ELSEVIER

Advances in Engineering Software 35 (2004) 237–246

ADVANCES IN ENGINEERING SOFTWARE

www.elsevier.com/locate/advengsoft

Improvements of real coded genetic algorithms based on differential operators preventing premature convergence

Ondřej Hrstka[a], Anna Kučerová[b,*]

[a]*Computing and Information Center, Faculty of Civil Engineering, Czech Technical University in Prague, Thákurova 7, 166 29 Prague, Czech Republic*
[b]*Department of Structural Mechanics, Faculty of Civil Engineering, Czech Technical University in Prague, Thákurova 7, 166 29 Prague, Czech Republic*

Received 17 June 2002; accepted 22 July 2003

Abstract

This paper presents several types of evolutionary algorithms used for global optimization on real domains. The interest has been focused on multimodal problems, where the difficulties of a premature convergence usually occur. First the standard genetic algorithm using binary encoding of real values and its unsatisfactory behavior with multimodal problems is briefly reviewed together with some improvements of fighting premature convergence. Two types of real encoded methods based on differential operators are examined in detail: the differential evolution (DE), a very modern and effective method first published by Storn and Price [NAPHIS, 1996], and the simplified real-coded differential genetic algorithm SADE proposed by the authors [Contributions to mechanics of materials and structures, 2000]. In addition, an improvement of the SADE method, called CERAF technology, enabling the population of solutions to escape from local extremes, is examined. All methods are tested on an identical set of objective functions and a systematic comparison based on a reliable methodology [Adv. Engng Software 32 (2000) 49] is presented. It is confirmed that real coded methods generally exhibit better behavior on real domains than the binary algorithms, even when extended by several improvements. Furthermore, the positive influence of the differential operators due to their possibility of self-adaptation is demonstrated. From the reliability point of view, it seems that the real encoded differential algorithm, improved by the technology described in this paper, is a universal and reliable method capable of solving all proposed test problems.

Keywords: Genetic algorithm; Binary algorithm; Reliability

1. Introduction

At present, genetic algorithms belong to the most modern and most popular optimization methods. They follow an analogy of processes that occur in living nature within the evolution of live organisms during a period of many millions of years. The principles of genetic algorithms were firstly proposed by Holland [9]; the books of Goldberg [7] and Michalewicz [14] are the most popular publications that deal with this topic. Genetic algorithms have been successfully used to solve optimization problems in combinatorics (see Ref. [8]) as well as in different engineering tasks, see for example Refs. [12,13,15].

Unlike the traditional gradient optimization methods, genetic algorithms operate on a set of possible solutions ('chromozomes'), called 'population'. In the basic scheme, chromozomes are represented as binary strings. This kind of representation seems to be very convenient for optimization problems in combinatoric area (e.g. the traveling salesman problem). Nevertheless, we usually deal with real valued parameters in engineering and scientific problems. The mapping of real values onto binary strings usually used within standard genetic algorithms (SGAs) may cause serious difficulties. As a result, this concept of optimization leads to an unsatisfactory behavior, characterized by a slow convergence and an insufficient precision, even in cases where the precision is especially in focus. Of course, the development of genetic algorithms has brought several proposals to solve these difficulties to optimize problems on real domains using binary algorithms.

Another possibility is to develop a genetic algorithm (or other evolutionary algorithm (EA)) that operates directly on real values [14]. In this case, the biggest problem is how to

* Corresponding author.
E-mail address: anicka@cml.fsv.cvut.cz (A. Kučerová).

0965-9978/$ - see front matter © 2003 Published by Elsevier Ltd.
doi:10.1016/S0965-9978(03)00113-3

propose genetic operators. One of them is to use so-called differential operators that are based on determining mutual distances of chromozomes—which are real vectors instead of binary strings.

This paper studies two evolutionary optimization methods based on differential operators with reference to the standard and improved genetic algorithms using binary encoding. In particular, the differential evolution (DE) proposed by Storn and Price [4,17] and the simplified real coded differential genetic algorithm [11,16] are examined in more detail.

Although the outstanding ability of genetic algorithms to find global optima of multimodal functions (functions which have several local extremes) is usually cited in the GA literature, it seems that both the binary genetic algorithms and the real coded ones tend to premature converge and to fall into local extremes, mainly in high dimensional cases. To fight this difficulty, we have proposed so-called CERAF method.

As the reference, test results for binary encoded algorithms from the outstanding paper of Andre et al. [1] were selected. These results come from two variants of binary GAs: the SGA and the version extended by several improvements that were documented in the same publication. The set of twenty test functions was used to classify reliability and performance of individual methods. In particular, the reliability is defined as a probability of finding the global extreme of a multimodal function while the performance is measured by the convergence rate of an optimization method. Since this methodology is able to filter out the influence of random circumstances we have used the same criteria to quantify the robustness and the efficiency of real encoded optimization methods.

2. Binary coded genetic algorithm

Although the present paper deals mainly with real encoded EAs, we present a brief description of binary genetic algorithms, their limitations and possible improvements for the sake of further reference.

