Dynamic Optimization using Self-Adaptive Differential Evolution

Janez Brest, *Member, IEEE*, Aleš Zamuda, *Student Member, IEEE*, Borko Bošković, *Student Member, IEEE*, Mirjam Sepesy Maučec, and Viljem Žumer, *Member, IEEE*

***Abstract*— In this paper we investigate a Self-Adaptive Differential Evolution algorithm (*jDE*) where F and CR control parameters are self-adapted and a multi-population method with aging mechanism is used. The performance of the *jDE* algorithm is evaluated on the set of benchmark functions provided for the CEC 2009 special session on evolutionary computation in dynamic and uncertain environments.**

I. Introduction

EVOLUTIONARY algorithms (EAs) have been widely applied to solve stationary optimization problems. However, many real-world applications are actually dynamic [1].

Differential Evolution (DE) [2], [3] algorithm belongs to EAs and it was proposed by Storn and Price [4], and since then it has been used in many practical cases. The DE has been shown to be a simple yet powerful evolutionary algorithm for optimizing continuous functions, e.g. static optimization environment.

The main objective of this paper is a performance evaluation of our self-adaptive differential evolution algorithm, named *jDE* [5], [6], which uses a self-adapting mechanism on the control parameters F and CR and multi-populations. The performance of the algorithm is evaluated on the set of benchmark functions provided for the CEC 2009 Dynamic Optimization [1].

The article is structured as follows. Section II gives an overview of work dealing with the *jDE* algorithm. Section III shortly presents dynamic optimization problems. In Section IV the self-adaptive differential evolution *jDE* algorithm with multi-populations is briefly described. In Section V experimental results of our *jDE* algorithm on benchmark functions are presented and performance analysis of the algorithm is given. Section VI concludes the paper.

II. Related Work

Historically, applying DE to static optimization problems have attracted much interest, including both theoretical and practical studies [2].

In static optimization process adaptive and self-adaptive DE approaches have become popular recently [7], [5], [8], [9], [10], [11]

Recently, there has been increased interest in evolutionary computation applied to changing optimization problems [12]. The papers [13], [14], [15] survey a number of approaches dealing with dynamic optimization problems (DOPs).

There are many papers about DE and papers about using the DE to solve DOP have also been introduced, recently. The DE is used in dynamic environments in [16], [17], [18].

Janez Brest, Aleš Zamuda, Borko Bošković, Mirjam Sepesy Maučec, and Viljem Žumer are with the Faculty of Electrical Engineering and Computer Science, University of Maribor, Smetanova ul. 17, 2000 Maribor, Slovenia, email: *janez.brest@uni-mb.si*

This work was supported in part by the Slovenian Research Agency under program P2-0041 – Computer Systems, Methodologies, and Intelligent Services.

III. Dynamic Optimization Problems

The report proposed by C. Li et al. [1] uses the generalized dynamic benchmark generator (GDBG) proposed in [19], which constructs dynamic environments for all the three solution spaces. Especially, in the real space, the authors introduce a rotation method instead of shifting the positions of peaks as in the "moving peaks" benchmark (MPB) and DF1 generators. The rotation method can overcome the problem of unequal challenge per change for algorithms of the MPB generator, which happens when the peak positions bounce back from the boundary of the landscape.

The report [1] gives two benchmark instances from the GDBG system in the real space. The source code for the two benchmark instances and a test example using the PSO algorithm are available at `http://www.cs.le.ac.uk/people/syang/ECiDUE/DBG.tar.gz` and `http://www.ntu.edu.sg/home/epnsugan/DBG.tar.gz`, respectively.

As CEC 2009 Dynamic Optimization benchmark problems the following (basic) test functions on real space are used:

- F_1: Rotation peak function,
- F_2: Composition of Sphere's function,
- F_3: Composition of Rastrigin's function,
- F_4: Composition of Griewank's function,
- F_5: Composition of Ackley's function, and
- F_6: Hybrid Composition function.

There are six change types of the system control parameters in the GDBG system. Change type of small step change, large step change, random change, chaotic change, recurrent change, and recurrent change with noise is denoted as T_1–T_6, respectively. Random change type with changed dimension is denoted as T_7.

IV. Our algorithm

In [5] (*jDE* algorithm) the self-adapting control parameter mechanism of "*rand/1/bin*" strategy was used. The self-adaptive control mechanism was used to change the control parameters F and CR during the run. The third control parameter, population size (NP), was not changed during the evolutionary process in [5].

