ELEKTROTEHNIŠKI VESTNIK 78(5): 275-280, 2011
ENGLISH EDITION

On Selection in Differential Evolution

Iztok Fajfar, Janez Puhan, Sašo Tomažič, and Árpád Bűrmen

University of Ljubljana, Faculty of Electrical Engineering, Tržaška 25, 1000 Ljubljana, Slovenia
E-mail: iztok.fajfar@fe.uni-lj.si

Abstract. Differential evolution is a simple algorithm for global optimization. Basically it consists of three operations: mutation, crossover and selection. Despite many research papers dealing with the first two, hardly any attention has been paid to the third one nor is there a place for this operation in the algorithm basic naming scheme. In the paper we show that employing different selection strategies combined with some random perturbation of population vectors notably improves performance in high-dimensional problems.

Keywords: global optimization, direct search methods, differential evolution, heuristic

1 INTRODUCTION

Differential Evolution (DE) is a simple yet powerful algorithm for global real parameter optimization proposed by Storn and Price [1]. Through the last decade, the algorithm has gained on popularity among research as well as engineering circles due to its extreme implementation simplicity and good convergence properties. The DE algorithm belongs to a broader class of Evolutionary Algorithms (EA), whose behavior mimics that of the biological processes of genetic inheritance and survival of the fittest. One outstanding advantage of EAs over other sorts of numerical optimization methods is that the objective function needs to be neither differentiable nor continuous, which makes them more flexible for a wide variety of problems.

A DE starts out with a generation of *NP* randomly generated *D*-dimensional parameter vectors. New parameter vectors are then generated by adding a weighted difference of two population vectors to a third vector. This operation is called mutation. One then mixes the mutated vector parameters with the parameters of another vector, called the target vector, to obtain the so-called trial vector. The operation of parameter mixing is usually called crossover in the EA community. Finally, the trial vector is compared to the target vector, and if it yields a better solution, it replaces the target vector. This last operation is referred to as selection. In each generation, each population vector is selected once as the target vector.

There exist several variants of the DE algorithm [2], of which the most commonly used is *DE/rand/1/bin*, which we explore in this paper. Before using the algorithm, one has to decide upon the values of three parameters affecting the behavior of a DE. The first is the population size *NP*, the other two are control parameters – a scaling factor *F*, and a crossover rate *CR*. Choosing the values of these parameters is usually a problem dependent task, which requires certain user expertise. Researchers have attempted to tackle the problem using several adapting and self-adapting strategies to govern the values of the control parameters *F* and *CR* [3, 4, 5] and even the population size *NP* [6, 7]. Others have proposed and studied different mutation and crossover strategies [8, 9, 10]. No explicit research work has been done so far on the third of the DE operators, the selection, neither is there any intended place in the algorithm variant naming scheme (i.e. DE/x/y/z) for this operator. In this paper we investigate how different selection schemes affect the behavior of the DE algorithm, in particular its ability to escape the local minima or stagnation. In addition to that we applied what would in genetic algorithm be called mutation, i.e. we randomly changed the population vector parameters with a fixed probability. Since the term mutation is already reserved in DE, we named this operation a random perturbation.

In the next section, we shortly describe the functioning of the basic DE algorithm, in Section 3 we propose a random vector perturbation and different selection schemes that we investigate and, in Section 4, we present some results on test functions.

2 A SHORT OVERVIEW OF DIFFERENTIAL EVOLUTION

Consider the objective (criterion) or *fitness* function $f: \mathbb{R}^n \rightarrow \mathbb{R}$, where one has to find a minimum $\vec{a} \in \mathbb{R}^n$ so that $\forall \vec{b} \in \mathbb{R}^n: f(\vec{a}) \leq f(\vec{b})$. In this case $\vec{a}$ is called a global minimum. It is rarely possible to find an exact global minimum in real problems, so for practical reasons one must accept a candidate with a reasonable good solution.

Received November 25, 2011
Accepted December 8, 2011

In order to search for a global minimum, differential evolution utilizes *NP* *D*-dimensional parameter vectors $x_{i,G}$, i=1,2,...,*NP* as a population in generation *G*. *NP* does not change from generation to generation. The initial population is chosen randomly and – if no prior information about the fitness function is known – it should cover the entire search space uniformly.

