Genetic Algorithm for Designing a Spread-Spectrum Radar Polyphase Code

Jozef Kratica[1], *Dušan Tošić*[2], *Vladimir Filipović*[2] and *Ivana Ljubić*[3]

[1] Serbian Academy of Sciences and Arts
Mathematical Institute
Kneza Mihaila 35/I, 11001 Belgrade, p.p. 367
YUGOSLAVIA
jkratica@mi.sanu.ac.yu
URL: http://www.geocities.com/jkratica/

[2] University of Belgrade
Faculty of Mathematics
Studentski trg 16,. 11000 Belgrade
YUGOSLAVIA
{dtosic | vladaf}@matf.bg.ac.yu

[3] Vienna University of Technology
Institute for Computer Graphics
Favoritenstrasse 9 11/186, A-1040 Vienna
AUSTRIA
ljubic@apm.tuwien.ac.at
URL: http://www.apm.tuwien.ac.at/people/Ljubic.html

ABSTRACT

A genetic algorithm (GA) is proposed for solving one continuous min-max global optimization problem arising from the synthesis of radar polyphase codes. GA implementation is successfully applied to this problem and it solves instances with n up to 20. Computational results are compared to other algorithms (implicit enumeration technique, Monte Carlo method and tabu search). In some cases, GA outperforms all of them.

1. INTRODUCTION

Problems of optimal design are important part of the global optimization. In [1] is proposed one such engineering problem: spread-spectrum radar polyphase code design (SSRPCD).

1.1. Problem Definition

The SSRPCD problem can be formulated as follows:

$$\textit{global min } f(x) \qquad (1)$$
$$x \in X$$

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

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

where m = 2n-1 and

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

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

Problem also can be formulated using by additional m functions:

$$f(x) = \max \{\varphi_1(x), \varphi_2(x), ..., \varphi_{2m}(x)\} \qquad (2)$$

where

$$\varphi_{m+i}(x) = -\varphi_i(x), \quad i=1, 2, ..., m$$

In GA approach can be used formulation (1) with only m=2n-1 functions. The objective function of this problem for the case *n* = 2 is illustrated in Fig. 1.

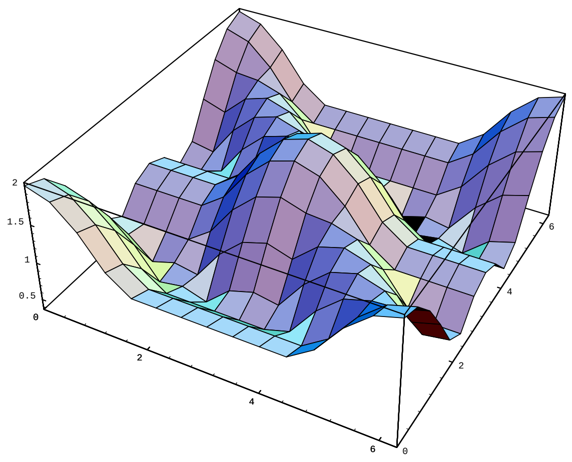


Fig. 1. Objective function for *n* = 2

Note that function is not differentiable and SSRPCD problem is inappropriate for solving by many numerical optimization methods. In addition, it has been proved that this problem is NP-hard [2], and requires some other methods for solving it. Because GA do not require differentiability and is successfully applied for solving some NP-complete problems, it may be good choice for solving this problem, as can be seen in section 3.

1.2. Literature Survey

Formal mathematical formulation and engineering background of this problem is given in [1]. In that paper is also given a technique for solving SSRPCD problem by nonlinear programming. Improved results of that heuristic method are presented in [3].

A one optimal technique for solving SSRPCD problem by implicit enumeration is described in [4]. Although the results in [4] show that the exact method was very successful on problems of linearly constrained separable concave minimization, it is not very suitable for this problem. By implicit enumeration are optimally solved only instances of smaller size (n = 2,3,4,5). Running time for proposed method grows exponentially, so this technique is not capable to compute solutions for n > 5.