978-1-4244-2959-2/09/$25.00 © 2009 IEEE

TABLE I

ERROR VALUES ACHIEVED FOR PROBLEMS F_1

Dimension(n)	Peaks(m)	Errors	T_1	T_2	T_3	T_4	T_5	T_6
10	10	Avg_best	0	0	0	0	0	0
		Avg_worst	0.910466	32.1705	31.7827	0.919964	18.392	32.7662
		Avg_mean	0.028813	3.5874	2.99962	0.015333	2.17757	1.1457
		STD	0.442537	7.83849	7.12954	0.288388	4.38812	5.72962
	50	Avg_best	0	0	0	0	0	0
		Avg_worst	3.92056	30.1958	27.6823	1.21212	9.08941	33.1204
		Avg_mean	0.172355	4.08618	4.29209	0.0877388	0.948359	1.76542
		STD	0.763932	6.4546	6.74538	0.24613	1.76552	5.82652
T_7(5-15)	10	Avg_best	—	—	0	—	—	—
		Avg_worst	—	—	34.8377	—	—	—
		Avg_mean	—	—	3.5017	—	—	—
		STD	—	—	7.89858	—	—	—
	50	Avg_best	—	—	0	—	—	—
		Avg_worst	—	—	29.768	—	—	—
		Avg_mean	—	—	4.36913	—	—	—
		STD	—	—	6.9321	—	—	—

TABLE II

ERROR VALUES ACHIEVED FOR PROBLEMS F_2

Dimension(n)	Errors	T_1	T_2	T_3	T_4	T_5	T_6
10	Avg_best	0	0	0	0	0	0
	Avg_worst	15.4426	435.019	468.43	10.6608	459.147	49.5327
	Avg_mean	0.963039	43.0004	50.1906	0.793141	67.0523	3.36653
	STD	3.08329	114.944	124.015	2.53425	130.146	12.9738
T_7(5-15)	Avg_best	—	—	0	—	—	—
	Avg_worst	—	—	226.332	—	—	—
	Avg_mean	—	—	13.2524	—	—	—
	STD	—	—	45.7797	—	—	—

The self-adaptive control parameters $F_i^{(G+1)}$ and $CR_i^{(G+1)}$ are calculated as follows [5]:

$$F_i^{(G+1)} = \begin{cases} F_l + rand_1 \cdot F_u & \text{if } rand_2 < \tau_1, \\ F_i^{(G)} & \text{otherwise,} \end{cases}$$

$$CR_i^{(G+1)} = \begin{cases} rand_3 & \text{if } rand_4 < \tau_2, \\ CR_i^{(G)} & \text{otherwise.} \end{cases}$$

They produce control parameters F and CR in a new parent vector. The quantities $rand_j, j \in \{1, 2, 3, 4\}$ represent uniform random values within the range $[0, 1]$. τ_1 and τ_2 are probabilities to adjust control parameters F and CR, respectively. τ_1, τ_2, F_l, F_u were taken fixed values $0.1, 0.1, 0.1, 0.9$, respectively. The new F takes value from $[0.1, 1.0]$ and the new CR from $[0, 1]$ in a random manner. The control parameter values $F_i^{(G+1)}$ and $CR_i^{(G+1)}$ are obtained before the mutation operation is performed. This means they influence the mutation, crossover and selection operations of the new vector $\vec{x}_i^{G+1}$.

Some ideas, how to improve the *jDE* algorithm, are reported in [20], [21], [10], but here we used the original *jDE* algorithm and the mechanism as described hereinafter.

For static optimization convergence is desired. If the problem is dynamic fast convergence could be dangerous. The idea of this paper is to extended our *jDE* algorithm by using more

TABLE III

ERROR VALUES ACHIEVED FOR PROBLEMS F_3

Dimension(n)	Errors	T_1	T_2	T_3	T_4	T_5	T_6
10	Avg_best	0	9.70434e-08	3.13019e-10	0	5.35102e-10	8.17124e-14
	Avg_worst	238.417	938.858	944.695	922.236	874.852	1226.38
	Avg_mean	11.3927	558.497	572.105	65.7409	475.768	243.27
	STD	58.1106	384.621	386.09	208.925	379.89	384.98
T_7(5-15)	Avg_best	—	—	0	—	—	—
	Avg_worst	—	—	853.061	—	—	—
	Avg_mean	—	—	153.673	—	—	—
	STD	—	—	286.379	—	—	—