2.1. Binary encoding

A binary genetic algorithms can be simply characterized by the binary encoding of possible solutions and appropriate binary genetic operators. The traditional binary genetic algorithms represent possible solutions as binary strings, usually derived from a division of the investigated interval into a several sub-intervals with a specified, usually rather limited, precision. The fact that different bits in the binary string have different importance depending on their position in the string is the serious problem of this type of encoding. This disadvantage can be resolved by several improvements, see Section 2.3 for particular examples and references to the literature.

2.2. Scheme of genetic algorithm and genetic operators

As the first step it is necessary to generate (in most cases randomly) the starting population of possible solutions that are assigned the values of the optimized (or so-called fitness) function. Then, the sequential loop is repeated until a stopping criterion is reached:

1. Create a prescribed number of new individuals (chromozomes) using genetic operators of crossing-over and mutation.
2. Values of fitness function are assigned to new individuals.
3. The population size is decreased to the original value using selection operator.

In the following, we present a sketchy description of basic genetic operators:

Mutation—the principle of this operator is an alteration of one or more bits in the binary string [6]; a parameter which gives a probability of performing this operation with a certain chromozome, is introduced.

Crossing-over—this operator chooses two chromozomes, so-called parents, and then creates their two descendants (children) using the following operation: it selects a position inside the binary string and starting from this position exchanges the remaining parts of the two chromozomes (see Fig. 1). The individuals subject to crossing-over are selected by an appropriate sampling method, which is not necessarily identical to the selection method employed in the next step.

Selection—this operation selects the individuals that should 'survive' into the next generation from the whole population. See, e.g. Refs. [2,7] for a comprehensive list of different variants of selection schemes.

2.3. Improvements of the standard binary genetic algorithm

A lot of improvements of the standard binary genetic algorithm that aim at suppressing the premature convergence have been proposed by different authors, starting from different encoding, e.g. the well-known *Gray code* [10]; proceeding with *threshold genetic algorithm* with varying mutation probability [5]. Other possibilities are various adaptations of the crossing-over and the mutation or an introduction of local, gradient-based operators such as *gradient optimizer* [3] or *evolutionary gradient operator* proposed in Ref. [18]. In particular, in the reference study

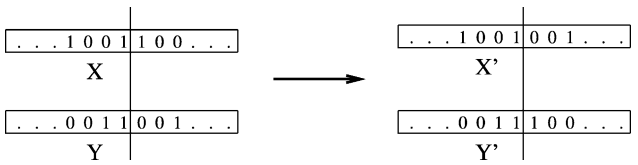

Fig. 1. Crossing-over operator.

[1] the authors performed the testing computations of the SGA extended by *adaptive rescaling* of the investigated area and by introducing the *scale-factor*. For the sake of clarity, a more detailed description of these improvements follows:

- *the adaptive rescaling* of investigated domain: the area where the method searches for the optimum is diminished into the regions around several best chromozomes; it makes possible to reach very good precision even if the division of investigated interval is rather rough because with the decreasing searched range the division becomes more and more refined,
- introducing so-called *scale-factor* which influences selection of individuals to be subject to the crossing-over; at the beginning the worse individuals gain higher probability and the better ones gain lower probability contrary to the SGA; with successive generations this parameters decreases and for the last generation the selection works in the same manner as for the standard version.

2.4. Genetic algorithm testing methodology

The methodology proposed in Ref. [1] minimizes an influence of random circumstances and different power of the used computers. In particular, the computation is run 100 times for each function of the test set. The number of successful runs is then taken as the probability of the success (the computation is considered to be successful if the difference between the best value found by the algorithm and the theoretical optimum is less than 1% of the optimum value, or a distance is less than 0.1 if the theoretical optimum is zero). If 500 generations pass and the optimum is still not reached, the computation is treated as a failure. For the cases where the amount of successful runs is greater than zero, the average fitness call number is also given. For the sake of completeness, we list a (corrected) set of twenty test functions in Appendix A while the results of the binary genetic algorithm testing are shown in Table 1.

3. Differential evolution

This section opens the main topic of the present work—a search for improvements of real coded genetic algorithms aimed at resolving the premature convergence. To this end, a thorough description of the DE, which is the stepping stone of our improvements, is presented.

The DE belongs to the wide group of EAs. It was invented as the solution method for the Chebychev trial polynomial problem by Storn and Price [17]. It is a very modern and efficient optimization method essentially relying on so-called differential operator, which works with real numbers in natural manner and fulfills the same purpose as the crossing-over operator in the SGA.

