Opposition-Based Differential Evolution

Shahryar Rahnamayan, *Member, IEEE*, Hamid R. Tizhoosh, and Magdy M. A. Salama, *Fellow, IEEE*

***Abstract*—Evolutionary algorithms (EAs) are well-known optimization approaches to deal with nonlinear and complex problems. However, these population-based algorithms are computationally expensive due to the slow nature of the evolutionary process. This paper presents a novel algorithm to accelerate the differential evolution (DE). The proposed opposition-based DE (ODE) employs opposition-based learning (OBL) for population initialization and also for generation jumping. In this work, opposite numbers have been utilized to improve the convergence rate of DE. A comprehensive set of 58 complex benchmark functions including a wide range of dimensions is employed for experimental verification. The influence of dimensionality, population size, jumping rate, and various mutation strategies are also investigated. Additionally, the contribution of opposite numbers is empirically verified. We also provide a comparison of ODE to fuzzy adaptive DE (FADE). Experimental results confirm that the ODE outperforms the original DE and FADE in terms of convergence speed and solution accuracy.**

***Index Terms*—Differential evolution (DE), evolutionary algorithms, opposition-based learning, opposite numbers, optimiztion.**

I. Introduction

DIFFERENTIAL EVOLUTION (DE) was proposed by Price and Storn in 1995 [1], [2]. It is an effective, robust, and simple global optimization algorithm [3] which only has a few control parameters. According to frequently reported comprehensive studies [3]–[6], DE outperforms many other optimization methods in terms of convergence speed and robustness over common benchmark functions and real-world problems. Generally speaking, all population-based optimization algorithms, no exception for DE, suffer from long computational times because of their evolutionary/stochastic nature. This crucial drawback sometimes limits their application to offline problems with little or no real-time constraints.

The concept of *opposition-based learning* (OBL) was introduced by Tizhoosh [7] and has thus far been applied to accelerate reinforcement learning [8]–[10] and backpropagation learning in neural networks [11]. The main idea behind OBL is the simultaneous consideration of an estimate and its corresponding opposite estimate (i.e., guess and opposite guess) in order to achieve a better approximation for the current candidate solution. In this paper, OBL has been utilized to accelerate the convergence rate of DE. Hence, our proposed approach has been called opposition-based differential evolution (ODE). ODE uses opposite numbers during population initialization and also for generating new populations during the evolutionary process. To the best of our knowledge, this is the first time that opposite numbers have been utilized to speed up the convergence rate of an optimization algorithm.

Purely random resampling or selection of solutions from a given population has the chance of visiting or even revisiting unproductive regions of the search space. As we will demonstrate, the chance of this occurring is lower for opposite numbers than it is for purely random ones. In fact, a mathematical proof has been proposed to show that, in general, opposite numbers are more likely to be closer to the optimal solution than purely random ones [12]. In this paper, the usefulness of opposite numbers is empirically investigated by replacing them with random numbers and investigating the consequences. Our experimental results serve to confirm the theoretical expectations. Furthermore, the proposed schemes for population initialization and generation jumping are simple and general enough to be embedded inside other DE versions.

This paper provides a comprehensive set of experimental verifications of our proposed approach. Specifically, we investigate: 1) the convergence speed and robustness; 2) the effect of dimensionality; 3) the advantage of opposite points over random ones; 4) the population size; 5) the mutation operator; 6) the setting of newly added control parameter, the jumping rate, and 7) ODE is compared with fuzzy adaptive DE (FADE). For all these experiments, a comprehensive set of 58 well-known benchmark functions is employed. Furthermore, in order to fairly compare the algorithms, a complete set of performance measures has been utilized. These measures are the average number of function calls (NFCs), success rate (SR), average SR, number of unsolved problems, number of functions for which the algorithm outperforms its competitor, acceleration rate (AR), and the average of the best found solutions and the corresponding standard deviation as calculated after specific number of function evaluations has been reached.

Organization of the rest of this paper is as follows. The DE algorithm is shortly reviewed in Section II. In Section III, the concept of OBL is briefly explained. The proposed algorithm is presented in Section IV and a comprehensive set of experimental results are provided in Section V. Finally, the work is concluded in Section VI. Appendix A lists all test functions, and Appendix B provides complementary results.

Manuscript received October 20, 2006; revised January 10, 2007. The work of S. Rahnamayan was supported by Ontario Graduate Scholarship (OGS).

S. Rahnamayan is with the Pattern Analysis and Machine Intelligence Laboratory, University of Waterloo, Department of Systems Design Engineering, Waterloo, ON N2L 3G1, Canada (e-mail: shahryar@pami.uwaterloo.ca).

H. R. Tizhoosh is with the Pattern Analysis and Machine Intelligence Laboratory, University of Waterloo, Department of Systems Design Engineering, Waterloo, ON N2L 3G1, Canada (e-mail: tizhoosh@uwaterloo.ca).

M. M. A. Salama is with the Department of Electrical and Computer Engineering, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: m.salama@ece.uwaterloo.ca).