TABLE IV

ERROR VALUES ACHIEVED FOR PROBLEMS F_4

Dimension(n)	Errors	T_1	T_2	T_3	T_4	T_5	T_6
10	Avg_best	0	0	0	0	0	0
	Avg_worst	19.623	475.7	544.92	16.6057	510.193	28.4483
	Avg_mean	1.48568	49.5044	51.9448	1.50584	69.4395	2.35478
	STD	4.47652	135.248	141.78	4.10062	144.041	5.78252
T_7(5-15)	Avg_best	—	—	0	—	—	—
	Avg_worst	—	—	163.727	—	—	—
	Avg_mean	—	—	11.7425	—	—	—
	STD	—	—	39.4469	—	—	—

TABLE V

ERROR VALUES ACHIEVED FOR PROBLEMS F_5

Dimension(n)	Errors	T_1	T_2	T_3	T_4	T_5	T_6
10	Avg_best	4.10338e-14	4.16556e-14	4.15668e-14	4.08562e-14	4.24549e-14	4.08562e-14
	Avg_worst	4.89413	9.6899	10.1371	4.75098	9.28981	4.78684
	Avg_mean	0.159877	0.333918	0.357925	0.108105	0.409275	0.229676
	STD	1.02554	1.64364	1.83299	0.826746	1.90991	0.935494
T_7(5-15)	Avg_best	—	—	4.12115e-14	—	—	—
	Avg_worst	—	—	11.8188	—	—	—
	Avg_mean	—	—	0.434294	—	—	—
	STD	—	—	2.22792	—	—	—

TABLE VI
ERROR VALUES ACHIEVED FOR PROBLEMS F_6

Dimension(n)	Errors	T_1	T_2	T_3	T_4	T_5	T_6
10	Avg_best	0	0	0	0	0	0
	Avg_worst	32.7204	51.8665	84.519	38.7914	191.895	45.0354
	Avg_mean	6.22948	10.3083	10.954	6.78734	14.9455	7.8028
	STD	10.4373	13.2307	23.2974	10.1702	45.208	10.9555
T_7(5-15)	Avg_best	—	—	0	—	—	—
	Avg_worst	—	—	58.9448	—	—	—
	Avg_mean	—	—	10.736	—	—	—
	STD	—	—	14.7267	—	—	—

TABLE VII
ALGORITHM OVERALL PERFORMANCE

	F_1(10)	F_1(50)	F_2	F_3	F_4	F_5	F_6
T_1	0.014768	0.0146876	0.0211049	0.0157107	0.0206615	0.021766	0.0170472
T_2	0.0136901	0.0135926	0.0135271	0.00298238	0.013148	0.0208661	0.0139488
T_3	0.0138256	0.0135304	0.0130808	0.00281439	0.013545	0.0209286	0.0141912
T_4	0.0147164	0.0146941	0.0210035	0.0127621	0.0199268	0.0221962	0.0153046
T_5	0.0139415	0.0143644	0.0123976	0.0044056	0.012376	0.0213094	0.0155184
T_6	0.0141265	0.013874	0.017776	0.00734523	0.0179501	0.0207361	0.0139512
T_7	0.00911221	0.00898569	0.0101876	0.00549392	0.0101813	0.0137894	0.00942562
Mark	0.0941803	0.0937288	0.109078	0.0515143	0.107789	0.141592	0.099387
Performance (sumed the mark obtained for each case and multiplied by 100): 69.7269							

populations. To maintain the diversity of general population, it was divided into few smaller population.

We used a multi-population *jDE* algorithm without any information sharing between the populations (except overlapping search between two best individuals of two subpopulations). The last is implemented in our algorithm in a way that random indexes r_1, r_2, and r_3 indicate vectors (individuals) that belong to same subpopulation as the trial vector $\vec{x}_i$.

In this paper, the size of sub-population is small and therefore we changed the value of F_l to 0.36. We follow suggestion of Zaharie [22] about critical values for the control parameters of DE. The values of the parameters which satisfy $2F - 2/NP + CR/NP = 0$ can be considered to be critical. In our case lower bound values are $CR = 0$ and $NP = 10$, therefore critical value for F is 0.308 (0.424 when $NP = 5$). As already mentioned we set $F_l = 0.36$ in this paper.

We have employed aging at individual level. An individual which stagnates in local minimum should be reinitialized in some way. We used the aging strategy for an individual, say i, as presented in Algorithm 1.