3.1. The differential operator

The typical differential operator has the sequential character: let $\mathrm{CH}_i(t)$ be the ith chromozome of a generation t

$$\mathrm{CH}_i(t) = (\mathrm{ch}_{i1}(t), \mathrm{ch}_{i2}(t), \ldots, \mathrm{ch}_{in}(t)), \tag{1}$$

where n is the chromozome length (which equals to the number of variables of the fitness function in the real encoded case). Next, let Λ be a subset of $\{1, 2, \ldots, n\}$.[1] Then, for each $j \in \Lambda$

$$\mathrm{ch}_{ij}(t+1) = \mathrm{ch}_{ij}(t) + F_1(\mathrm{ch}_{pj}(t) - \mathrm{ch}_{qj}(t)) + F_2(\mathrm{ch}_{\mathrm{best}j}(t) - \mathrm{ch}_{ij}(t)), \tag{2}$$

and for each $j \notin \Lambda$

$$\mathrm{ch}_{ij}(t+1) = \mathrm{ch}_{ij}(t), \tag{3}$$

where ch_{pj} and ch_{qj} are the jth coordinates of two randomly chosen chromozomes and $\mathrm{ch}_{\mathrm{best}j}$ is the jth coordinate of the best chromozome in generation t. F_1 and F_2 are random coefficients usually taken from the interval $(0, 1)$.

3.2. The differential evolution algorithmic scheme

The DE can be understood as a stand-alone evolutionary method or it can be taken as a special case of the genetic algorithm. The algorithmic scheme is similar to the genetic algorithms but it is much simpler:

1. At the beginning the initial population is created (e.g. randomly) and the fitness function value is assigned to each individual.
2. For each chromozome in the population, its possible replacement is created using the differential operator discussed above.
3. Each chromozome in the population has to be compared with its possible replacement and if an improvement occurs, it is replaced.
4. Steps 2 and 3 are repeated until a stopping criterion is reached.

As could be seen, there are certain different features in contrary to the SGA, namely:

- the crossing-over is performed by applying the differential operator,
- selection for crossing-over, e.g. by the roulette wheel method, is not performed; the individuals

[1] It may be chosen randomly, for example.

Table 1
Comparison of results of investigated methods

Test function	N	SBGA		EBGA		DE		SADE	
		SR%	NFC	SR%	NFC	SR%	NFC	SR%	NFC
F1	1	100	5566	100	784	100	52	100	72
F3	1	100	5347	100	744	100	98	100	88
Branin	2	81	8125	100	2040	100	506	100	478
Camelback	2	98	1316	100	1316	100	244	100	273
Goldprice	2	59	8125	100	4632	100	350	100	452
PShubert 1	2	63	7192	100	8853	83	1342	100	2738
PShubert 2	2	59	7303	100	4116	90	908	100	1033
Quartic	2	83	8181	100	3168	97	313	100	425
Shubert	2	93	6976	100	2364	94	10,098	100	585
Hartman 1	3	94	1993	100	1680	100	284	100	464
Shekel 1	4	1	7495	97	36,388	72	1968	99	61,243
Shekel 2	4	0	–	98	36,774	91	1851	100	17,078
Shekel 3	4	0	–	100	36,772	89	1752	99	11,960
Hartman 2	6	23	19,452	92	53,792	16	4241	67	2297
Hosc 45	10	0	–	2	126,139	100	1174	100	6438
Brown 1	20	0	–	0	–	100	65,346	95	163,919
Brown 3	20	5	8410	5	106,859	100	41,760	100	43,426
F5n	20	0	–	100	99,945	96	38,045	66	17,785
F10n	20	0	–	49	113,929	90	71,631	47	110,593
F15n	20	0	–	100	102,413	100	44,248	93	28,223

SR = success rate, NFC = average number of function calls, N = dimension of the problem, SBGA = Standard Binary GA, EBGA = Extended Binary GA, DE = Differential Evolution, SADE = Simplified Atavistic DE.

subjected to the differential operator are chosen purely randomly,
- selection of the individuals to survive is simplified: each chromozome has its possible replacement and only the worse in terms of fitness is replaced,
- the mutation operator is omitted.

3.3. Test computations

The DE was examined on previously introduced set of test functions as the binary encoded algorithms and using the same methodology. All computations were performed with identical parameters setting:

$$F_1 = F_2 = 0.85 \text{ and } \Lambda = \{1, 2, \ldots, n\}. \quad (4)$$

The population size was set to pop $= 10n$ for all examined functions. The results are presented in Table 1.

Comparing the results of the DE to those reported in Ref. [1], several interesting findings are apparent. First, the DE shows substantially better reliability solving the most difficult functions where even the extended binary genetic algorithm failed. Only for the *Hartman 2* function the result is unsatisfactory (only 16%). In all other cases the probability of success is better than 70% and in 10 cases it reaches 100% (including *Hosc 45* and both *Brown 1* and *3* functions, for which the extended binary genetic algorithm failed). On the other hand, for several functions, where the binary algorithm shows 100% success, the DE rests at about 70–95%.

Another effect that is evident from the comparison is the fact that the DE is able to find a solution with the same precision much faster (for example, 52 fitness calls contrary to 784 fitness calls using the extended binary algorithm for the $F1$ function). We suppose that this improvement is a consequence of the very good precision adaptability of the differential operators as all alterations of the chromozomes are determined from their mutual distances, see Fig. 2.