Digital Object Identifier 10.1109/TEVC.2007.894200

TABLE I
COMPARISON OF DE AND ODE ON FOUR DIFFERENT MUTATION STRATEGIES ($f_1 - f_{30}$). THE BEST RESULT OF EACH MUTATION STRATEGY IS EMPHASIZED IN **BOLDFACE** AND THE BEST RESULT AMONG FOUR MUTATION STRATEGIES (EIGHT RESULTS) IS HIGHLIGHTED IN *ITALIC* FONTS

		DE/rand/1/bin					**DE/rand/1/exp**					**DE/rand/2/exp**					**DE/rand/2/bin**				
		DE		*ODE*			*DE*		*ODE*			*DE*		*ODE*			*DE*		*ODE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	87748	1	**47716**	1	1.83	86096	1	*47544*	1	1.81	675148	1	**74652**	1	9.05	683932	1	**76920**	1	8.89
f_2	30	96488	1	*53304*	1	1.81	95840	1	**53612**	1	1.88	742468	1	**84728**	1	8.76	741908	1	**84904**	1	8.74
f_3	20	177880	1	**168680**	1	1.05	*146336*	1	147652	1	0.99	781680	1	**393652**	1	1.99	–	0	**481992**	**1**	–
f_4	30	**403112**	**1**	–	0	–	*302072*	**1**	–	0	–	–	0	**343340**	**1**	–	–	0	**463880**	**1**	–
f_5	10	328844	**1**	*70389*	0.76	4.67	391528	**1**	**70983**	0.92	5.52	–	0	**186370**	**0.72**	–	–	0	**172860**	**0.72**	–
f_6	30	113428	**1**	**69342**	0.96	1.64	112892	**1**	**69475**	0.96	1.62	923540	0.16	**126370**	**0.92**	7.30	*29968*	1	29760	1	1.00
f_7	30	25140	1	**8328**	1	3.01	25520	1	*8032*	1	3.18	296028	1	**11316**	1	26.16	302972	1	**11736**	1	25.81
f_8	30	169152	1	*98296*	1	1.72	168468	1	**98392**	1	1.71	–	0	**159172**	**1**	–	–	0	**159296**	**1**	–
f_9	2	**4324**	1	4776	1	0.90	*4268*	1	4684	1	0.91	**5512**	1	5712	1	0.97	**5676**	1	5776	1	0.98
f_{10}	4	*16600*	1	19144	1	0.87	**17020**	1	19216	1	0.89	30292	1	**29896**	1	1.01	–	0	–	0	–
f_{11}	2	8016	1	**6608**	1	1.21	8052	1	*6464*	1	1.25	12828	1	**7744**	1	1.66	12916	1	**7404**	1	1.74
f_{12}	3	*3376*	1	3580	1	0.94	**3552**	1	3612	1	0.98	4352	1	**4146**	1	1.05	44444	1	**4196**	1	1.06
f_{13}	6	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{14}	2	5352	1	**4468**	1	1.20	5236	1	*4260*	1	1.23	7928	1	**5004**	1	1.58	8020	1	**4920**	1	1.63
f_{15}	30	101460	1	*70408*	1	1.44	100264	1	**71040**	1	1.41	948920	0.96	**188496**	**1**	5.03	963740	0.84	**195096**	**1**	4.94
f_{16}	100	3608	1	**3288**	1	1.09	3496	1	*3148*	1	1.11	4440	**1**	**3616**	1	1.23	4588	1	**3724**	1	1.23
f_{17}	4	549850	0.04	**311800**	**0.12**	1.76	*255230*	**0.20**	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{18}	10	**191340**	**0.76**	213330	0.56	0.90	*189830*	**0.60**	279260	0.32	0.68	–	0	**825330**	**0.08**	–	–	0	–	0	–
f_{19}	30	385192	1	**369104**	1	1.04	331864	1	*331040*	1	1.00	–	0	–	0	–	–	0	–	0	–
f_{20}	2	*4884*	1	5748	1	0.85	**5196**	1	6032	1	0.86	**9628**	1	10052	1	0.96	**9612**	1	9704	1	0.99
f_{21}	30	**187300**	1	155636	1	1.20	*185176*	1	155572	1	1.19	–	0	**281560**	**1**	–	–	0	**280328**	**1**	–
f_{22}	30	570290	0.28	**72250**	**0.88**	7.89	560100	0.08	*71296*	**0.96**	7.86	–	0	**97824**	**1**	–	–	0	**98380**	**1**	–
f_{23}	30	41588	1	**23124**	1	1.80	41112	1	*23024*	1	1.79	366972	1	**39144**	1	9.37	366852	1	**38552**	1	9.52
f_{24}	30	818425	0.12	*199810*	**1**	1.78	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{25}	4	*13925*	**0.80**	15280	0.60	0.91	**14625**	**0.96**	85507	0.52	0.17	**18921**	**0.76**	21830	0.40	0.87	**18421**	**0.76**	104890	0.44	0.18
f_{26}	4	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{27}	4	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–	–	0	–	0	–
f_{28}	4	4576	1	**4500**	1	1.02	4800	1	*4368*	1	1.10	7944	1	**6800**	1	1.17	8032	1	**6832**	1	1.18
f_{29}	2	**10788**	1	11148	1	0.97	*10724*	1	10836	1	0.99	15280	1	**14644**	1	1.04	15168	1	**14884**	1	1.02
f_{30}	2	1016	1	**996**	1	1.02	1072	1	*980*	1	1.10	1240	1	**1108**	1	1.12	1292	1	**1120**	1	1.15

II. A BRIEF INTRODUCTION TO DIFFERENTIAL EVOLUTION

Differential evolution (DE) is a population-based and directed search method [25], [26]. Like other evolutionary algorithms (EAs), it starts with an initial population vector, which is randomly generated when no preliminary knowledge about the solution space is available. Let us assume that $X_{ri,G}(i = 1, 2, \ldots, N_p)$ are solution vectors in generation G ($N_p =$ population size). Successive populations are generated by adding the weighted difference of two randomly selected vectors to a third randomly selected vector. For classical DE ($\mathrm{DE/rand/1/bin}$), the mutation, crossover, and selection operators are straightforwardly defined as follows.