During the optimization process, the new parameter vectors are generated by adding a weighted difference of two randomly chosen population vectors to a third vector: $v_{i,G+1}=x_{r1,G}+F\cdot(x_{r2,G}-x_{r3,G})$ with integer, mutually different, random indices $r_1,r_2,r_3\in\{1,2,...,NP\}$, which must all be different from i as well, and a real constant factor $F\in[0,2]$. This operation is called *mutation*, and the thus obtained vector the *mutated* vector.

The mutated vector parameters are then mixed with another vector, the so-called *target* vector, in order to produce a *trial* vector $u_{i,G+1}=(u_{1i,G+1},u_{2i,G+1},...,u_{Di,G+1})$ where

$$u_{ji,G+1}=\begin{cases} v_{ji,G+1} & \text{if}(randb(j)\le CR) \text{ or } j=rnbr(i) \\ x_{ji,G} & \text{if}(randb(j)>CR) \text{ and } j\ne rnbr(i)' \end{cases}$$

$$j=1,2,\dots,D. \quad (1)$$

Here, *randb*(*j*) is the *j*th execution of the uniform random generator with output ∈ [0,1], *CR* is user-determined constant ∈ [0,1], and *rnbr*(*i*) is a random index ∈ {1,2,...,*D*}. The latter insures that the trial vector gets at least one parameter from the mutated vector. This operation of parameter mixing is usually called *crossover*.

Finally, a *selection* is performed in order to decide whether or not the trial vector should become a member of generation *G*+1. The value of the fitness function at the trial vector $u_{i,G+1}$ is compared to its value at the target vector $x_{i,G}$ using the greedy criterion. Only if the trial vector yields a better fitness value than the target vector, the target vector is replaced. Otherwise the trial vector is discarded and the target vector retained.

3 MODIFICATIONS TO THE ORIGINAL DE ALGORITHM

We focus our work on the stage of the DE algorithm after crossover, i.e. on the stage when the trial vector is already fully formed.

The idea for our modification came first from a simple observation that with a crossover rate *CR* approaching 1 not much of the target vector survives in its offspring (trial vector). In that sense one can argue that the search direction from the target to the trial vector can be as good (or as bad) as any other direction. The hypothesis we want to test is that there might exist some other (possibly better) candidate for replacement than the target vector itself.

In what follows, we propose and separately test three different rules for selecting the candidate to compete with the trial vector. We denote that candidate $c_{i,G}$ and select it according to one of the three selection criteria:

$$\begin{aligned} &\text{Cr1: } c_{i,G}=x_{n,G}, \exists n: \min_n\left(d(x_{n,G},x_{i,G})\right)\ \forall n: f(u_{i,G+1})<f(x_{n,G}) \\ &\text{Cr2 :} c_{i,G}=x_{n,G}, \exists n: \min_n\left(d(x_{n,G},u_{i,G+1})\right)\ \forall n: f(u_{i,G+1})<f(x_{n,G}) \\ &\text{Cr3: } c_{i,G}=x_{n,G}, \exists n: \begin{cases} n=i, f(u_{i,G+1})<f(x_{i,G}) \\ \text{smallest } n\in\left\{1,2,\dots\frac{NP}{2}\right\}: f(u_{i,G+1})<f(x_{n,G}), \text{otherwise} \end{cases} \end{aligned} \quad (2)$$

where $d(\cdot,\cdot)$ denotes an Euclidean distance. Note that there still exist cases where no appropriate candidate is selected in which case the trial vector is discarded.

Under criterion Cr1 one replaces, of all the vectors that yield a worse fitness value than the trial vector, the one that is geometrically closest to the target vector. Note that this strategy, the same as the original algorithm, always replaces the target vector as long as it is worse than the trial vector. Otherwise, it seeks after the candidate which is closest possible to the target vector to replace it. As in the original algorithm, the target vectors with a relatively bad fitness value will be replaced more likely, while those with a better fitness value will survive. In addition to that, however, some near vector is moved to the place where the target vector would move were it not worse than the trial vector. This speeds up the clustering of the population members around the members with generally better fitness values. On one hand this can accelerate the convergence significantly, on the other hand, however, there is a danger of losing a necessary diversity too soon and thus not finding a global solution.