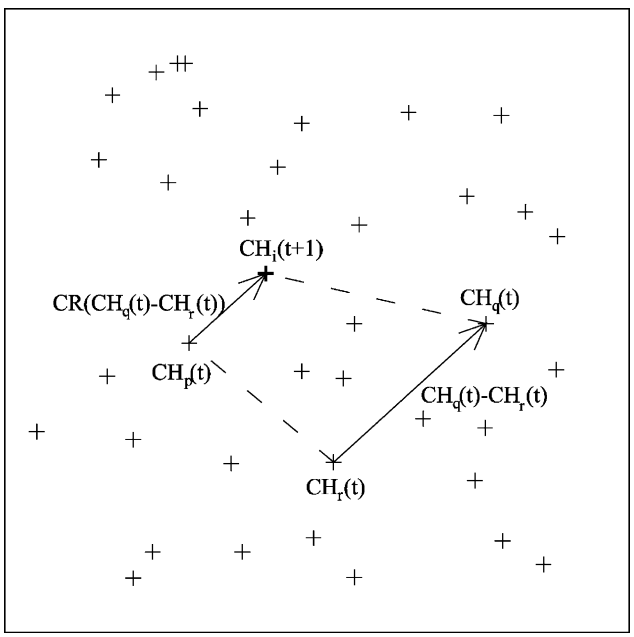

Fig. 2. Geometric meaning of the simplified differential operator.

4. Differential genetic algorithm SADE

This method was proposed as an adaption of the DE after relatively long time of development. Its aim was to acquire a method which is able to solve optimization problems on real domains with a high number of variables (it was tested on problems with up to 200 variables). This algorithm combines the features of the DE with those of the traditional genetic algorithms. It uses the simplified differential operator and the algorithmic scheme similar to the SGA.

4.1. The simplified differential operator

The simplified version of the differential operator taken from the DE is used for the same purpose as the crossing-over in the SGA. Let (again) $CH_i(t)$ be the ith chromozome in a generation t

$$CH_i(t) = (ch_{i1}(t), ch_{i2}(t), \ldots, ch_{in}(t)), \tag{5}$$

where n is the number of variables of the fitness function. Then the simplified differential operator can be written as

$$ch_{ij}(t+1) = ch_{pj}(t) + CR(ch_{qj}(t) - ch_{rj}(t)), \tag{6}$$

where ch_{pj}, ch_{qj} and ch_{rj} are the jth coordinates of three randomly chosen chromozomes and CR is so-called *cross-rate*. Fig. 2 shows the geometrical meaning of this operator. Due to its independence on j, this operator can be also rewritten in the vector form as

$$CH_i(t+1) = CH_p(t) + CR(CH_q(t) - CH_r(t)). \tag{7}$$

4.2. The algorithmic scheme and the operators in detail

Contrary to the DE, the SADE method uses the algorithmic scheme very similar to the SGA:

1. As the first step, the initial population is generated randomly and the fitness function value is assigned to all chromozomes in the population.
2. Several new chromozomes are created using the mutation operators—the mutation and the local mutation (their total number depends on the value of a parameter called *radioactivity*—it gives the mutation probability).
3. Another new chromozomes are created using the simplified differential operator as was described above; the whole amount of chromozomes in the population is now doubled.
4. The fitness function values are assigned to all newly created chromozomes.
5. The selection operator is applied to the double-sized population. Hence, the amount of individuals is decreased to its original value.
6. Steps 2–5 are repeated until a stopping criterion is reached.

Next, we describe the introduced operators in more detail:

Mutation—if a certain chromozome $CH_i(t)$ was chosen to be mutated, a new random chromozome RP is generated and the mutated one $CH_k(t+1)$ is computed using the following relation

$$CH_k(t+1) = CH_i(t) + MR(RP - CH_i(t)), \tag{8}$$

where MR is a parameter called *mutation-rate*.

Local mutation—if a certain chromozome was chosen to be locally mutated, all its coordinates are altered by a random value from a given (usually very small) range.

Crossing-over—instead of traditional cross-over, the SADE method uses the simplified differential operator described above.[2]

Selection—this method uses modified tournament strategy to reduce the population size: two chromozomes are randomly chosen, compared and the worse is rejected. Therefore, the population size is decreased by one. This step is repeated until the population reaches its original size.[3]

The detailed description of the SADE method including source codes in C/C++ and the tests documentation for high-dimensional problems can be obtained from the article [11] and the web-page [16].

4.3. Testing and results

The test computations were performed with the same functions and under the same circumstances as in all previous cases. The population size was set to pop $= 10n$, which is the same value inherited from the DE. Another parameters were set, after several trial runs, to CR $= 0.2$ and MR $= 0.5$, the local mutation range to 0.25% of the domain range of a corresponding variable and the *radioactivity* was considered 20%. The results are shown in Table 1.

Similarly to the DE method, the SADE algorithm shows better behavior concerning the convergence rate and the reliability than binary encoded methods. The overall reliability is even better than for the DE, but for the more complicated problems, the number of the fitness calls is somewhat bigger, even several times. This is caused by different behavior of both methods from the character of the convergence process point of view. While the DE covers relatively large area of the investigated domain during the whole process, the SADE algorithm tends to create a cluster of individuals at a limited sub-area that wanders through

[2] Contrary to the binary genetic algorithm the real encoded method may generate chromozomes outside the given domain. In our implementation, this problem is solved by returning these individuals to the feasible domain boundary.