Mutation—For each vector $X_{ri,G}$ in generation G a mutant vector $V_{i,G}$ is defined by

$$V_{i,G} = X_{r1,G} + F(X_{r2,G} - X_{r3,G}) \tag{1}$$

where $i = \{1, 2, \ldots, N_p\}$ and r_1, r_2, and r_3 are mutually different random integer indices selected from $\{1, 2, \ldots, N_p\}$. Further, i, r_1, r_2, and r_3 are different so that $N_p \geq 4$ is required. $F \in [0, 2]$ is a real constant which determines the amplification of the added differential variation of $(X_{r2,G} - X_{r3,G})$. Larger values for F result in higher diversity in the generated population and lower values cause faster convergence.

Crossover—DE utilizes the crossover operation to generate new solutions by shuffling competing vectors and also to increase the diversity of the population. It defines the following trial vector:

$$U_{i,G} = (U_{1i,G}, U_{2i,G}, \ldots, U_{Di,G}) \tag{2}$$

where $j = 1, 2, \ldots, D$ ($D =$ problem dimension) and

$$U_{ji,G} = \begin{cases} V_{ji,G}, & \text{if } \mathrm{rand}_j(0,1) \leq C_r \vee j = k \\ X_{ji,G}, & \text{otherwise} \end{cases}. \tag{3}$$

$C_r \in (0, 1)$ is the predefined crossover rate constant, and $\mathrm{rand}_j(0, 1)$ is the jth evaluation of a uniform random number generator. $k \in \{1, 2, \ldots, D\}$ is a random parameter index, chosen once for each i to make sure that at least one parameter is always selected from the mutated vector $V_{ji,G}$. Most popular values for C_r are in the range of $(0.4, 1)$ [27].

Selection—It is an approach which must decide which vector ($U_{i,G}$ or $X_{i,G}$) should be a member of next (new) generation $G + 1$. For a maximization problem, the vector with the higher fitness value is chosen. There are other variants based on different mutation and crossover strategies [2].

Other studies have been conducted to enhance the performance of the original DE algorithm, such as, adaptively determining DE's control parameters. The FADE algorithm was introduced by Liu and Lampinen [13]. They employed a fuzzy logic controller to set the mutation and crossover rates. In the same direction, Brest *et al.* [15] proposed self-adapting control parameter settings. Teo proposed a dynamic population sizing strategy based on self-adaptation [17] and Ali and Törn [18] introduced auxiliary population and automatic calculating of the amplification factor F, for the difference vector.

Some others have experimented with multipopulation ideas. Tasoulis *et al.* proposed parallel DE [19] where they assign each subpopulation to a different processor node. Shi *et al.* [20] partitioned high-dimensional search spaces into smaller spaces and used multiple cooperating subpopulations to find the solution. They called this method cooperative coevolutionary differential evolution.

Hybridization with different algorithms is another direction for improvement. Sun *et al.* [21] proposed a new hybrid algorithm based on a combination of DE and estimation of distribution algorithm. This technique uses a probability model to determine promising regions in order to focus the search process on those areas. Noman and Iba [22] incorporated local search

TABLE II
COMPARISON OF DE AND ODE ON FOUR DIFFERENT MUTATION STRATEGIES CONTINUED FROM TABLE I ($f_{31} - f_{58}$). THE BEST RESULT OF EACH MUTATION STRATEGY IS EMPHASIZED IN **BOLDFACE** AND THE BEST RESULT AMONG FOUR MUTATION STRATEGIES (EIGHT RESULTS) IS HIGHLIGHTED IN *ITALIC* FONTS