The approach with the criterion Cr2 is quite different in that it searches for the candidate that is geometrically closest to the trial vector. In that sense replacements are made that favor smaller jumps and encourage searching over less promising areas as well.

The construction of the criterion Cr3 is not so obvious at the first glance. Similarly to the original algorithm and Cr1, we first check whether the target vector is to be replaced, i.e. if the trial vector yields a better fitness value than the target vector. Otherwise we replace the first member of the first half of the population whose fitness value is worse than that of the trial vector. The idea behind that is to have a half of the population evolve under the original DE rules while

accelerating the other half with further replacements. Even these additional replacements are applied asymmetrically with the members with a smaller index affected more often. That way we wanted to induce as little a change to the original method as possible, while inducing a relatively strong drag on a limited number of population members.

Before going into experiments, let us introduce one more tiny modification to the algorithm. It is interesting to note that although the algorithm itself belongs to a class of metaheuristics and stochastic optimization, the randomness in the original concept is only used for the selection of the vectors from which the mutated vector will be formed and for mixing the mutated and target vector parameters. The vector parameters themselves are changing randomly only indirectly through the mutation and crossover, and the obtained values are limited to a set of linear combinations of parameters already contained in a population. Some authors have already introduced some more randomness into DE, either directly by randomization of the vector parameters [11, 12] or indirectly by randomizing the algorithm control parameters *F* and *CR* [13, 14].

In our study we decided simply to mutate every single parameter of the trial vector with a fixed probability just before the selection procedure takes place:

$$u_{ji,G+1} = \begin{cases} rand(j), \text{if}(randb(j) \le 0.05) \\ u_{ji,G+1}, \text{otherwise} \end{cases},$$

$$j = 1,2,\dots,D, \tag{3}$$

where *rand*(*j*) is the call of the random generator that returns the uniformly distributed values along the entire *j*th axis of the parameter space. The constant probability of 0.05 was obtained empirically by a few preliminary test runs of the algorithm, which also indicated that the uniform distribution over the whole parameter space yielded somewhat superior performance compared to a normal distribution around the current parameter value often used in literature. We call this operation *perturbation*.

4 RESULTS

4.1 Overall Performance

In order to get an overall picture and the first impression of the impact of the three proposed selection strategies and random vector perturbations, we carried out a simple test. For testing purposes, fourteen standard benchmark functions from [15] were selected, thirteen high-dimensional (D=30) and one low-dimensional (D=4) function. Then we randomly selected the three parameters from the intervals $NP \in \{10,\dots,100\}$, $F \in [0,1]$, and $CR \in [0,1]$, and initialized a random population of the *NP* parameter vectors lying within the boundaries given for the test function in question. Next we executed eight optimization runs of the 150,000 criterion function evaluations (CFEs) with the same parameter values and initial vector population, but each time applying either the original or one of the three proposed selection schemes, once without and once with a random perturbation. We repeated this 5,000 times for each test function, each time with different control parameter values and initial vector population. The results are summarized in Table 1.

Table 1: Comparison of Different Modifications of the Algorithm with the Original

Selection method	Without Perturbation		With Perturbation	
	50,000 CFEs	***150,000 CFEs***	***50,000 CFEs***	***150,000 CFEs***
Original	–	–	44.1/51.7	43.7/44.1
Cr1	53.5/43.4	45.1/48.0	69.9/26.9	63.1/29.0
Cr2	52.8/44.1	45.7/47.4	62.4/34.4	57.3/35.4
Cr3	61.4/34.6	53.0/37.1	75.2/20.6	68.5/20.5

The fourteen pairs of the numbers in the table stand for the seven different comparisons (each of the modifications separately compared to the original) at two different times of the algorithm run: after 50,000 CFEs and after 150,000 CFEs. The numerator represents the percentage of cases in which the corresponding modification yielded a better fitness value (at the precision of 6 significant digits) than the original, while the denominator speaks of the percentage of cases in which the original method performed better. The sum is generally smaller than 100, because in some cases both variants gave the same result. The counting was carried out over all runs regardless of the control parameter setting or the selected test function. In real-life problems, often the practitioner has little or no knowledge about the fitness function and consequently about the best control parameter settings. Therefore, it seems that averaging over a range of different test functions and control parameter settings, selected in the Monte-Carlo manner is an appropriate measure of the algorithm overall performance.