In [2, 5] is proposed a Multi Level Tabu Search (MLTS) strategy for solving this problem, where each level is characterized by a different size of the move step. Satisfactory results are obtained with the two- and three-level strategy for dimensions not greater than 10. Results are also compared to standard Monte Carlo method and MLTS gave much better results with respect both to the objective function value and to the computational effort.

1.3. Genetic Algorithms

Genetic algorithms (GAs), the development of which is inspired by the analogy of natural evolution model, are robust and adaptive methods for solving search and optimization problems. GAs traditionally works with a population of items, as candidate solutions of search space. Population most commonly contains 10-200 items, that are usually fixed-length strings over some given, finite alphabet. Fitness function is defined to evaluate the quality of the items in population. The goal of the GA is to produce better items by using the stochastic genetic operators. Most commonly used genetic operators are selection, crossover and mutation.

The selection operator that produces a new population favors the reproduction of the items of better fitness value more than the one of the worse ones. As children inherit pieces of their gene patterns from parents, GA implicitly exploits the correlation between inner structure and quality of items. Crossover applies to one or more parents and exchange genetic material between them, with the possibility that good parents can generate better new candidates (children). Mutation involves the modification of the value of each solution gene with some small probability p_{mut}. The role of mutation is to prevent the premature convergence of the GA to suboptimal solutions, by restoring lost or unexplored genetic material into the population.

Initial population is randomly initialized in most cases, but some GA implementations allow some heuristic initialization of the population. The basic principles of GAs were first laid down rigorously by Holland [6] and later are well described in many texts. Some of interesting survey articles are [7, 8, 9, 10]. Informal description leads to the rough outline of a GA is given in Fig. 2.

```
Input_Data();
Population_Init();
while not Finish() do
    /* $N_{pop}$ is the number of individuals in a
        population $p_i$ is objective value of $i$th
        individual.*/
    for $i$ := 1 to $N_{pop}$ do $p_i$: = Objective_Function($i$)
    endfor;
    Fitness_Function();
    Selection();
    Crossover();
    Mutation();
endwhile
Output_Data();
```

Fig. 2. Pure genetic algorithm

Extensive studies of solving the combinatorial optimization problems with focus to NP-hard problems, through GAs, are given in [11, 12, 13, 17]. For detailed overview of this scientific area, reader may consult some comprehensive bibliographies, for example [14, 15].

2. GA IMPLEMENTATION

Outline of GA implementation for solving SSRPCD problem is schematically described in Fig. 3.

```
Input_GA_Parameters();
Input_SPLP_Data();
Population_Init();
while not Finish() do
    for $i := N_{elite}+1$ to $N_{pop}$ do
        /*   $N_{elite}$ - number of elite individuals
         string(i) – genetic code of i-th individual  */
        if Contain(cache_memory, string(i))
        then  $p_i$ := Get(cache_memory, string(i));
        else
            $p_i$ := Objective_Function(i);
            Put(cache_memory, string(i));
        endif
    endfor
    Fitness_Funct_and_Rank_Based_Selection():
    One-Point_Crossover();
    Simple_Mutation();
endwhile
Output_Data();
```

Fig. 3. Outline of GA implementation

2.1. Representation

Real valued binary code is used for encoding variables x_k (k = 1, 2, …, n). Each of them is represented by one 32-bit binary number. This representation is improved by Gray codes, because they are useful in combination with applied genetic operators. In practice, this method produced better result than pure real valued binary code, but differences is not too big.

2.2. Fitness function and selection

Rank-based selection is applied as selection operator, with rank value that decreases linearly from the best individual $r_{best} = 2.5$ to the worst individual $r_{worst} = 0.712$ by step of 0.012. In this rank-based selection scheme fitness of particular individual is equal to its rank, and then individuals go to the roulette, with chance proportional to its fitness, as can be seen in Fig. 4.

```
QSort(population);
for i: =1 to $N_{pop}$ do
   /*   $f_i$ -  fitness of the i-th individual  */
   if Duplicate(string (i)) then  $f_i$ := 0
                                  else $f_i$ := rank(i);
   endif
endfor
$\bar{f}$ := Sum_Fitnesses(population) / $N_{pop}$;
for i := 1 to $N_{elite}$ do
      if $f_i \geq \bar{f}$  then $f_i := f_i - \bar{f}$
              else $f_i$ := 0;
      endif
endfor
Roulette_Selection(population);
```

Fig. 4. Fitness function and rank-based roulette selection

In GA implementation for solving SSRPCD problem, this selection scheme successfully prevents premature convergence in local optima and loosing the genetic material. This is large improvement when it is compared to experimental results for simple roulette selection, which concords with the direction in literature ([9] and [16]).