		DE/rand/1/bin					DE/rand/1/exp					DE/rand/2/exp					DE/rand/2/bin				
		DE		*ODE*			*DE*		*ODE*			*DE*		*ODE*			*DE*		*ODE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_{31}	30	411164	1	**337532**	1	1.22	365492	**1**	*287330*	0.92	1.27	–	0	**653600**	**0.40**	–	–	0	**613900**	**0.24**	–
f_{32}	2	7976	1	**5092**	1	1.56	8004	1	*4856*	1	1.65	15956	1	**6100**	1	2.62	14848	1	**5776**	1	2.57
f_{33}	5	2163	0.88	**2024**	**1**	1.07	2204	0.96	**2068**	**1**	1.07	2136	1	**1944**	1	1.09	2108	1	*1896*	1	1.11
f_{34}	5	38532	1	*16340*	1	2.36	35888	1	**18836**	1	1.90	131756	1	**22968**	1	5.74	145348	1	**23196**	1	6.26
f_{35}	2	2052	1	*1856*	1	1.11	2052	1	**1952**	1	1.05	2528	1	**2236**	1	1.13	2492	1	**2144**	1	1.16
f_{36}	2	8412	1	**5772**	1	1.46	8320	1	*5660*	1	1.47	15904	1	**6680**	1	2.38	16144	1	6408	1	2.52
f_{37}	2	5284	1	**4728**	1	1.12	5320	1	*4668*	1	1.14	6756	1	**5320**	1	1.27	6588	1	**5252**	1	1.25
f_{38}	2	5280	1	**4804**	1	1.10	5260	1	*4676*	1	1.12	6636	1	**5304**	1	1.25	6844	1	**5344**	1	1.28
f_{39}	2	3780	1	**3396**	1	1.11	3736	1	*3288*	1	1.14	4620	1	**3676**	1	1.26	4784	1	**3780**	1	1.27
f_{40}	2	2424	1	**2152**	1	1.13	2388	1	*1984*	1	1.20	2960	1	**2292**	1	1.29	3288	1	**2276**	1	1.44
f_{41}	10	19528	1	**15704**	1	1.24	19452	1	*15452*	1	1.26	41212	1	**22908**	1	1.80	41440	1	**23100**	1	1.80
f_{42}	2	4780	1	**4684**	1	1.02	*4564*	1	4568	1	0.99	6000	1	**5384**	1	1.11	5988	1	**5360**	1	1.12
f_{43}	3	*6852*	1	8484	1	0.81	**6892**	1	7868	1	0.88	**9864**	1	11588	1	0.85	**9592**	1	11480	1	0.84
f_{44}	3	7036	1	*6172*	1	1.14	7092	1	**6196**	1	1.14	9160	1	**7208**	1	1.27	9096	1	**7220**	1	1.26
f_{45}	2	3256	1	*3120*	1	1.04	3292	1	**3168**	1	1.04	4072	1	**3568**	1	1.14	4064	1	**3568**	1	1.14
f_{46}	3	6184	1	*5472*	1	1.13	6100	1	**5536**	1	1.10	8304	1	**6748**	1	1.23	8156	1	**6660**	1	1.22
f_{47}	2	2976	1	*2872*	1	1.03	3016	1	**2924**	1	1.03	3748	1	**3244**	1	1.16	3708	1	**3144**	1	1.18
f_{48}	4	*1108*	1	1232	1	0.90	**1032**	1	1348	1	0.76	**892**	1	1244	1	0.72	**1100**	1	1132	1	0.97
f_{49}	2	*5232*	**1**	5956	0.92	0.88	**5343**	0.92	5988	**1**	0.89	**7344**	1	8504	1	0.86	**7525**	0.96	8496	**1**	0.86
f_{50}	4	379900	**1**	*250260*	0.20	1.52	450960	**1**	**359540**	0.16	1.25	–	0	–	0	–	–	0	–	0	–
f_{51}	10	–	0	*16681*	**0.84**	–	–	0	**17613**	**0.92**	–	–	0	**23385**	**0.80**	–	–	0	**56289**	**0.74**	–
f_{52}	10	*14968*	1	15104	1	0.99	**15028**	1	15140	1	0.99	30468	1	**24504**	1	1.24	30496	1	**24440**	1	1.25
f_{53}	2	7888	1	*4272*	1	1.85	7652	1	**4384**	1	1.75	13280	1	**5192**	1	2.56	12428	1	**5276**	1	2.36
f_{54}	4	8856	1	*7504*	1	1.18	8756	1	**7624**	1	1.15	12520	1	**9196**	1	1.36	12664	1	**9316**	1	1.36
f_{55}	9	*78567*	0.24	97536	**0.56**	0.81	**85360**	0.20	163280	**0.52**	0.52	**301510**	0.48	324204	**0.68**	0.93	299239	0.32	**279310**	**0.76**	1.07
f_{56}	10	37824	1	*24260*	1	1.56	37860	1	**24972**	1	1.52	93652	1	**36384**	1	2.57	90764	1	**38016**	1	2.39
f_{57}	2	**30704**	**1**	55980	0.76	0.55	33428	1	**6178**	**0.92**	5.41	62468	1	**45824**	0.96	1.36	84120	1	*5404*	1	15.56
f_{58}	4	*16600*	1	19144	1	0.87	**16888**	1	18352	1	0.92	30292	1	**29896**	1	1.01	**29824**	1	31168	1	0.96

into the classical DE. They employed fittest individual refinement which is a crossover-based local search. Fan and Lampinen [23] introduced a new local search operation, trigonometric mutation, in order to obtain a better tradeoff between convergence speed and robustness. Kaelo and Ali [24] employed reinforcement learning and different schemes for generating fitter trial points.

Although the proposed algorithm in this paper also attempts to enhance DE, its methodology is completely different from all aforementioned works. That is, a first attempt to accelerate convergence speed of DE by utilizing the scheme of OBL. We use OBL for population initialization and the production of new generations.