In the table, the pairs of the numbers in the white cells represent the state after 50,000 CFEs. We conjectured that at that stage of optimization the convergence is generally not yet fully reached. Consequently those pairs of the numbers hint at the convergence speed rather than at the overall ability to find a global minimum.

The numbers in the shaded cells represent the state after 150,000 CFEs, when we assume that the number of the cases reaching the final solution is considerably larger than of those after 50,000 CFEs. Hence we consider these results to reflect the ability of the algorithm to find a good final solution.

From the table one can infer some quite interesting observations. Replacing – instead of target vector – the

candidate closest to target (Cr1) or closest to trial vector (Cr2) without using perturbation performed just slightly better after 50,000 CFEs (1st column, 2nd and 3rd row, respectively) and slightly worse after 150,000 CFEs (2nd column, 2nd and 3rd row, respectively). That implies that the more frequent replacements in both cases speed up the convergence as expected but, in general, they more often stuck in local minima or reach stagnation in the end. That is, however, not the case with the selection strategy using Cr3. This strategy outperformed the original for almost twice more cases after 50,000 CFEs and still remained much better after 150,000 CFEs (4th row, 1st and 2nd column, respectively). It is important to note that with this kind of modification the algorithm still performs more replacements than the original one, which obviously speeds-up the convergence. The main difference here is that we perform these additional replacements only on a limited number of the population members, the others still undergoing the original selection scheme. Technically, we can speak of two different schemes running in parallel.

Comparing the original method with and without perturbations gives us no noticeable difference (1st row, 3rd and 4th column). As reasonably expected, random perturbations slow down convergence to some extent (1st row, 3rd column), but in the long run no variant outperforms the other (1st row, 4th column). It is quite interesting to note that while perturbation seems to have no observable effect when applied to the original algorithm, it improves the other three variants noticeably. It seems that in these cases the perturbation not only makes up for the loss of the population variance – which might have occurred due a to too fast convergence induced by more frequent replacements – but also improves the overall performance. It seems that the changed selection schemes and random perturbations support each other. Nevertheless, comparing the results of the selection criteria Cr1, Cr2, and Cr3 with perturbation after 50,000 and 150,000 CFEs shows that in all the three cases, in the long run, the original method compensates a little for the much worse performance during the first part of the run. This leaves us, possibly, some room for improvement by balancing the factors that affect the convergence speed and the rate of change in the population diversity.

4.2 A Closer Look

Let us now focus a little closer on the selection criterion Cr3 combined with vector perturbation exhibiting the best overall improvement in the previous analysis. In order to get a more accurate picture, we made the same comparisons as before, only this time for each test function separately. The results are summarized in Table 2. The table shows comparisons of the original method with the original method with perturbation, and with the selection criterion Cr3 with and without perturbation. The numbers in normal writing represent the state after 50,000 CFEs, while the ones in boldface the state after 150,000 CFEs.