Algorithm 1 Aging of the i-th individual

1. **if** i is global-best **then** do not use aging
2. **else if** i is local-best (best in sub-population) **and** $age > 30$ **and** $mRand() < 0.1$ **then**
3. reinitialize sub-population that contains i
4. **else if** $age > 25$ **and** $mRand() < 0.1$ **then** reinitialize i

Usually, overlapping search between two subpopulations can be checked by comparing the distance of the best individuals of the subpopulations (Algorithm 2).

Algorithm 2 Overlapping search

1. **if** i-th individual is local best of first sub-population **and** distance to the best individual of second sub-populations is lower than 0.05 **then**
2. reinitialize sub-population that contains i-th individual.

We never reinitialize subpopulation that contains the global best individual.

In subpopulation we use distance measure to reinitialize an individual if it is placed near the local best (Algorithm 3).

Algorithm 3 Individual is close to local best

1. **if** i-th individual is not local-best and it is close (0.0001) to the local best **then**
2. reinitialize i-th individual

Let us denote with x currently generated new individual, and with y its parent (both x and y are being compared in DE selection operational and better one survives). The age of individual is incremented by 1 in each generation. Here we additionally introduce a rule how to control age of an individual, based on distance between x and y and their fitnesses, as shown in Algorithm 4.

Algorithm 4 Improvement and aging

1. **if** `x.Distance(y) < 0.01` **or** `|x.fitness-y.fitness| < 0.1` **then**
2. $age = \min\{age, 20\}$
3. **else**
4. $age = \min\{age, 5\}$

We have implemented reinitialization of individual i to get new individual y as presented in Algorithm 5.

Algorithm 5 Reinitialization

$subSize$: size of subpopulation
ARC: archive of individuals
$arcNum$: size of archive

1. **if** ($mRand() < 0.5$ **and** $i < subSize$ **and** $arcNum > 0$ **and** $i < arcNum$) **then**
2. t is random number from set $\{0, arcNum\}$
3. $y = ARC[t]$
4. $w = \frac{0.1}{1+(i \mod subSize)}$
5. **if** $(i \mod 2) == 1$ **then**
6. $y = y + w \cdot N(0, 1)$
7. **else**
8. $y = y + w \cdot U(-0.5, 0.5)$
9. **end if**
10. **else**
11. generate y in a random manner
12. **end if**

Archive is empty at the beginning of the evolutionary process, later the archive is increased by 1 after each change is detected – currently best individual is added to the archive. If condition in line 1 (Algorithm 5) is satisfied, an individual is selected from archive only for the first subpopulation, otherwise it is generated randomly.

When the whole subpopulation is reinitialized it is done in same manner as individual reinitialization – we reinitialize $subSize$ individuals.

V. Experimental Results

The *jDE* algorithm was tested on CEC 2009 special session benchmark functions [1]. 20 runs of algorithm were needed for each function.

For each change type of each function, we present the following values for $x_{best}(t)$ over 20 runs: average best, average mean, average worst values and standard deviations.

Tables I–VI show the values over 20 runs.

Parameter settings used in the experiments were:

- F was self-adaptive, initially set to 0.5,
- CR was self-adaptive, initially set to 0.9,
- NP was 50,
- number of sub-populations was 5.

The size of each sub-population was 10.

The obtained results as required for CEC 2009 Dynamic Optimization competition are presented in Tables I–VII and Figures 1–6. Overall performance of our algorithm is presented in Table VII. Our algorithm performed excellent on small step (T_1) and chaotic (T_4) change types for functions F_1, F_2, F_3, and F_4. It obtained good results over all change type for function F_5 (Ackley's function). It performed very well for function F_6 (Hybrid composition function) over all change types if we compared average best values. Function F_3 (Composition of Rastrigin's function) is the most difficult one among all test problems. This is also clearly reflected from mark scores in Table VII.

Convergence graphs for all functions are depicted in Figures 1–6. Note, that relative values depicted on the figures are presented as $r(t)+(i-1)$, where $0 \leq r(t) \leq 1, i = 1, 2, \cdots, 7$.

Experimental environment:

- System: GNU/Linux x86_64
- CPU: 2.4 GHz (2x Dual Core AMD Opteron)
- RAM: 8 GB
- Language: C/C++
- Algorithm: *jDE* – Differential Evolution, multi-population
- Runs/problem: 20.