2.3. Crossover

One-point crossover is performed (see Fig. 5.), that provides minimal disruption of individual's genes. It is important part for success of GA, because interaction among genes in individual's gene code is not too big. Crossover rate is $p_{cross} = 0.85$, and approximately 85% pairs of individuals exchanged their genetic material.

```
Crossover position       ↓
1. parent             XXXXXX
2. parent             YYYYYY
1. offspring          XXXYYY
2. offspring          YYYXXX
```

Fig. 5. One-point crossover

2.4. Mutation

The simple mutation is used, but for faster execution, it is performed by using a Gausian (normal) distribution. Let $N_{mut} = (N_{pop} - N_{elite}) * p_{mut}$ be an average number of mutations in population and $\sigma^2 = (N_{pop} - N_{elite}) * p_{mut} * (1 - p_{mut})$ a standard deviation. Number of mutations is generated by a random pick in Gausian (N_{mut}, σ^2) distribution. After that, positions of mutation sites in the population strings are randomly generated and their number being the same as the number of mutations.

Only the muted genes are processed by this method, while other genes in population are not. Number of muted genes is relatively small when compared to entire population, which improves the run-time performance of a simple mutation operator without changing its nature.

Mutation rate p_{mut} depends on problem size and changes as GA run, because the diversity of the genetic material is large at the first generation and decreases with the time. This adaptation of mutation rate promotes a fast convergence to good solutions during the starting generations and introduces more diversity for escaping from local optima during later stages. The mutation rate for instance of dimension n at generation t is given by formula (3):

$$p_{mut}(n, t) = \frac{0.15}{n} + \frac{0.25}{n} \cdot 2^{-t/1000} \qquad (3)$$

Example 1. For instance $n = 5$, in first generation is $p_{mut}(5, 0) = 0.03 + 0.05 = 0.08$, after 1000 generations $p_{mut}(5, 1000) = 0.03 + 0.025 = 0.055$ and at the end $p_{mut}(5, 10000) = 0.03 + 0.0000488 = 0.0300488$.

As we see from experimental results presented in Table 1, this mutation rate is good compromise between exploration and exploitation components of GA search applied to our problem.

2.5. Generation Replacement Policy

The population size is $N_{pop} = 150$ individuals. Steady-state generation replacement with elitist strategy is used. In each generation, only 1/3 of population (50 individuals) is replaced. Remaining 2/3 of population is directly passed to the next generation. Those elitist individuals do not need reevaluation (their objective value is already evaluated).

Every elite individual is passed directly into the next generation, giving one copy of it. To prevent undeserved domination of elite individuals over the population their fitness are decreased by formula (4):

$$f_i = \begin{cases} f_i - \bar{f}, & f_i > \bar{f} \\ 0, & f_i \le \bar{f} \end{cases} \qquad 1 \le i \le \quad N_{elite} \qquad (4)$$

where $\bar{f} = \frac{1}{N_{pop}} \cdot \sum_{i=1}^{N_{pop}} f_i$ is average fitness in entire population.

In this implementation, duplicate individual strings in population are discarded, and more diversity of the population is maintained to avoid premature convergence. Particular individual is discarded by setting its fitness to zero. Therefore, the duplicate individual strings are not removed physically but their occurrence is discarded in next generation.

This technique will be effective against the lack of variety in the population, maintaining maintain the diversity of genetic material. Discarding of duplicate strings decreases the appearance of dominating items in population, and effectively decreases the possibility of premature convergence in local optimum. This is a very important factor of successful working GA, particularly in cases of large size test problems.