III. OPPOSITION-BASED LEARNING (OBL)

Generally speaking, evolutionary optimization methods start with some initial solutions (initial population) and try to improve them toward some optimal solution(s). The process of searching terminates when some predefined criteria are satisfied. In the absence of *a priori* information about the solution, we usually start with *random guesses*. The computation time, among others, is related to the distance of these initial guesses from the optimal solution. We can improve our chance of starting with a closer (fitter) solution by simultaneously checking *the opposite solution*. By doing this, the fitter one (guess or opposite guess) can be chosen as an initial solution. In fact, according to probability theory, 50% of the time a guess is further from the solution than its opposite guess. Therefore, starting with the closer of the two guesses (as judged by its fitness) has the potential to accelerate convergence. The same approach can be applied not only to initial solutions but also continuously to each solution in the current population. However, before concentrating on OBL, we need to define the concept of opposite numbers [7].

Definition (Opposite Number)—Let $x \in [a, b]$ be a real number. The opposite number $\breve{x}$ is defined by

$$\breve{x} = a + b - x. \tag{4}$$

Similarly, this definition can be extended to higher dimensions as follows [7].

Definition (Opposite Point)—Let $P = (x_1, x_2, \ldots, x_D)$ be a point in D-dimensional space, where $x_1, x_2, \ldots, x_D \in R$ and $x_i \in [a_i, b_i]\ \forall i \in \{1, 2, \ldots, D\}$. The opposite point $\breve{P} = (\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_D)$ is completely defined by its components

$$\breve{x}_i = a_i + b_i - x_i. \tag{5}$$

Now, by employing the opposite point definition, the opposition-based optimization can be defined as follows.

Opposition-Based Optimization—Let $P = (x_1, x_2, \ldots, x_D)$ be a point in D-dimensional space (i.e., a candidate solution). Assume $f(\cdot)$ is a fitness function which is used to measure the candidate's fitness. According to the definition of the opposite point, $\breve{P} = (\breve{x}_1, \breve{x}_2, \ldots, \breve{x}_D)$ is the opposite of $P = (x_1, x_2, \ldots, x_D)$. Now, if $f(\breve{P}) \geq f(P)$, then point P can be replaced with $\breve{P}$; otherwise, we continue with P. Hence, the point and its opposite point are evaluated simultaneously in order to continue with the fitter one.

IV. PROPOSED ALGORITHM

Similar to all population-based optimization algorithms, two main steps are distinguishable for DE, namely, population initialization and producing new generations by evolutionary operations such as mutation, crossover, and selection. We will enhance these two steps using the OBL scheme. The original DE is chosen as a parent algorithm and the proposed opposition-based ideas are embedded in DE to accelerate its convergence speed. Corresponding pseudocode for the proposed approach (ODE) is

TABLE III
PSEUDOCODE FOR OPPOSITION-BASED DIFFERENTIAL EVOLUTION (ODE). P_0: INITIAL POPULATION, OP_0: OPPOSITE OF INITIAL POPULATION, N_p: POPULATION SIZE, P: CURRENT POPULATION, OP: OPPOSITE OF CURRENT POPULATION, V: NOISE VECTOR, U: TRIAL VECTOR, D: PROBLEM DIMENSION, $[a_j, b_j]$: RANGE OF THE jTH VARIABLE, BFV: BEST FITNESS VALUE SO FAR, VTR: VALUE TO REACH, NFC: NUMBER OF FUNCTION CALLS, $\mathrm{MAX_{NFC}}$: MAXIMUM NUMBER OF FUNCTION CALLS, F: MUTATION CONSTANT, $\mathrm{rand}(0,1)$: UNIFORMLY GENERATED RANDOM NUMBER, C_r: CROSSOVER RATE, $f(\cdot)$: OBJECTIVE FUNCTION, P': POPULATION OF NEXT GENERATION, J_r: JUMPING RATE, $\min_j^p$: MINIMUM VALUE OF THE jTH VARIABLE IN THE CURRENT POPULATION, $\max_j^p$: MAXIMUM VALUE OF THE jTH VARIABLE IN THE CURRENT POPULATION. STEPS **1–5** AND **26–32** ARE IMPLEMENTATIONS OF OPPOSITION-BASED INITIALIZATION AND OPPOSITION-BASED GENERATION JUMPING, RESPECTIVELY

Opposition-Based Differential Evolution

```
   /* Opposition-Based Population Initialization */
1.    Generate uniformly distributed random population P_0;
2.    for (i = 0 ; i < N_p ; i++)
3.       for (j = 0 ; j < D ; j++)
4.          OP_{0i,j} = a_j + b_j - P_{0i,j};
5.    Select N_p fittest individuals from set the {P_0, OP_0} as initial population P_0;
   /* End of Opposition-Based Population Initialization */

6.       while ( BFV > VTR and NFC < MAX_NFC )
7.       {
8.          for (i = 0 ; i < N_p ; i++ )
9.          {
10.                Select three parents P_{i1}, P_{i2}, and P_{i3} randomly from current population where i ≠ i1 ≠ i2 ≠ i3;
11.                V_i = P_{i1} + F × (P_{i2} - P_{i3});
12.                for (j = 0 ; j < D ; j++ )
13.                {
14.                   if (rand(0,1) < C_r)
15.                      U_{i,j} = V_{i,j};
16.                   else
17.                      U_{i,j} = P_{i,j};
18.                }
19.                Evaluate U_i;
20.                if (f(U_i) ≤ f(P_i))
21.                   P'_i = U_i;
22.                else
23.                   P'_i = P_i;
24.          }
25.                P = P';

              /* Opposition-Based Generation Jumping */
26.                if (rand(0,1) < J_r)
27.                {
28.                   for (i = 0 ; i < N_p ; i++)
29.                      for (j = 0 ; j < D ; j++)
30.                         OP_{i,j} = MIN_j^p + MAX_j^p - P_{i,j};
31.                   Select N_p fittest individuals from set the {P, OP} as current population P;
32.                }
              /* End of Opposition-Based Generation Jumping */

33.       }
```

given in Table III. Newly added/extended code segments will be explained in the following subsections.