VI. Conclusions

In this paper the performance of the *jDE* algorithm was evaluated on the set of benchmark functions provided by CEC 2009 special session on dynamic optimization problems.

A self-adaptive control mechanism was used by the algorithm to change the control parameters (F and CR) during the optimization process. Additionally, the algorithm has used multi-populations and aging mechanism to solve DOPs.

One of future plans is to apply additional cooperation between sub-population in our algorithm.

Our goal in this work was to make experiments and to present obtained results as required by technical report for CEC 2009 [1]. The overall performance (see Table VII) obtained by the *jDE* is approximately 69.7.

Acknowledgments

The authors would also like to acknowledge the efforts of the organizers of this session and availability of test function suite code.

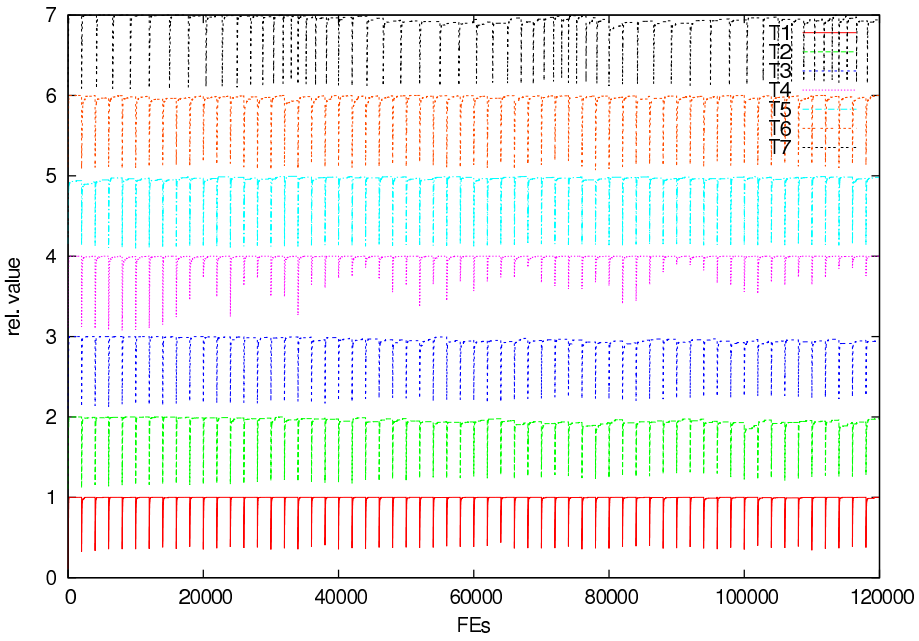

Fig. 1. Convergence graphs for problems F_1

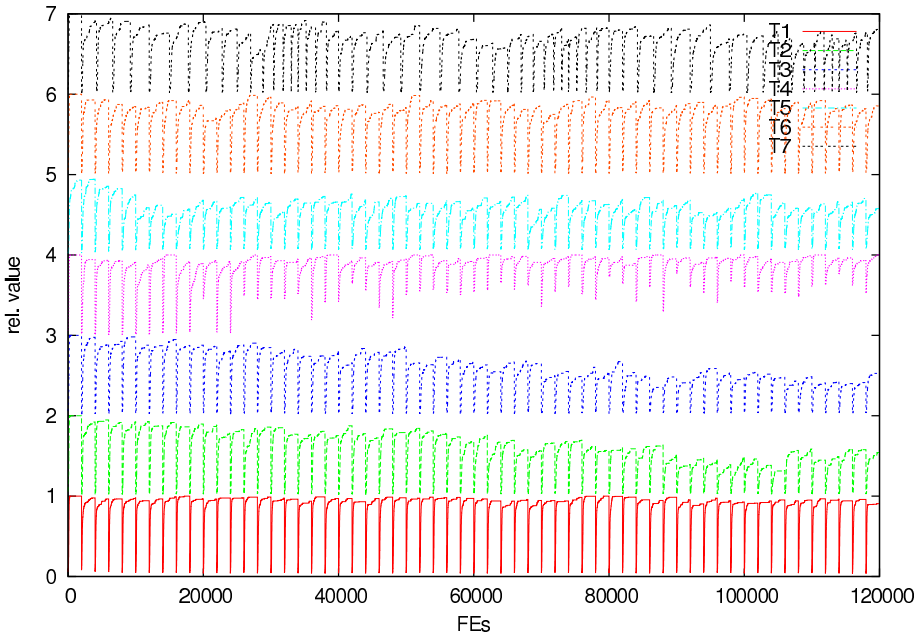

Fig. 2. Convergence graphs for problems F_2

REFERENCES

[1] C. Li, S. Yang, T. T. Nguyen, E. L. Yu, X. Yao, Y. Jin, H.-G. Beyer, and P. N. Suganthan, "Benchmark Generator for CEC 2009 Competition on Dynamic Optimization," University of Leicester, University of Birmingham, Nanyang Technological University, Tech. Rep., 2008.

