established ($y_i = 1$) or not ($y_i = 0$).

The first SGA method for solving SPLP was proposed in [8] solving ORLIB instances ([2]) in reasonably short time. It is significantly improved using better objective value function, caching GA ([11]) and Add-heuristic ([9]). GA implementation capable for solving large-scale instances (over 2000 facilities and customers) is described in [12] and [14].

In that approach the population size is 150 individuals and population is randomly initialized. Binary representation of facility sites and efficient objective value function is used. Rank based selection is performed with rank 2.5 for best individual down to 0.7 for worst one, by step 0.012. This selection scheme successfully prevents premature convergence in local optimum and loosing the genetic material. The uniform crossover and simple mutation are used with steady-state generation replacement and elitist strategy. In each generation only 1/3 of population is replaced, with 2/3 of population that is directly passed into the next generation. The rank based selection scheme is compared with fine grained tournament selection in [4] and fine grained tournament selection improved overall results.

The other GA method adapted with local search and clustering technique was proposed in [7] and tested on ORLIB instances. Comparative results on ORLIB instances show that our GA-implementation gives better quality of solutions. Also, the run-time is about 10 times faster estimated for equal CPU, as can be seen from Table 1.

Table 1. Experimental results for SPLP

Instances	Size	GANP Rank-B. 486/133	GANP RankB. PIII/600	GANP FGT PIII/600	Horng Pent. 166
41-74	16×50	260/260 0.86	0.10	0.05	200/200 2.883
81-104	25×50	240/240 1.26	0.17	0.18	180/200 4.48
111-134	50×50	198/240 2.97	0.51	0.55	173/200 8.296
A-C	100×1000	42/60 83.1	16.28	12.24	83/150 443.61
MO	100×100	93/100 5.49	0.59	0.35	-
MP	200×200	100/100 16.60	1.18	0.51	-
MQ	300×300	100/100 34.97	2.69	0.93	-
MR	500×500	99/100 93.69	4.68	2.52	-
MS	1000×1000	100/100 379.6	23.18	14.47	-
MT	2000×2000	100/100 1812	-	-	-

4.2. Biconnectivity augmentation

During the designing of communication networks, the robustness against failures is a very important issue. The terms *vertex-biconnectivity* and *edge-biconnectivity* are used to describe this kind of robustness in graph theory. If any vertex (edge) is broken, this graph remains connected. Biconnectivity augmentation is process that converts any graph into the biconnected one with minimal cost. In both cases (vertex or edge) it is NP-hard problem.

GA method for solving the vertex biconnectivity augmentation problem (V2AUG) is considered in [16] and [17]. Similar approach is used in ([18]) for solving the edge biconnectivity augmentation (E2AUG). The population size is N_{pop}=150, binary coding is performed, steady-state generation replacement with elitist strategy is used and uniform crossover, simple mutation and caching are implemented. For this problem tournament selection and exponential change of mutation rate p_{mut} gave best results. The first approach uses discarding of invalid individuals, second performs additional repair heuristic converting graphs with articulation points into the biconnected graphs. The obtained results for vertex and edge case are given in Table 2. and Table 3.

Table 2. Experimental results for V2AUG on AMD K6/266MHz

Instances	Size (nodes × edges)	GA (gap – n.gener. - time)	GA + imp. heur
A1-A5	20-40 × 43-79	0 – 25 - 0.7 s	0 - 8 - 0.5 s
B1-B5	20-70 × 81-150	0.4% - 107 - 3.8 s	0 - 22 - 2.6 s
C1-C4	20-100 × 205-248	0 – 970 - 16.4 s	0 - 128 - 35.6 s
D1-D5	40-100 × 384-497	3.9% - 11977 - 556.3 s	0 - 513 - 549.3 s
M1-M3	70-90 × 312-438	2.9% - 10639 - 614.3 s	0 - 128 - 67.7 s
N1-N2	100-110 × 1203-1270	27.1% - 21221 - 1520 s	0 - 1176 - 1312 s