Example 2. If population contains 5 items *0101, 1010, 0101, 0001, 1010* with fitnesses f_1, f_2, f_1, f_4, f_2 respectively then after this phase they are equal to f_1, f_2, *0*, f_4, *0*.

2.6. Other Parameters

Initial population is randomly initialized. Initialization by some heuristics is experimented and fitnesses in first generation are better, but gradient in the fitness function of subsequent generations is significantly smaller, and overall obtained results are worse.

The maximal number of generation is $N_{gener} = 2000*n$. The finishing criterion is also based on their number of consecutive generations with unchanged best individual. If that number exceeds value $N_{repeat} = 1000*n$, execution of GA is stopped.

2.7. Caching the GA

Finally, run-time performance of the GA is improved by caching technique. The idea of caching is used to avoid the attempt to compute the same objective value repeatedly many times. Instead of that, objective values are remembered and reused. If 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 its objective value.

Least Recently Used (LRU) strategy, which is simple but effective, is used for caching GA. Caching is implemented by hash-queue data structure. More information about caching GA can be found in [17, 18].

3. COMPUTATIONAL RESULTS

In this section are given results obtained by GA implementation, and compared to other methods mentioned in section 1.2. Optimal solutions or best-obtained results through all methods is emphasized by bold typeface and underlined.

3.1. Results of GA

The execution is performed on a Pentium III/600MHz PC, with 20 independent runs per problem instance. Number of function evaluations is approximately equal to (5).

$$(N_{pop} - N_{elite}) * N_{gener} \approx 100\,000 * n$$
(5)

Because all other methods presented only best results in multiple runs, we also report results on that way.

Table 1. Results obtained by GA

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

3.1. Results obtained by other methods

Optimal values given by implicit enumeration technique ([4]) are presented in Table 2. As can be seen from it and are mentioned earlier, number of iterations (and running time) grows exponentially and this approach is able to solve only instances of small size (n ≤ 5).

Table 2. Implicit enumeration technique

n	Optimal value	Number of iterations	Avg. num. of subregions
2	**0.3852**	486	30
3	**0.2610**	624	29
4	**0.0560**	8321	388
5	**0.3371**	97496	3768

The results of Multi-Level Tabu Search is given in Table 3. For n ≤ 5 are applied two-level strategy, while for n ≥ 6 are used three levels. In third column are given results of improved approach of MLTS in combination with local search for improving the terminal point. Other parameters for the MLTS is described briefly described in [2, 5].

Table 3. Multi-Level Tabu Search (MLTS)

n	MLTS best sol.	MLTS+LS best sol.	TS iterations	Func. eval.
2	**0.3852**	-	7581	151 620
3	**0.2610**	-	5280	132 000
4	0.0599	-	2839	85 170
5	0.3418	-	3123	109 305
6	0.4603	**0.4588**	5307	212 280
7	0.4985	**0.4976**	1919	86 355
8	0.4288	**0.3871**	7385	369 250
9	0.3820	0.3492	7297	401 335
10	0.4615	**0.4342**	7175	430 500

Table 4. contains results obtained by standard Monte Carlo method with 10 million points, presented in [2].

Table 4. Monte Carlo method

n	Best solution
2	0.3857
3	0.2687
4	0.0823
5	0.3988
6	0.5000
7	0.6772
8	0.6179
9	0.8802
10	0.9106

SSRPCD problem is also solved by nonlinear programming method and results given in [3] are presented in Table 5.

Table 5. Nonlinear programming

n	Best solution
3	0.261447
4	0.057048
5	0.340255
6	0.590729
7	0.517258
8	0.488383
9	0.350310
10	0.589604

3.3. Result comparison

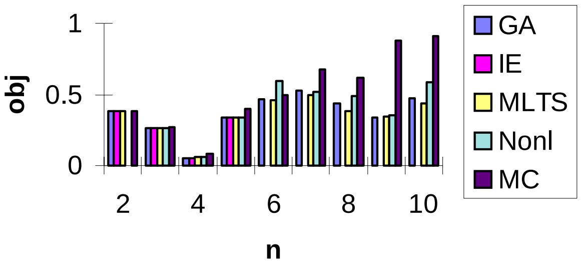
As can be seen from Tables 1-5 and Fig. 6. Monte Carlo method is worst for all instances (never achieved optimal solution (Opt.) or best obtained result (BS)) although it perform the most number of function evaluation.