[2] K. V. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution, A Practical Approach to Global Optimization*. Springer, 2005.

[3] V. Feoktistov, *Differential Evolution: In Search of Solutions (Springer Optimization and Its Applications)*. Secaucus, NJ, USA: Springer-Verlag New York, Inc., 2006.

[4] R. Storn and K. Price, "Differential Evolution – A Simple and Efficient Heuristic for Global Optimization over Continuous Spaces," *Journal of Global Optimization*, vol. 11, pp. 341–359, 1997.

[5] 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 Transactions on Evolutionary Computation*, vol. 10, no. 6, pp. 646–657, 2006.

[6] J. Brest, V. Žumer, and M. S. Maučec, "Self-adaptive Differential Evolution Algorithm in Constrained Real-Parameter Optimization," in *The 2006 IEEE Congress on Evolutionary Computation CEC2006*. IEEE Press, 2006, pp. 919–926.

[7] J. Teo, "Exploring dynamic self-adaptive populations in differential evolution," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 10, no. 8, pp. 673–686, 2006.

[8] Z. Yang, K. Tang, and X. Yao, "Self-adaptive differential evolution with neighborhood search," *Evolutionary Computation, 2008. CEC 2008.*

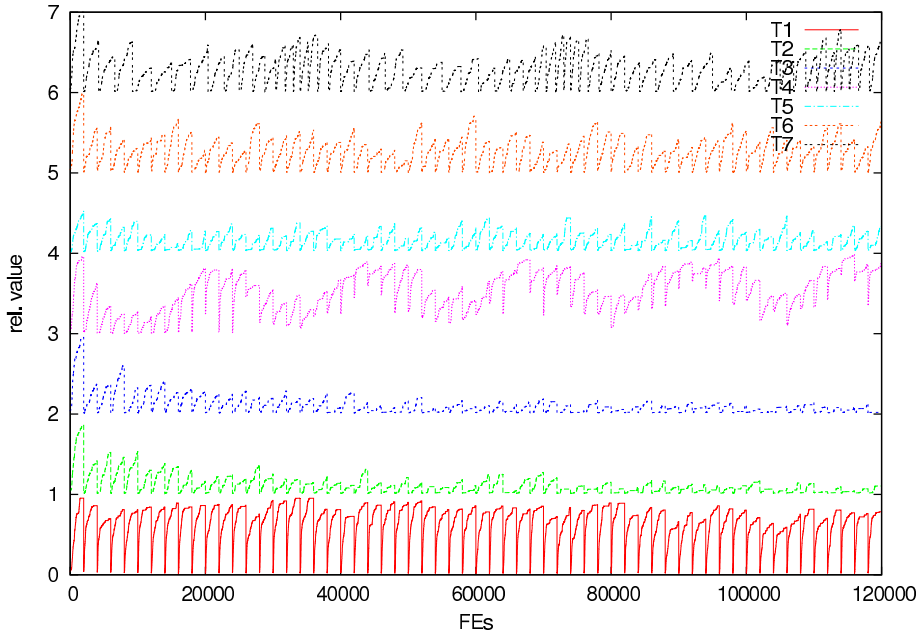

Fig. 3. Convergence graphs for problems F_3

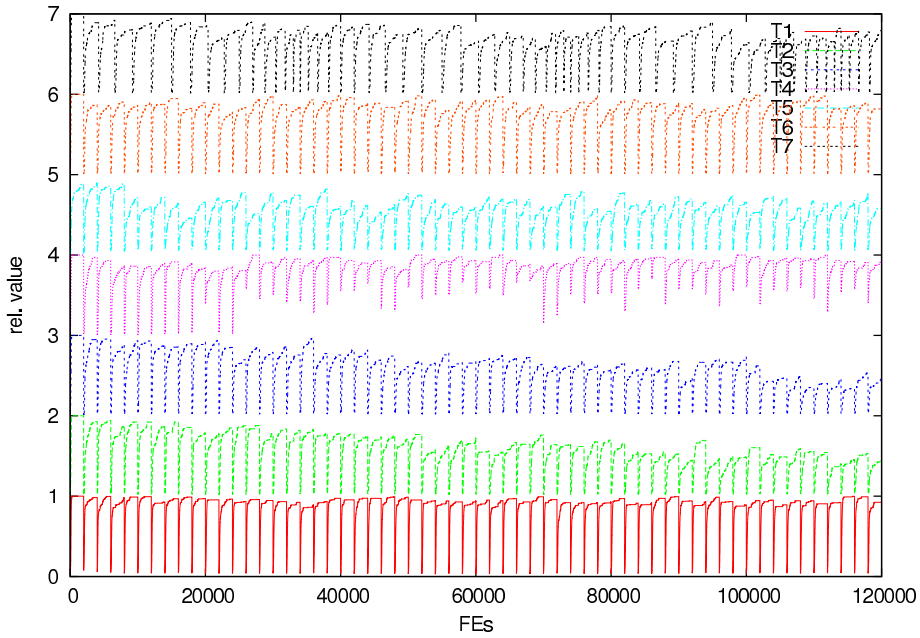

Fig. 4. Convergence graphs for problems F_4

(IEEE World Congress on Computational Intelligence). IEEE Congress on, pp. 1110–1116, June 2008.