Table 3. Experimental results for E2AUG on P III/500MHz

Instances	GA (gap - n.gener. - time)	GA + imp. heur
A1-A5	0 - 73 - 0.18 s	0 - 2 - 0.02 s
B1-B5	0.3% - 520 - 1.71 s	0 - 40 - 0.87 s
C1-C4	0.4% - 2165 - 8.21 s	0 - 32 - 1.54 s
D1-D5	9.5% - 6853 - 42.74 s	0.17% - 123 - 10.06 s
M1-M3	5.9% - 7308 - 48.20 s	0 - 183 - 10.99 s
N1-N2	26.7% - 14616 - 210.47 s	2.0% - 2157 - 387.66 s

4.3. Problem of designing a spread-spectrum radar polyphase code

This problem belongs to class of continuous min-max global optimization. It arises from the synthesis of radar polyphase codes and it is NP-hard. The spread-spectrum radar polyphase code design (SSRPCD) problem can be mathematically formulated by (4)-(7):

$$\textit{global min}\ \ f(x) \qquad x \in X \tag{4}$$

$$f(x) = \max \{ |\varphi_1(x)|, |\varphi_2(x)|, ..., |\varphi_m(x)| \} \tag{5}$$

$$X = \{(x_1, x_2, ..., x_n) \in R^n \mid 0 \le x_j \le 2\pi,\ j=1, 2, ..., n\} \tag{6}$$

where m = 2n-1 and

$$\varphi_{2i-1}(x) = \sum_{j=i}^{n} \cos\left(\sum_{k=|2i-j-1|+1}^{j} x_k \right), \quad i=1, 2, ..., n \tag{7}$$

$$\varphi_{2i}(x) = 0.5 + \sum_{j=i+1}^{n} \cos\left(\sum_{k=|2i-j|+1}^{j} x_k \right), \quad i=1, 2, ..., n-1$$

One GA technique is proposed in [13] successfully solving problem up to n=20. The comparison of results obtained by various techniques for solving this problem is also presented in [13].

Similarly to previous problems, the population has 150 individuals, steady-state generation replacement with elitist strategy, rank-based selection and simple mutation are used. Run-time performance of implementation is also improved by caching technique. Real valued binary code improved by Gray codes is used. The mutation rate depends on problem size and decreases exponentially during the generations. Experimental results on PIII/600MHz are shown in Table 4.

Table 4. Experimental results for SSRPCD

n	Best solution	No. gener	Time (sec)
2	0.3852	3980	1.64
3	0.2610	6000	2.83
4	0.0560	8000	4.53
5	0.3374	10000	6.71
6	0.4645	12000	9.79
7	0.5232	14000	13.64
8	0.4328	16000	18.48
9	0.3386	18000	24.23
10	0.4709	20000	31.52
11	0.6387	22000	40.11
12	0.6786	24000	50.06
13	0.8307	26000	61.93
14	1.0042	28000	75.58
15	0.9674	30000	91.43
16	1.0385	32000	109.02
17	1.0984	34000	129.36
18	1.3503	36000	152.41
19	1.2291	38000	177.86
20	1.4753	40000	206.45

4.4. Other problems

The possibility of applying our GA implementation on several other combinatorial optimization problems is currently in investigation. We consider the following: Index selection problem, Uncapacitated network design, Satisfiability problem.

This GA implementation is also used for solving some other kind of problems: Geophysical inversion problem ([20]) and Hydraulic pipeline design ([15] and [19]).

5. Conclusions and future work

GA-based approach for solving several quite different problems is presented in this paper. Although the problems have different nature, the most part of GA implementation is common for all proposed problems. Caching GA technique and fine-grained tournament selection are completely new techniques providing significant improvement of our GA implementation.

The results of this GA implementation obtained on presented problems are comparable to the other GA implementations. In some cases, the obtained results are better. The problems described in this paper could be solved by hybridization of GA with the other methods. The parallelization of GA is interesting extension of our implementation. Also, the applying to the similar combinatorial and global optimization problems is possible.

References

[1] Beasley D., Bull D.R. and Martin R.R., "An Overview of Genetic Algorithms, Part 1: Fundamentals", *University Computing*, Vol. 15, No. 2, pp. 58-69, 1993.