[3] Contrary to the traditional tournament strategy, this approach can ensure that the best chromozome will not be lost even if it was not chosen to any tournament.

the domain. As a consequence, if the cluster is deadlocked in a local extreme, it is necessary to wait until the mutation gives a chance to escape to another sub-area with better values. Of course, the probability of this event is very low and hence the algorithm must wait a long period of time. This effect causes much worse results for problems with a rather large number of local extremes.

5. Improvement of the differential genetic algorithm to prevent the premature convergence—the CERAF method

As already mentioned in Section 4, the SADE algorithm tends to create clusters of chromozomes, which rather quickly wander through the domain. This behavior somehow recalls gradient optimization methods, however, with several differences: firstly, it operates with more than one possible solution at a time, therefore it is able to better locate the sub-area with the desired solution. Secondly, since the changes of individuals are determined from their mutual distances, this method is able to adapt the step size to reach an optimal solution.

However, each time this method is caught in a local extreme, it has no chance to escape unless a mutation randomly finds a sub-area with better values. But the probability of this effect is very small, especially for the high-dimensional problems. If the gradient optimization methods are applied, this case is usually resolved by so-called *multi-start* principle. It consists of restarting the algorithm many times with different starting points. Similarly, any type of a genetic algorithm could be restarted many times. Nevertheless, the experience shows that there are functions with so-called deceptive behavior, characterized by a high probability that the restarted algorithm would fall again into the same local extreme rather than focus on another sub-area.

Generally speaking, there are several solutions to this obstacle. All of them are based on the leading idea of preventing the algorithm from being trapped in the local extreme that has been already found and to force the algorithm to avoid all of these. As the most natural way, we tried some penalization that deteriorates the fitness function value in the neighborhood of all discovered local extremes. However, this approach did not approve itself—if the shape of a penalization function is not determined appropriately, new local extremes appear at the boundary of a penalization function activity area.

As an alternative, the CERAF[4] method has been introduced. It produces areas of higher level of 'radioactivity' in the neighborhood of all previously found local extremes by increasing the mutation probability in these areas many times (usually we set this probability directly to 100%). The range of the radioactivity area (an n-dimensional ellipsoid) is set to a certain percentage of the domain—we denote it as RAD. The time of stagnation that precedes the markup of a local extreme and initiation of a radioactive zone is another parameter of the method. Similarly to the living nature, the radioactivity in the CERAF method is not constant in time but decreases in an appropriate way: each time some individual is caught in that zone and mutated, the radioactivity zone range is decreased by a small value[5] (for example 0.5%); this recalls the principle of disintegration of a radioactive matter. The radioactive area never disappears completely, so the chromozomes can never find the marked local extreme again.

5.1. The SADE algorithm extended by the CERAF method

Hereafter, the algorithmic scheme of the SADE method is supplied with several steps of the CERAF method. It determines whether some individuals got into any of the discovered 'radioactive zones' and if so, mutates them with a high level of probability. Moreover, when the algorithm stagnates too long, it declares a new radioactivity area:

1. As the first step, the initial population is generated randomly and the fitness function value is assigned to all chromozomes in the population.
2. Several new chromozomes are created using the mutation operators—the mutation and the local mutation (their total number depends on the value of a parameter called *radioactivity*—it gives the mutation probability).
3. Another new chromozomes are created using the simplified differential operator as was described above; the whole amount of chromozomes in the population is now doubled.
4. If any radioactive zone already exists, each chromozome caught in a radioactive area is, with a high probability, subjected to the mutation operation.
5. Depending on the number of chromozomes determined in the previous step, the ranges of radioactive zones are appropriately decreased.
6. The fitness function values are assigned to all newly created chromozomes.
7. The selection operator is applied to the double-sized population. Hence, the amount of individuals is decreased to its original value.
8. The number of stagnating generations is determined and if it exceeds a given limit, the actual best solution is declared as the center of the new radioactive area.
9. Steps 2–8 are repeated until a stopping criterion is reached.

[4] Abbreviation of the French expression *CEntre RAdioactiF*—the radioactivity center.

[5] During the numerical experiments it turned up that the chromozomes created by the mutation parameter should not affect the radioactivity zone range.

Extensive test computations have shown that this methodology can be considered as a universal technique capable of solving any multimodal optimization problem provided that the method that is running underneath (i.e. the algorithm that generates new chromozomes) has a sufficient ability to find new possible solutions. In our case, the SADE algorithm works as the 'exploration' method.

5.2. Test computations results

For the purpose of the algorithm performance testing, the same functions set was used. Also, all parameters of the SADE method rest at the same values as before. The CERAF method parameters were assigned the following values: RAD is a 1/4 of a domain range (for each variable) and the mutation probability inside the radioactive zones is considered 100%. The limit of stagnating generations was set to 1700/pop; this simple heuristic formula seems to work well for a wide variety of problems. The results are given in Table 2 for the cases where the CERAF technology was activated. In all the others the results are the same as for the stand-alone SADE method.