[9] A. K. Qin, V. L. Huang, and P. N. Suganthan, "Differential evolution algorithm with strategy adaptation for global numerical optimization," *Evolutionary Computation, IEEE Transactions on*, dOI: 10.1109/TEVC.2008.927706, Accepted.

[10] J. Brest and M. S. Maučec, "Population Size Reduction for the Differential Evolution Algorithm," *Applied Intelligence*, vol. 29, no. 3, pp. 228–247, 2008.

[11] J. Zhang and A. C. Sanderson, "JADE: Adaptive Differential Evolution with Optional External Archive," *Evolutionary Computation, IEEE Transactions on*, dOI: 10.1109/TEVC.2008.927706, Accepted.

[12] J. Branke, "Memory enhanced evolutionary algorithms for changing optimization problems," *Proceedings of the 1999 Congress on Evolutionary Computation, CEC'99.*, vol. 3, pp. 1875–1882 Vol. 3, 1999.

[13] Y. Jin and J. Branke, "Evolutionary optimization in uncertain environments-a survey," *Evolutionary Computation, IEEE Transactions on*, vol. 9, no. 3, pp. 303–317, June 2005.

[14] T. Blackwell and J. Branke, "Multiswarms, exclusion, and anti-convergence in dynamic environments," *Evolutionary Computation, IEEE Transactions on*, vol. 10, no. 4, pp. 459–472, Aug. 2006.

[15] S. Yang and X. Yao, "Population-based incremental learning with associative memory for dynamic environments," *Evolutionary Computation, IEEE Transactions on*, vol. 12, no. 5, pp. 542–561, Oct. 2008.

[16] R. Mendes and A. Mohais, "Dynde: a differential evolution for dynamic optimization problems," *Evolutionary Computation, 2005. The 2005*