[2] Beasley J.E., "Obtaining Test Problems via Internet", *Journal of Global Optimization*, Vol. 8, pp. 429-433, 1996. http://mscmga.ms.ic.ac.uk/info.html

[3] Filipović V., "Proposition for Improvement of Tournament Selection Operator in Genetic Algorithms", M.Sc. thesis, University of Belgrade, Faculty of Mathematics, 1998. (in Serbian)

[4] Filipović V., Kratica J., Tošić D. and 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*, pp. 152-158, September 2000. http://www.matf.bg.ac.yu/vladaf/Works/paper35/paper35.ps

[5] Goldberg D.E., *Genetic Algorithms in Search, Optimization and Machine Learning*, Addison-Weseley Publ. Comp., Reading, Mass., 412p, 1989.

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

[7] Horng J.T., Lin L.Y., Liu B.J. and Kao C.Y., "Resolution of Simple Plant Location Problems Using an Adapted Genetic Algorithm", *Proceedings of the Conference on Evolutionary Computation - CEC'99* pp. 1186-1193, 1999.

[8] Kratica J., Filipović V., Šešum V. and Tošić D., "Solving of the Uncapacitated Warehouse Location Problem Using a Simple Genetic Algorithm", *Proceedings of the XIV International Conference on Material Handling and Warehousing*, pp. 3.33-3.37, Belgrade, Yugoslavia, 1996. http://www.geocities.com/jkratica/cmhw96.pdf

[9] Kratica J., Filipović V. and Tošić D., "Solving the Uncapacitated Warehouse Location Problem by SGA with Add-heuristic", In: XV ECPD International Conference on Material Handling and Warehousing, pp. 2.28-2.32, Belgrade, Yugoslavia, 1998. http://www.geocities.com/jkratica/cmhw98.pdf

[10] Kratica J., "Improving Performances of the Genetic Algorithm by Caching", *Computers and Artificial Intelligence,* Vol. 18, No. 3, pp. 271-283., 1999.

[11] 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), pp. 390-402, Springer-Verlag London Limited, 1999.

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

[13] Kratica J., Tošić D., Filipović V. and Ljubić I., "Genetic Algorithm for Designing a Spread-Spectrum Radar Polyphase Code", *Proceedings of the 5th Online World Conference on Soft Computing Methods in Industrial Applications - WSC5*, pp. 191-197, September 2000. http://www.matf.bg.ac.yu/vladaf/Works/paper36/paper36.pdf

[14] Kratica J., Tošić D., Filipović V. and Ljubić I., "Solving the Simple Plant Location Problem by Genetic Algorithms", Accepted for publication in *RAIRO Operations Research* journal, 2001.

[15] Kratica J., Stojanović Ž., Stojanović Z., "The Application of Genetic Algorithm to Hydraulic Pipeline Design", In preparation.

[16] Ljubić (Trmčić) I., "Application of Genetic Algorithms in Graph Connectivity Problems", M.Sc. thesis, University of Belgrade, Faculty of Mathematics, 2000. (in Serbian)

[17] Ljubić I. and Kratica J., "A Genetic Algorithm for the Biconnectivity Augmentation Problem", *Proceedings of the Conference on Evolutionary Computation - CEC 2000*, pp. 89-96, San Diego, CA, USA, July 16-19, 2000.

[18] Ljubić I., Raidl G.R. and Kratica J., "A Hybrid GA for the Edge-Biconnectivity Augmentation Problem", *Proceedings of the Parallel Problem Solving from Nature VI Conference - PPSN 2000*, Paris, France, *Springer Lecture Notes in Computer Science* 1917, pp. 641-650, September 2000.

[19] Stojanović Z., Stojanović Ž. and Kratica J., "The Influence of the Hydropower Plant Penstock Profile Upon the Penstock Mass", *Mobility & Vehicle Mechanics*, Vol. 23, No. 4, pp. 78-87, December 1997.

[20] Šešum V., Kratica J. and Tošić D., "Solving inverse geophysical problem by genetic algorithm", *Yugoslav Journal of Operational Research - YUJOR*, Vol. 10, No. 2, pp. 283-292, 2000.