Nonlinear programming method also never achieved optimal solution or best obtained result, but it is quite better than MC and not too far from Opt (BS).

GA and MLTS both achieve optimal solution for *n* = 2 and *n* = 3. Result of GA is still optimal for *n* = 4.

For n ≥ 6 optimal solution is not known, and MLTS in average produce better results (BS for *n* = 6, 7, 8 and 10) than GA (BS for *n* = 9), but differences are not very large.

Results of other approaches in [1-5] are not reported for n ≥ 11 and it is not possible to compare them with GA.



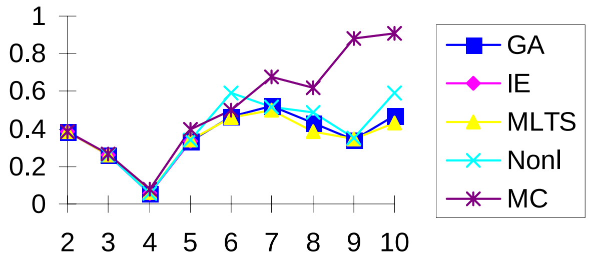


Fig. 6. Result comparison by objective value

Errors from optimal solution (BS) for all methods except implicit enumeration (it has not produce any errors in the cases when finishing its work) and Monte Carlo (has very large error) is displayed in Fig. 7.

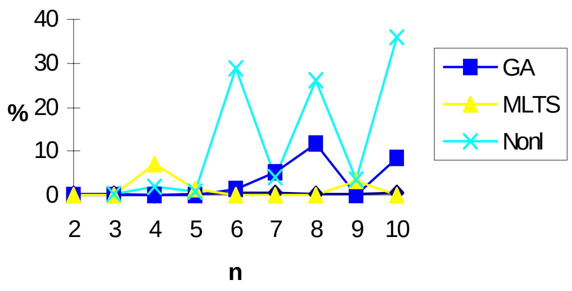


Fig. 7. Errors from Opt. (BS)

4. CONCLUSION

In this paper is proposed a new GA-based approach for designing a spread spectrum radar polyphase code. Using the representation by Gray codes, rank-based selection and steady-state generation's replacement with elitist strategy, the convergence of GA is significantly improved. Running time is additionally improved by caching GA technique.

By GA implementation is solved SSRPCD instances up to n = 20. Computational results indicate that GA implementation is comparable to other methods for all instances. Moreover, in some cases is obtained better results compared to all presented methods.

The research presented in this paper can be extended by the hybridization of GA with other methods, possible parallelization and application to other similar global optimization problems.

ACKNOWLEDGEMENT

We thank Vera Kovačević-Vujičić, Mirjana Čangalović and Milan Dražić for giving us important information related to the spread-spectrum radar polyphase code design problem.

REFERENCES

[1] M.L. Dukić and Z.S. Dobrosavljević, "A method of a spread-spectrum radar polyphase code design", *IEEE Journal on Selected Areas in Communications,* vol. 5, pp. 743-749, 1990.

[2] V.V. Kovačević-Vujičić, M.M. Čangalović, M.D. Ašić, L. Ivanović and M Dražić, "Tabu search methodology in global optimization", *Computers and Mathematics with Applications*, vol. 37, pp. 125-133, 1999.

[3] M.L. Dukić and Z.S. Dobrosavljević, "A method of spread spectrum radar polyphase code design by nonlinear programming", *European Transactions on Telecommunications and Related Techniques* (to appear).

[4] M.D. Ašić and V.V. Kovačević-Vujičić, "An implicit enumeration method for global optimization problems", *Computers and Mathematics with Applications*, vol. 21, no. 6/7, pp. 191-201, 1991.