Several interesting facts are evident when comparing these results with the previous cases:

- This method has reached the 100% success for all test functions.
- In many cases the number of fitness calls is the same as for the single SADE algorithm; in those cases the CERAF technology was not even activated because the simple algorithm found the global extreme itself.
- For the last (and the most complicated) functions $F5n$, $F10n$ and $F15n$, the success has been improved from 50–90 to 100%, however, at the cost of slowing down the computation. We consider indeed that the reliability of the method is of greater value than the speed. These are the cases where the algorithm extended by the CERAF method was able to continue searching even after the previous simple method has been caught in a local extreme hopelessly.
- In several cases the computation was even accelerated by the CERAF method, while the reliability was not decreased; in one particular case (the *Hartman 2* function) the reliability was even increased from 67 to 100%. This may appear as a paradox, because the CERAF method needs long periods of stagnation and repeated optimum searching. The acceleration comes from the fact that the method does not have to wait until the random mutation hits an area with better values, but it is forced to start searching in a different location.

Table 2
Results for the SADE + CERAF method

Test function	Dimension	Success rate (%)	Fitness calls
Pshubert 1	2	100	2388
Pshubert 2	2	100	1014
Shekel 1	4	100	3942
Shekel 2	4	100	3746
Shekel 3	4	100	3042
Hartman 2	6	100	15,396
Brown 1	20	100	137,660
F5n	20	100	20,332
F10n	20	100	200,136
F15n	20	100	31,574

6. Conclusions

As we have assumed, the presented results of test computations show definitely that for the optimization of multimodal but still continuous problems on real domains the evolutionary methods based on real encoding and differential operators approve themselves much better than traditional binary genetic algorithms, even when extended by certain improvements. The real encoded algorithms produced better results both in simple cases, where they have reached much better (several times) convergence rates as well as in the complicated cases, where the obtained results were very satisfactory from the reliability point of view, even for functions where binary algorithms have completely failed (e.g. the functions *Brown 1*, *Brown 3* and *Hosc 45*).

The overall reliability-based comparison of all the tested methods is provided in Table 3 (× marks the cases, where the success was better than 95%). Note that the SADE algorithm extended by the CERAF technology have achieved the 100% success for all the test functions.

Table 3
Comparison of reliability

Function	SGA	EGA	DE	SADE	CERAF
F1	×	×	×	×	×
F3	×	×	×	×	×
Branin		×	×	×	×
Camelback	×	×	×	×	×
Goldprice		×	×	×	×
PShubert 1		×		×	×
PShubert 2		×		×	×
Quartic		×		×	×
Shubert		×		×	×
Hartman 1		×	×	×	×
Shekel 1		×		×	×
Shekel 2		×		×	×
Shekel 3		×		×	×
Hartman 2					×
Hosc 45			×	×	×
Brown 1			×		×
Brown 3			×	×	×
F5n		×	×	×	×
F10n					×
F15n		×	×	×	×

Table 4
Comparison of convergence rate

Function	SGA	EGA	DE	SADE	CERAF
F1			×		
F3				×	×
Branin				×	×
Camelback			×		
Goldprice			×		
Pshubert 1			×		
Pshubert 2			×		
Quartic			×		
Shubert				×	×
Hartman 1			×		
Shekel 1			×		
Shekel 2			×		
Shekel 3			×		
Hartman 2				×	
Hosc 45			×		
Brown 1			×		
Brown 3			×		
F5n				×	
F10n			×		
F15n				×	

The next interesting result is that the single SADE algorithm has approximately the same reliability as the binary algorithm extended by several, rather sophisticated, improvements. The reliability of a DE is somehow fluctuating and the standard binary algorithm does not show satisfactory behavior except the most simple cases.

Table 4 shows the comparison of all methods from the convergence rate point of view (× marks the case, where the method reached the result at the shortest time). The DE seems to be the most effective (the fastest optimization method). For other cases, the SADE method or its CERAF extended version were the fastest ones. As could be seen, the binary algorithm has never reached the best convergence rate within this test computations.

Acknowledgements

Authors would like to thank anonymous referees for their careful revision and comments that helped us to substantially improve the quality of the paper. This work has been supported by the Ministry of Education, Youth and Sports of the Czech Republic (MŠMT ČR) under the project No. 210000003.

Appendix A. List of test functions

F1:

$$f(x) = 2(x - 0.75)^2 + \sin(5\pi x - 0.4\pi) - 0.125, \quad (A1)$$

where $0 \le x \le 1$.

F3:

$$f(x) = -\sum_{j=1}^{5} [j \sin[(j+1)x + j]], \quad (A2)$$

where $-10 \le x \le 10$.

Branin:

$$f(x,y) = a(y - bx^2 + cx - d)^2 + h(1-f)\cos x + h, \quad (A3)$$

where $a = 1$, $b = 5.1/4\pi^2$, $c = 5/\pi$, $d = 6$, $h = 10$, $f = 1/8\pi$, $-5 \le x \le 10$, $0 \le y \le 15$.