A. Opposition-Based Population Initialization

According to our review of optimization literature, random number generation, in absence of *a priori* knowledge, is the common choice to create an initial population. Therefore, as mentioned in Section III, by utilizing OBL, we can obtain fitter starting candidate solutions even when there is no *a priori* knowledge about the solution(s). Steps 1–5 from Table III show the implementation of opposition-based initialization for ODE. The following steps present that procedure.

1) Initialize the population $\mathrm{P}(N_P)$ randomly.
2) Calculate opposite population by

$$\mathrm{OP}_{i,j} = \mathrm{a}_j + \mathrm{b}_j - \mathrm{P}_{i,j}, \quad i = 1, 2, \ldots, N_p; \quad j = 1, 2, \ldots, D \tag{6}$$

where $\mathrm{P}_{i,j}$ and $\mathrm{OP}_{i,j}$ denote jth variable of the ith vector of the population and the opposite-population, respectively.
3) Select the N_p fittest individuals from $\{\mathrm{P} \cup \mathrm{OP}\}$ as initial population.

B. Opposition-Based Generation Jumping

By applying a similar approach to the current population, the evolutionary process can be forced to jump to a new solution candidate, which ideally is fitter than the current one. Based on a jumping rate J_r (i.e., jumping probability), after generating new populations by mutation, crossover, and selection, the opposite population is calculated and the N_p fittest individuals are selected from the union of the current population and the opposite population. Unlike opposition-based initialization, generation jumping calculates the opposite population dynamically. Instead of using variables' predefined interval boundaries

($[a_j, b_j]$), generation jumping calculates the opposite of each variable based on minimum (MIN_j^p) and maximum (MAX_j^p) values of that variable in the current population

$$\mathrm{OP}_{i,j} = \mathrm{MIN}_j^p + \mathrm{MAX}_j^p - \mathrm{P}_{i,j}, \quad i = 1, 2, \ldots, N_p; \qquad j = 1, 2, \ldots, D. \tag{7}$$

By staying within variables' interval static boundaries, we would jump outside of the already shrunken search space and the knowledge of the current reduced space (converged population) would be lost. Hence, we calculate opposite points by using variables' current interval in the population ($[\mathrm{MIN}_j^p, \mathrm{MAX}_j^p]$) which is, as the search does progress, increasingly smaller than the corresponding initial range $[a_j, b_j]$. Steps 26–32 from Table III show the implementation of opposition-based generation jumping for ODE. Discussion about the suitable value for jumping rate J_r is given in Section V-F.

V. Experimental Verification

Test Functions—A comprehensive set of benchmark functions, including 58 different global optimization problems, has been used for performance verification of the proposed approach (although utilizing a much smaller set of benchmark functions for this purpose is commonly acceptable, e.g., [32]). The definition of the benchmark functions and their global optimum(s) are listed in Appendix A.

Comparison Strategies and Metrics—We compare the convergence speed of DE and ODE by measuring the *number of function calls* (NFCs) which is the most commonly used metric in literature [3]–[6], [28], [29], [35]. A smaller NFC means higher convergence speed. The termination criterion is to find a value smaller than the value-to-reach (VTR) before reaching the maximum number of function calls $\mathrm{MAX}_{\mathrm{NFC}}$. In order to minimize the effect of the stochastic nature of the algorithms on the metric, the reported NFCs for each function is the average over 50 trials. In order to compare convergence speeds, we use the *acceleration rate* (AR) which is defined as follows, based on the NFCs for the two algorithms DE and ODE:

$$\mathrm{AR} = \frac{\mathrm{NFC}_{\mathrm{DE}}}{\mathrm{NFC}_{\mathrm{ODE}}} \tag{8}$$

where $\mathrm{AR} > 1$ means ODE is faster.

The number of times, for which the algorithm successfully reaches the VTR for each test function is measured as the *success rate* (SR)

$$\mathrm{SR} = \frac{\text{number of times reached VTR}}{\text{total number of trials}}. \tag{9}$$

Further, the *average acceleration rate* ($\mathrm{AR}_{\mathrm{ave}}$) and the *average success rate* ($\mathrm{SR}_{\mathrm{ave}}$) over n test functions are calculated as follows:

$$\mathrm{AR}_{\mathrm{ave}} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{AR}_i \tag{10}$$

$$\mathrm{SR}_{\mathrm{ave}} = \frac{1}{n}\sum_{i=1}^{n} \mathrm{SR}_i. \tag{11}$$

Parameter Settings—Parameter settings for all conducted experiments are as follows unless a change is mentioned (the same setting has been used in literature cited after of each parameter).