[5] M.M. Čangalović, V.V. Kovačević-Vujičić, L. Ivanović, M Dražić and M.D. Ašić, "Tabu search: A brief survey and some real-life applications", *Yugoslav Journal of Operations Research*, vol. 6, no. 1, pp. 5-18, 1996.

[6] J.H. Holland, *Adaptation in natural and artificial systems*, Ann Arbor., USA, The University of Michigan Press, 1975.

[7] D.E. Goldberg, *Genetic algorithms in search, optimization and machine learning*, Reading, Mass., USA, Addison-Weseley Publ. Comp., 1989.

[8] D. Beasley, D.R. Bull and R.R. Martin, "An overview of genetic algorithms, part 1, fundamentals", *University Computing*, vol. 15, no. 2, pp. 58-69, 1993.

[9] D. Beasley, D.R. Bull and R.R. Martin, "An overview of genetic algorithms, part 2, research topics", *University Computing*, vol. 15, no. 4, pp. 170-181, 1993.

[10] M. Srinivas and L.M. Patnaik, "Genetic algorithms: A survey", *IEEE Computer*, pp. 17-26., June 1994.

[11] K.E. De Jong and W.M. Spears, "Using genetic algorithms to solve NP-complete problems", in *Proceedings of the Third International Conference on Genetic Algorithms*, Morgan Kaufmann, San Mateo, CA, USA, pp. 124-132, 1989

[12] S. Khuri, T. Back and J. Heitkotter, "An evolutionary approach to combinatorial optimization problems, in *Proceedings of CSC'94*, Phoenix, AR, USA, 1994.

[13] Z. Michalewicz, *Genetic algorithms + Data structures = Evolution programs*, Third Edition, Berlin - Heidelberg, Springer-Verlag, 1995.

[14] J.T. Alander, "Indexed bibliography of genetic algorithms in operations research", Report 94-1-OR, Department of Information Technology and Production Economics, University of Vassa, Finland, 1995. Available from <ftp://ftp.uwasa.fi/cs/report94-1/gaORbib.ps.Z>

[15] I.H. Osman and G. Laporte, "Metaheuristic: A bibliography", *Annals of Operations Research*, vol. 63, pp. 513-623, 1996.

[16] D. Whitley, "The GENITOR algorithm and selection pressure: Why rank-based allocation of reproductive trials is best", in *Proceedings of the Third International Conference on Genetic Algorithms*, San Mateo, CA, USA, Morgan Kaufmann, pp. 116-123, 1989.

[17] J. Kratica, "Parallelization of genetic algorithms for solving some NP-complete problems", PhD thesis, Faculty of Mathematics, Belgrade, 2000. (in Serbian).

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

THE AUTHORS

Jozef Kratica was born in 1966 in Belgrade, Serbia, Yugoslavia. He received his B.S. 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 2000, he joined Mathematical Institute as a researcher. As a delegation leader participated on 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 was born in Knjaževac, Serbia, Yugoslavia (1949). He received his B.S. degree in mathematics (1972), M.Sc. in mathematics (1978) and Ph.D. in mathematics (1984) from University of Belgrade, Faculty of Mathematics. Since 1985 he has been professor of computer science at Faculty of Mathematics His research interests include parallel algorithms, optimization, and evolutionary computation.

Vladimir Filipović was born in Podgorica, Montenegro, Yugoslavia (1968). He received his B.S. degree in computer science (1993) and M.Sc. in computer science (1998) from University of Belgrade, Faculty of Mathematics. Since 1994 is teaching assistant at the Faculty of Mathematics. Since 1994 is teaching assistant at the Faculty of Mathematics. Also working as a software consultant for Analytx, Inc. USA. His research interests include genetic algorithms, parallel algorithms and operational research.

Ivana Ljubić was born in 1973 in Prokuplje, Serbia, Yugoslavia. She received his B.S. degree in computer science (1996) and M.Sc. in mathematics (2000) from University of Belgrade, Faculty of Mathematics. She currently working on Ph.D. studies at Vienna University of Technology, Institute for Computer Graphics. Her research interests include evolutionary computation and biconnectivity augmentation problems.