Camelback:

$$f(x,y) = \left(4 - 2.1x^2 + \frac{x^4}{3}\right)x^2 + xy + (-4 + 4y^2)y^2, \quad (A4)$$

where $-3 \le x \le 3$, $-2 \le y \le 2$.

Goldprice:

$$f(x,y) = [1 + (x + y + 1)^2(19 - 14x + 3x^2 - 14y + 6xy + 3y^2)][30 + (2x - 3y)^2(18 - 32x + 12x^2 + 48y - 36xy + 27y^2)], \quad (A5)$$

where $-2 \le x \le 2$, $-2 \le y \le 2$.

PShubert 1 and 2:

$$f(x,y) = \left\{\sum_{i=1}^{5} i\cos[(i+1)x + i]\right\}\left\{\sum_{i=1}^{5} i\cos[(i+1)y + i]\right\} + \beta[(x + 1.2513)^2 + (y + 0.80032)^2], \quad (A6)$$

where $-10 \le x \le 10$, $-10 \le y \le 10$, for PShubert 1: $\beta = 0.5$ for PShubert 2: $\beta = 1.0$.

Quartic:

$$f(x,y) = \frac{x^4}{4} - \frac{x^2}{2} + \frac{x}{10} + \frac{y^2}{2}, \quad (A7)$$

where $-10 \le x \le 10$, $-10 \le y \le 10$.

Shubert:

$$f(x,y) = \left\{\sum_{i=1}^{5} i\cos[(i+1)x + i]\right\}\left\{\sum_{i=1}^{5} i\cos[(i+1)y + i]\right\}, \quad (A8)$$

where $-10 \le x \le 10$, $-10 \le y \le 10$.

Hartman 1:

$$f(x_1,x_2,x_3) = -\sum_{i=1}^{4} c_i \exp\left(-\sum_{j=1}^{3} a_{ij}(x_i - p_{ij})\right)^2, \quad (A9)$$

where $0 \le x_i \le 1$, $i = 1,\ldots,3$ $x = (x_1,\ldots,x_3)$, $p_i = (p_{i1},\ldots,p_{i3})$, $a_i = (a_{i1},\ldots,a_{i3})$.

i	a_{ij}			c_i	p_{ij}		
1	3.0	10.0	30.0	1.0	0.36890	0.1170	0.2673
2	0.1	10.0	35.0	1.2	0.46990	0.4387	0.7470
3	3.0	10.0	30.0	3.0	0.10910	0.8732	0.5547
4	0.1	10.0	35.0	3.2	0.03815	0.5743	0.8828

Shekel 1, 2 and 3:

$$f(x) = -\sum_{i=1}^{m} \frac{1}{(x-a_i)^{\mathrm{T}}(x-a_i)+c_i}, \tag{A10}$$

where $0 \le x_j \le 10$, for Shekel 1: $m = 5$, for Shekel 2: $m = 7$, for Shekel 3: $m = 10$.

$x = (x_1, x_2, x_3, x_4)^{\mathrm{T}}, \qquad a_i = (a_{i1}, a_{i2}, a_{i3}, a_{i4})^{\mathrm{T}}.$

i	a_{ij}				c_i
1	4.0	4.0	4.0	4.0	0.1
2	1.0	1.0	1.0	1.0	0.2
3	8.0	8.0	8.0	8.0	0.2
4	6.0	6.0	6.0	6.0	0.4
5	3.0	7.0	3.0	7.0	0.4
6	2.0	9.0	2.0	9.0	0.6
7	5.0	5.0	3.0	3.0	0.6
8	8.0	1.0	8.0	1.0	0.7
9	6.0	2.0	6.0	2.0	0.5
10	7.0	3.6	7.0	3.6	0.5

Hartman 2:

$$f(x_1, \ldots, x_6) = -\sum_{i=1}^{4} c_i \exp\Big(-\sum_{j=1}^{6} a_{ij}(x_i - p_{ij})\Big)^2, \tag{A11}$$

where $0 \le x_j \le 1$, $j = 1, \ldots, 6$.

$x = (x_1, \ldots, x_6), \qquad p_i = (p_{i1}, \ldots, p_{i6}), \qquad a_i = (a_{i1}, \ldots, a_{i6}).$

i	a_{ij}						c_i
1	10.00	3.00	17.00	3.50	1.70	8.00	1.0
2	0.05	10.00	17.00	0.10	8.00	14.00	1.2
3	3.00	3.50	1.70	10.00	17.00	8.00	3.0
4	17.00	8.00	0.05	10.00	0.01	14.00	3.2
i	p_{ij}						
1	0.1312	0.1696	0.5569	0.0124	0.8283	0.5886	
2	0.2329	0.4135	0.8307	0.3736	0.1004	0.9991	
3	0.2348	0.1451	0.3522	0.2883	0.3047	0.6650	
4	0.4047	0.8828	0.8732	0.5743	0.1091	0.0381	

Hosc 45:

$$f(x) = 2 - \frac{1}{n!}\prod_{i=1}^{n} x_i, \tag{A12}$$

where $x = (x_1, \ldots, x_n)$, $0 \le x_i \le i$, $n = 10$.