- Population size, $N_p = 100$ [15], [33], [34].
- Differential amplification factor, $F = 0.5$ [2], [4], [13], [15], [18].
- Crossover probability constant, $C_r = 0.9$ [2], [4], [13], [15], [18].
- Jumping rate constant, $J_r = 0.3$ (discussed in Section V-F).
- Mutation strategy [2]: DE/rand/1/bin (classic version of DE) [2], [3], [15], [16], [26].
- Maximum NFCs, $\mathrm{MAX}_{\mathrm{NFC}} = 10^6$.
- Value to reach, VTR$= 10^{-8}$ [35].

In order to maintain a reliable and fair comparison, for all 7 experiments: 1) the parameter settings are the same as above for all experiments, unless we mention new settings for one or some of them to serve the purpose of that parameter study; 2) for all conducted experiments, the reported values are the average of the results for 50 independent runs, and most importantly; 3) extra fitness evaluation is required for the opposite points (both in population initialization and also generation jumping phases) are counted.

Results—A comprehensive set of experiments are conducted and categorized as follows. In Section V-A, DE and ODE are compared in terms of convergence speed and robustness. The effect of problem dimensionality is investigated in Section V-B. The contribution of opposite points to the achieved acceleration results is demonstrated in Section V-C. The effect of population size is studied in Section V-D. Comparison of DE and ODE over some other mutation strategies is performed in Section V-E. Discussion about the newly added control parameter, jumping rate, is covered in Section V-F. Finally, comparison of the ODE with FADE is given in Section V-G.

A. Experiment Series 1: Comparison of DE and ODE

First of all, we need to compare the parent algorithm DE with ODE in terms of convergence speed and robustness. The results of solving 58 benchmark functions (see Appendix A) are given in Table IV. The best result of the NFCs and the SR for each function are highlighted in **boldface**. The average success rates ($\mathrm{SR}_{\mathrm{ave}}$) and the average acceleration rate ($\mathrm{AR}_{\mathrm{ave}}$) on 58 test functions are shown in the last row of the table.

ODE outperforms DE on 40 test functions, while DE surpasses ODE on 15 functions. Except for f_4, the rest of the 14 functions are low-dimensional functions ($D \leq 10$). Average acceleration rate ($\mathrm{AR}_{\mathrm{ave}}$) is 1.44, which means ODE is on average 44% faster than DE. While the average success rate ($\mathrm{SR}_{\mathrm{ave}}$) for both of them is equal to 0.86, both algorithms fail to solve f_{13}, f_{26}, and f_{27}; in addition, DE fails to solve f_{51} and ODE is unsuccessful on f_4. Some sample graphs for the performance comparison between DE and ODE are given in Fig. 1. These curves (best solution versus NFCs) show that how ODE converges faster than DE toward the optimal solution.

Results analysis—With the same control parameter settings for both algorithms and fixing the jumping rate for the ODE ($J_r = 0.3$), their SRs are comparable, while ODE shows better

TABLE IV
COMPARISON OF DE AND ODE. D: DIMENSION, NFC: NUMBER OF FUNCTION CALLS, SR: SUCCESS RATE, AR: ACCELERATION RATE. THE LAST ROW OF THE TABLE PRESENTS THE AVERAGE SUCCESS RATES ($\mathrm{SR_{ave}}$) AND THE AVERAGE ACCELERATION RATE ($\mathrm{AR_{ave}}$). THE BEST RESULTS FOR EACH CASE ARE HIGHLIGHTED IN **BOLDFACE**