Table 2: Comparison of Different Modifications by Separate Test Functions

Test Function	**Modification Compared with the Original Algorithm**		
	Original with Perturbation	***Cr3***	***Cr3 with Perturbation***
f_1 (Quadratic)	34.72/65.28 **31.60/68.40**	70.83/29.17 **68.75/31.25**	80.21/19.79 **76.39/23.61**
f_2 (Schwefel 2.22)	33.45/66.55 **31.71/68.29**	70.73/29.27 **66.90/33.10**	74.22/25.78 **70.03/29.97**
f_3 (Schwefel 1.2)	51.04/48.26 **54.51/45.49**	75.35/23.96 **70.49/29.51**	77.08/22.57 **78.47/21.53**
f_4 (Schwefel 2.21)	49.48/48.08 **48.43/49.83**	63.07/34.49 **59.58/39.72**	83.97/12.80 **79.79/18.82**
f_5 (Generalized Rosenbrock)	49.83/50.17 **52.96/47.04**	58.19/41.81 **56.10/43.90**	77.00/23.00 **73.87/26.13**
f_6 (Step)	31.36/24.39 **29.97/9.76**	31.36/27.87 **18.12/26.48**	47.04/6.62 **35.19/3.48**
f_7 (Quartic noisy)	46.50/53.50 **49.30/50.70**	70.28/29.72 **67.83/32.17**	77.97/22.03 **77.62/22.38**
f_8 (Generalized Schwefel 2.26)	57.14/42.86 **58.54/29.62**	49.83/50.17 **36.59/58.19**	82.58/17.42 **78.75/11.15**
f_9 (Generalized Rastrigin)	50.69/48.96 **49.65/43.40**	66.67/33.33 **57.64/39.24**	80.90/19.10 **79.17/15.28**
f_{10} (Ackley)	48.26/50.69 **48.96/33.33**	60.07/39.24 **43.75/41.67**	81.60/17.36 **68.40/15.63**
f_{11} (Generalized Griewank)	32.17/61.19 **31.47/36.71**	63.99/29.72 **42.31/29.02**	73.08/20.28 **53.50/17.83**
f_{12} (Generalized penalty function 1)	43.36/56.29 **36.01/45.80**	68.53/31.12 **52.45/33.22**	81.47/18.53 **65.38/20.63**
f_{13} (Generalized penalty function 2)	45.10/54.90 **42.66/43.01**	62.59/37.06 **50.35/37.06**	82.17/17.48 **72.03/15.03**
f_{15} (Kowalik)	44.41/52.45 **45.80/46.50**	48.25/47.55 **50.70/45.10**	52.80/45.80 **49.65/45.80**

The first and foremost important observation here is that the modification combined with perturbation shows a noticeable and consistently better performance in all cases except for the Kowalik test function where there is no observable difference. Again we see that perturbation alone does not really improve performance of the original method, two notable exceptions being the Schwefel 2.26 and Step functions.

The Schwefel function is somehow tricky in that the global minimum is placed geometrically remote from the next few best local minima. The original method exhibits quite a good convergence at the beginning, while later on perturbations help find a global minimum as without them the original method would be stuck in a local minimum (see the 1st column, Schwefel 2.26 function). Interestingly enough, modification without perturbation in that case performs much worse than the original method. This probably stems from the fact that this method replaces candidates of one half of the population excessively, thus additionally forcing the population in one of the local minima. The modified method with perturbation, however, performs much better in this case.

The same goes for the step function. This function too, poses some difficulties for the original algorithm because it consists of many plateaus and discontinuities. All points within a small neighborhood will have the same fitness value, making it very difficult for the process to move from one plateau to another. Perturbations seem to help here significantly.

4.3 Parameter Impact

In our experiments so far we didn't pay any attention to the actual control-parameter or population-size selection. The values were picked up completely randomly within the set intervals. In this section we want to investigate the effect of different parameter settings on the algorithm performance with the proposed modifications. We compare the original algorithm to the one using the selection scheme Cr3 with perturbation. Although the focus here is on a single test function (Generalized Schwefel 2.26) we should note that a similar behavior was observed also elsewhere.

We started by choosing the control parameter settings most commonly found in literature, i.e. $F = 0.5$ and $CR = 0.9$. Our experimenting showed that at these values the best fitness (assuming a fixed number of 150,000 CFEs) is generally obtained at the population size $NP = 40$. The results in this section are obtained by changing one of the three values while keeping the other two fixed. The best fitness values were averaged over 25 independent runs.

Fig. 1 shows that the original method completely failed to reach the global minimum (at –12569.5) safe for the lowest values of *CR*. Interesting, however, is that the modification enables DE to find the global minimum at lower and higher values of *CR*, but not at the values around 0.7. A similar behavior can be observed in Fig. 2, only that here the improvement is somehow worse only at the highest values of *F*.