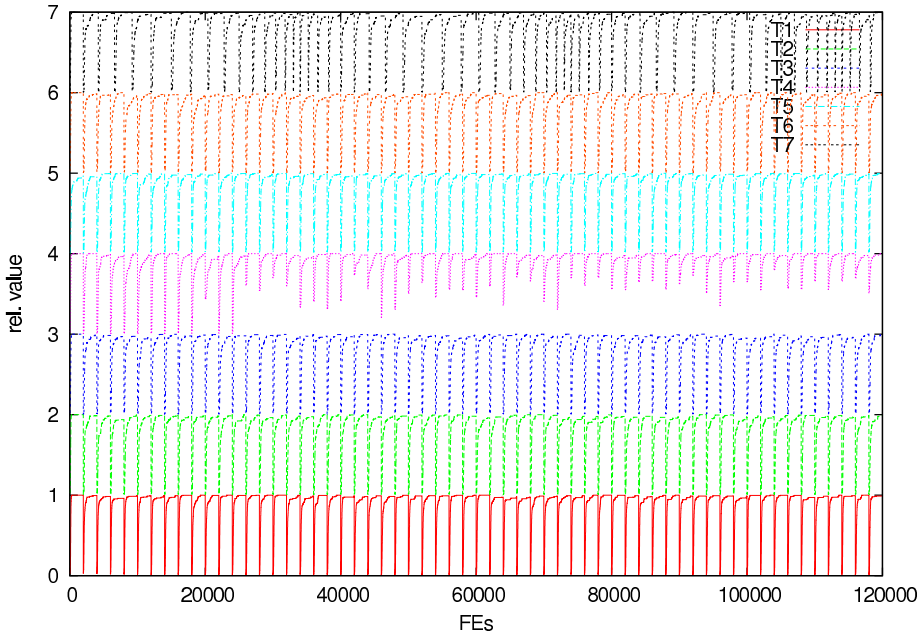

Fig. 5. Convergence graphs for problems F_5

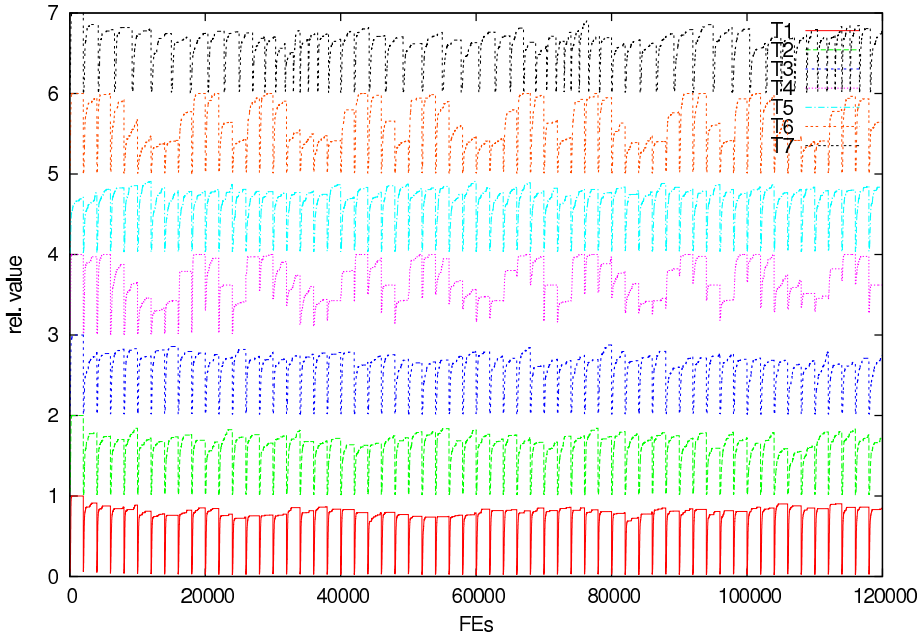

Fig. 6. Convergence graphs for problems F_6

IEEE Congress on, vol. 3, pp. 2808–2815 Vol. 3, Sept. 2005.

[17] D. Zaharie and F. Zamfirache, "Diversity Enhancing Mechanisms for Evolutionary Optimization in Static and Dynamic Environments," in *Proc. of 3rd Romanian-Hungarian Joint Symposium on Applied Computational Intelligence*, 2006, pp. 460–471.

[18] A. E. Kanlikilicer, A. Keles, and A. S. Uyar, "Experimental analysis of binary differential evolution in dynamic environments," in *GECCO '07: Proceedings of the 2007 GECCO conference companion on Genetic and evolutionary computation*. New York, NY, USA: ACM, 2007, pp. 2509–2514.

[19] C. Li and S. Yang, "A Generalized Approach to Construct Benchmark Problems for Dynamic Optimization," in *Proc. of the 7th Int. Conf. on Simulated Evolution and Learning*, 2008.

[20] J. Brest, B. Bošković, S. Greiner, V. Žumer, and M. S. Maučec, "Performance comparison of self-adaptive and adaptive differential evolution algorithms," *Soft Computing - A Fusion of Foundations, Methodologies and Applications*, vol. 11, no. 7, pp. 617–629, 2007.

[21] J. Brest, A. Zamuda, B. Bošković, M. S. Mauec, and V. Žumer, "High-dimensional Real-parameter Optimization Using Self-adaptive Differential Evolution Algorithm with Population Size Reduction," in *2008 IEEE World Congress on Computational Intelligence*. IEEE Press, 2008, pp. 2032–2039.

[22] D. Zaharie, "Critical Values for the Control Parameters of Differential Evolution Algorithms," in *Proc. of Mendel 2002, 8th International Conference on Soft Computing*, 2002, pp. 62–67.