Brown 1:

$$f(x) = \left[\sum_{i\in J}(x_i - 3)\right]^2 + \sum_{i\in J}[10^{-3}(x_i-3)^2 - (x_i - x_{i+1}) + e^{20(x_i - x_{i+1})}], \tag{A13}$$

where $J = \{1, 3, \ldots, 19\}$, $-1 \le x_i \le 4$, $1 \le i \le 20$, $x = (x_1, \ldots, x_{20})^{\mathrm{T}}$.

Brown 3:

$$f(x) = \sum_{i=1}^{19}[(x_i^2)^{(x_{i+1}^2+1)} + (x_{i+1}^2)^{(x_i^2+1)}], \tag{A14}$$

$x = (x_1, \ldots, x_{20})^{\mathrm{T}}, \ -1 \le x_i \le 4, \ 1 \le i \le 20.$

F5n:

$$f(x) = (\pi/20)\left\{10\sin^2(\pi y_1) + \sum_{i=1}^{19}[(y_i - 1)^2 \times (1 + 10\sin^2(\pi y_i + 1))] + (y_{20} - 1)^2\right\}, \tag{A15}$$

where $x = (x_1, \ldots, x_{20})^{\mathrm{T}}, -10 \le x_i \le 10, y_i = 1 + 0.25(x_i - 1)$.

F10n:

$$f(x) = (\pi/20)\left\{10\sin^2(\pi x_1) + \sum_{i=1}^{19}[(x_i - 1)^2 \times (1 + 10\sin^2(\pi x_{i+1}))] + (x_{20} - 1)^2\right\}, \tag{A16}$$

where $x = (x_1, \ldots, x_{20})^{\mathrm{T}}, \ -10 \le x_i \le 10.$

F15n:

$$f(x) = (1/10)\left\{\sin^2(3\pi x_1) + \sum_{i=1}^{19}[(x_i - 1)^2(1 + \sin^2(3\pi x_{i+1}))] + \left((1/10)(x_{20} - 1)^2[1 + \sin^2(2\pi x_{20})]\right\},$$

where $x = (x_1, \ldots, x_{20})^{\mathrm{T}}, \ -10 \le x_i \le 10.$

References

[1] Andre J, Siarry P, Dognon T. An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization. Adv Engng Software 2000;32(1):49–60.

[2] Baker JE. Reducing bias and inefficiency in the selection algorithm. In: Grefenstette J, editor. Proceedings of the First International Conference on Genetic Algorithms. London: Lawrence Erlbaum. p. 101–11.

[3] Davis L. Handbook of genetic algorithms. New York: Van Nostrand Reinhold; 1990.

[4] DE homepage. http://www.icsi.berkeley.edu/storn/code.html

[5] Fan H-Y, Lu JW-Z, Xu Z-B. An empirical comparison of three novel genetic algorithms. Engng Comput 2000;17(8):981–1001.

[6] Foo NY, Bosworth JL. Algebraic, geometric, and stochastic aspects of genetic operators. Technical Report 003120-2-T. University of Michigan; 1972.

[7] Goldberg ED. Genetic algorithms in search, optimization and machine learning. Reading, MA: Addison-Wesley; 1989.

[8] Grefenstette J. Genetic algorithms and their applications. In: Grefenstette J, editor. Proceedings of the Second International Conference on Genetic Algorithms and their Applications. London: Lawrence Erlbaum; 1987.

[9] Holland JH. Adaptation in natural and artificial systems. Internal report. Ann Arbor, MI: University of Michigan; 1975.

[10] Hollstein RB. Artificial genetic adaptation in computer control systems. PhD Thesis. University of Michigan; 1971.
[11] Hrstka O, Kučerová A. Search for optimization method on multi-dimensional real domains. In: Contributions to Mechanics of Materials and Structures. CTU Reports, vol. 4. Czech Technical University in Prague; 2000. p. 87–104.
[12] Lepš M, Šejnoha M. New approach to optimization of reinforced concrete beams. Comput Struct 2003; 81(18–19). 1957–1966.
[13] Matouš K, Lepš M, Zeman J, Šejnoha M. Applying genetic algorithms to selected topics commonly encountered in engineering practice. Comput Meth Appl Mech Engng 2000;190(13–14):1629–50.
[14] Michalewicz Z. Genetic algorithms + data structures = evolution programs, 3rd ed. Berlin: Springer; 1996.
[15] Rafiq YM, Southcombe C. Genetic algorithms in optimal design and detailing of reinforced biaxial columns supported by a declarative approach for capacity checking. Comput Struct 1998; 69(4):443–57.
[16] SADE homepage. http://klobouk.fsv.cvut.cz/~ondra/sade/sade.html
[17] Storn R. On the usage of differential evolution for function optimization. In: NAPHIS; 1996.
[18] Yamamoto K, Inoue O. New evolutionary direction operator for genetic algorithms. AIAA J 1995;33(10):1990–3.