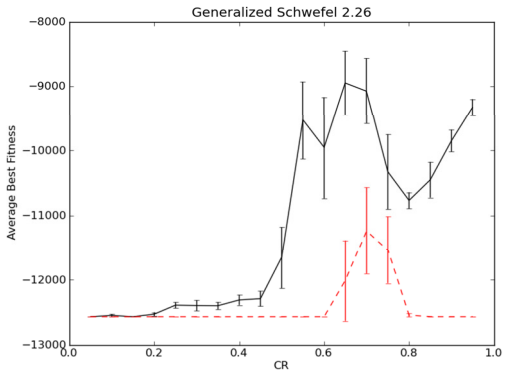

Figure 1: Impact of the *CR* control parameter. The solid line represents the results using the original algorithm while the dashed line depicts the values obtained using the Cr3 selection scheme with perturbation.

In Fig. 3, which depicts the inpact of the population size, we can see that the major improvement is achieved at lower population sizes. The large population size in DE usually guarantees the larger probability of finding a global minimum, and originally, the proposed population size was $NP = 10D$ [16]. Other sizes were proposed later but also all considerably greater than the fitness function dimensionality *D*. As seen from Fig. 3, at larger population sizes our modification does not bring any improvement over the original method whatsoever. That is somehow expected since the DE should be quite stable at larger *NP*. The problem however is that the stability is of no great practical use if after the relatively large number of CFEs the algorithm is still very far from the actual solution. We see one of the strongest values of our modification in having instead of one large population many smaller ones running in parallel which could bring together the ability to actually find the global minimum and speed up of convergence.

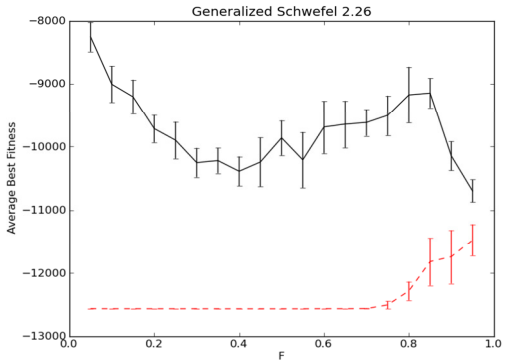

Figure 2: Impact of the *F* control parameter.

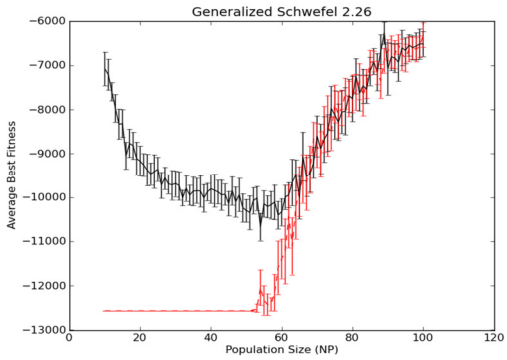

Figure 3: Impact of the population size.

5 CONCLUSION

In the paper we studied different replacement schemes in the DE algorithm combined with the additional random perturbation of vector parameters. By experimenting with a suite of standard test functions we observed that only one replacement scheme provided observably better results than the original algorithm. It

was somehow surprising to observe that perturbation doid not improve behavior of the original replacement scheme while it improved all the others.

Studying the performance of the replacement scheme Cr3 combined with random perturbation showed quite considerable improvement in all higher dimensional test functions. We also saw that the improvement is greater at certain values of the control parameters and population sizes, i.e. at lower values of *F* and *NP*, and at lower as well as higher values of *CR*. Especially outstanding was the improvement in smaller population sizes which could be useful in implementing parallel DE algorithms using a number of smaller populations.

One of the advantages of the approach proposed in this paper is the fact that its intervention with the original method does not interfere with any other operation and can therefore be applied independently and combined with many other approaches proposed in literature.

All in all, the beauty of the original DE algorithm is its utmost implementation simplicity. Our research tried not to stray away from this simplicity and we showed that it is possible to improve the algorithm performance by only changing the rule for replacing the population members combined with simple random perturbation. We believe that further work in this direction is worthwhile.

ACKNOWLEDGEMENT

The research has been supported by the Ministry of Higher Education, Science, and Technology of Republic of Slovenia within the research program P2-0246 – Algorithms and optimisation methods in telecommunications.

REFERENCES

[1] R. Storn and K. Price, "Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces," J. Glob. Optim., vol. 11, pp. 341–359, 1997.

[2] K. Price, "An introduction to differential evolution," in New ideas in optimization, D. Corne, M. Dorigo, and F. Glover, Eds. London (UK): McGraw-Hill Ltd., 1999, pp. 79–108.

[3] J. Brest, S. Greiner, B. Bošković and M. Mernik, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., vol. 10, pp. 646–657, 2006.

[4] J. Liu and J. Lampinen, "A fuzzy differential evolution algorithm," Soft Comput., vol. 9, pp. 448–462, 2005.

[5] D. Zaharie, "Control of population diversity and adaptation in differential evolution algorithms," in Proceedings of 9th International Conference on Soft Computing, R. Matoušek, P. Ošmera, Eds. Brno (Czech Republic): Mendel 2003, pp. 41–46.

[6] J. Brest and M. Sepesy Maučec, "Population size reduction for the differential evolution algorithm," Appl. Intell., vol. 29, pp. 228–247, 2008.

[7] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," Soft Comput., vol. 10, pp. 673–686, 2006.

[8] H. Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," J. Glob. Optim., vol. 27, pp. 105–129, 2003.

[9] S. Das, A. Abraham, U. K. Chakraborty, and A. Konar, "Differential evolution using a neighborhood-based mutation operator," IEEE Trans. Evol. Comput., vol. 13, pp. 526–553, 2009.

[10] D. Zaharie, "Influence of crossover on the behavior of differential evolution algorithms," Appl. Soft Comput., vol. 9, pp. 1126–1138, 2009.

[11] Z. Yang, J. He, and X. Yao, "Making a difference to differential evolution," in Advances in metaheuristics for hard optimization, Z. Michalewicz and P. Siarry, Eds.: Springer, 2007, pp. 415–432.

[12] J. Olenšek, Á. Bűrmen, J. Puhan, and T. Tuma, "DESA: a new hybrid global optimization method and its application to analog integrated circuit sizing," J. Glob. Optim., vol. 44, pp. 53–77, 2009.

[13] J. Brest, S. Greiner, B. Bošković, M. Mernik, and V. Žumer, "Self-adapting control parameters in differential evolution: a comparative study on numerical benchmark problems," IEEE Trans. Evol. Comput., vol. 10, pp. 646–657, 2006.

[14] S. Das, A. Konar, and U. K. Chakraborty, "Two improved differential evolution schemes for faster global search," in Proceedings of GECCO, Washington D.C.: 2005, pp. 991–998.

[15] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," IEEE Trans. Evol. Comput., vol. 3, pp. 82–102, 1999.

[16] R. Storn, "On the usage of differential evolution for function optimization," in Biennial Conference of the North American Fuzzy Information Processing Society (NAFIPS), Berkeley: 1996, pp. 519–523.

Iztok Fajfar received his B.Sc., M.Sc., and Ph.D. degrees in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia in 1991, 1994, and 1997, respectively. In 1991 he was researcher at the Jozef Stefan Institute in Ljubljana. At the end of the same year he was granted a research position at the Faculty of Electrical Engineering, University of Ljubljana. Currently he holds the position of an associate professor at the same faculty. He teaches several introductory and advanced courses in computer programming. He has also participated in several industrial software projects. His research interests include design and optimisation of electronic circuits.

Janez Puhan received his Ph.D degree in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia, in 2000. He is an assistant professor at same faculty. His research interests include modelling, simulation and optimization techniques in computer-aided circuit design.

Sašo Tomažič is a professor at the Faculty of Electrical Engineering, University of Ljubljana, Slovenia. He is the Head of the Laboratory of Communication Devices and the chair of the Telecommunication Department. His work includes research in the field of signal processing, security in telecommunications, electronic commerce and information systems.

Árpád Bűrmen received his Ph.D. degree in electrical engineering from the Faculty of Electrical Engineering, University of Ljubljana, Slovenia, in 2003. Currently, he is an associate professor at the same Faculty. His research interests include continuous and event-driven simulation of circuits and systems, optimization methods and their convergence theory and applications, and algorithms for parallel and distributed computation. He is one of the principal developers of the SPICE OPUS circuit simulator and has published over 20 papers in peer-reviewed journals.