		DE		*ODE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	87748	1	**47716**	1	1.83
f_2	30	96488	1	**53304**	1	1.81
f_3	20	177880	1	**168680**	1	1.05
f_4	30	**403112**	**1**	–	0	–
f_5	10	328844	**1**	**70389**	0.76	4.67
f_6	30	113428	**1**	**69342**	0.96	1.64
f_7	30	25140	1	**8328**	1	3.01
f_8	30	169152	1	**98296**	1	1.72
f_9	2	**4324**	1	4776	1	0.90
f_{10}	4	**16600**	1	19144	1	0.87
f_{11}	2	8016	1	**6608**	1	1.21
f_{12}	3	**3376**	1	3580	1	0.94
f_{13}	6	–	0	–	0	–
f_{14}	2	5352	1	**4468**	1	1.20
f_{15}	30	101460	1	**70408**	1	1.44
f_{16}	100	3608	1	**3288**	1	1.09
f_{17}	4	549850	0.04	**311800**	**0.12**	1.76
f_{18}	10	**191340**	**0.76**	213330	0.56	0.90
f_{19}	30	385192	1	**369104**	1	1.04
f_{20}	2	**4884**	1	5748	1	0.85
f_{21}	30	187300	1	**155636**	1	1.20
f_{22}	30	570290	0.28	**72250**	**0.88**	7.89
f_{23}	30	41588	1	**23124**	1	1.80
f_{24}	30	818425	0.12	**199810**	**1**	1.78
f_{25}	4	**13925**	**0.80**	15280	0.60	0.91
f_{26}	4	–	0	–	0	–
f_{27}	4	–	0	–	0	–
f_{28}	4	4576	1	**4500**	1	1.02
f_{29}	2	**10788**	1	11148	1	0.97
f_{30}	2	1016	1	**996**	1	1.02
f_{31}	30	411164	1	**337532**	1	1.22
f_{32}	2	7976	1	**5092**	1	1.56
f_{33}	5	2163	0.88	**2024**	**1**	1.07
f_{34}	5	38532	1	**16340**	1	2.36
f_{35}	2	2052	1	**1856**	1	1.11
f_{36}	2	8412	1	**5772**	1	1.46
f_{37}	2	5284	1	**4728**	1	1.12
f_{38}	2	5280	1	**4804**	1	1.10
f_{39}	2	3780	1	**3396**	1	1.11
f_{40}	2	2424	1	**2152**	1	1.13
f_{41}	10	19528	1	**15704**	1	1.24
f_{42}	2	4780	1	**4684**	1	1.02
f_{43}	3	**6852**	1	8484	1	0.81
f_{44}	3	7036	1	**6172**	1	1.14
f_{45}	2	3256	1	**3120**	1	1.04
f_{46}	3	6184	1	**5472**	1	1.13
f_{47}	2	2976	1	**2872**	1	1.03
f_{48}	4	**1108**	1	1232	1	0.90
f_{49}	2	**5232**	**1**	5956	0.92	0.88
f_{50}	4	379900	**1**	**250260**	0.20	1.52
f_{51}	10	–	0	**16681**	**0.84**	–
f_{52}	10	**14968**	1	15104	1	0.99
f_{53}	2	7888	1	**4272**	1	1.85
f_{54}	4	8856	1	**7504**	1	1.18
f_{55}	9	**78567**	0.24	97536	**0.56**	0.81
f_{56}	10	37824	1	**24260**	1	1.56
f_{57}	2	**30704**	**1**	55980	0.76	0.55
f_{58}	4	**16600**	1	19144	1	0.87
Ave.			*0.86*		*0.86*	**1.44**

convergence speed than DE (44% faster in overall). The jumping rate is an important control parameter which, if optimally set, can achieve even better results. The discussion about this parameter is covered in Section V-F.

B. Experiment Series 2: Influence of Dimensionality

In order to investigate the effect of the problem dimensionality, the same experiments are repeated for $D' = D/2$ and $D' = 2D$ for each *scalable function* from our test set. All other control parameters are kept unchanged. Results for $D/2$ and $2D$ are given in Table V for 40 test functions.

According to the obtained results, ODE surpasses DE on 34 test functions, while DE outperforms ODE on two functions ($f_{4(D=15)}$ and $f_{18(D=5)}$). Both algorithms are unable to solve $f_{4(D=60)}$, $f_{19(D=60)}$, $f_{22(D=60)}$, and $f_{51(D=20)}$ before meeting the maximum NFCs. The average AR is equal to 1.73, meaning that ODE performs 73% faster than DE. The average SR for DE and ODE are 0.82 and 0.81, respectively.

For 11 functions ($f_1, f_2, f_6, f_7, f_8, f_{15}, f_{16}, f_{18}, f_{21}, f_{41}$, and f_{56}), the AR is increased by growing dimensionality. ODE achieves a more desirable results for all but four of the functions (f_3, f_5, f_{23}, and f_{31}) where no improvement can be observed. An interesting effect for f_{18} is that for dimensions 5 and 10 (for $D = 10$, see Table IV), DE performs better than ODE; but when the dimension is increased to 20, ODE shows better results in terms of NFC and SR. Furthermore, DE cannot solve f_{24} for $\mathrm{D} = 60$, but ODE solves it in 35% of the trials.

In the bottom of Table V, the average SRs and the average ARs for functions with $D/2$, $2D$ are presented. For functions with dimension of $D/2$, the overall SR for DE is 4% higher than ODE's (0.98 versus 0.94) and the overall AR is 1.67. For functions with dimension of $2D$, the overall AR is 1.81 and the SRs for DE and ODE are 0.66 and 0.67, respectively.

Results analysis—Decreasing the overall SR for DE and ODE was predictable because by doubling the problem dimension, algorithms are sometimes unable to solve the problem before reaching the maximum NFCs (which is a fixed number for all experiments). However, as seen, ODE performs better for high-dimensional problems. The higher average AR belongs to the functions with dimension $2D$.

C. Experiment Series 3: Contribution of Opposite Points

In this section, we want to verify that the achieved AR is really due to utilizing opposite points. For this purpose, all parts of the proposed algorithm are kept untouched and instead of using opposite points for the population initialization and the generation jumping, uniformly generated random points are employed. In order to have a fair competition for this case, (exactly like what we did for opposite points), the current interval (dynamic interval) of the variables are used to generate new random points in the generation jumping phase. So, line 4 from Table III should be changed to

$$RP_{0i,j} = a_j + (b_j - a_j) \times \mathrm{rand}(0,1)$$

where $\mathrm{rand}(0,1)$ generates a uniformly distributed random number on the interval $(0,1)$. This change is for the initialization part, so the predefined boundaries of variables ($[a_j, b_j]$) have been used to generate new random numbers. In fact, instead of generating N_p random individuals, this time we generate $2N_p$ candidate solutions.

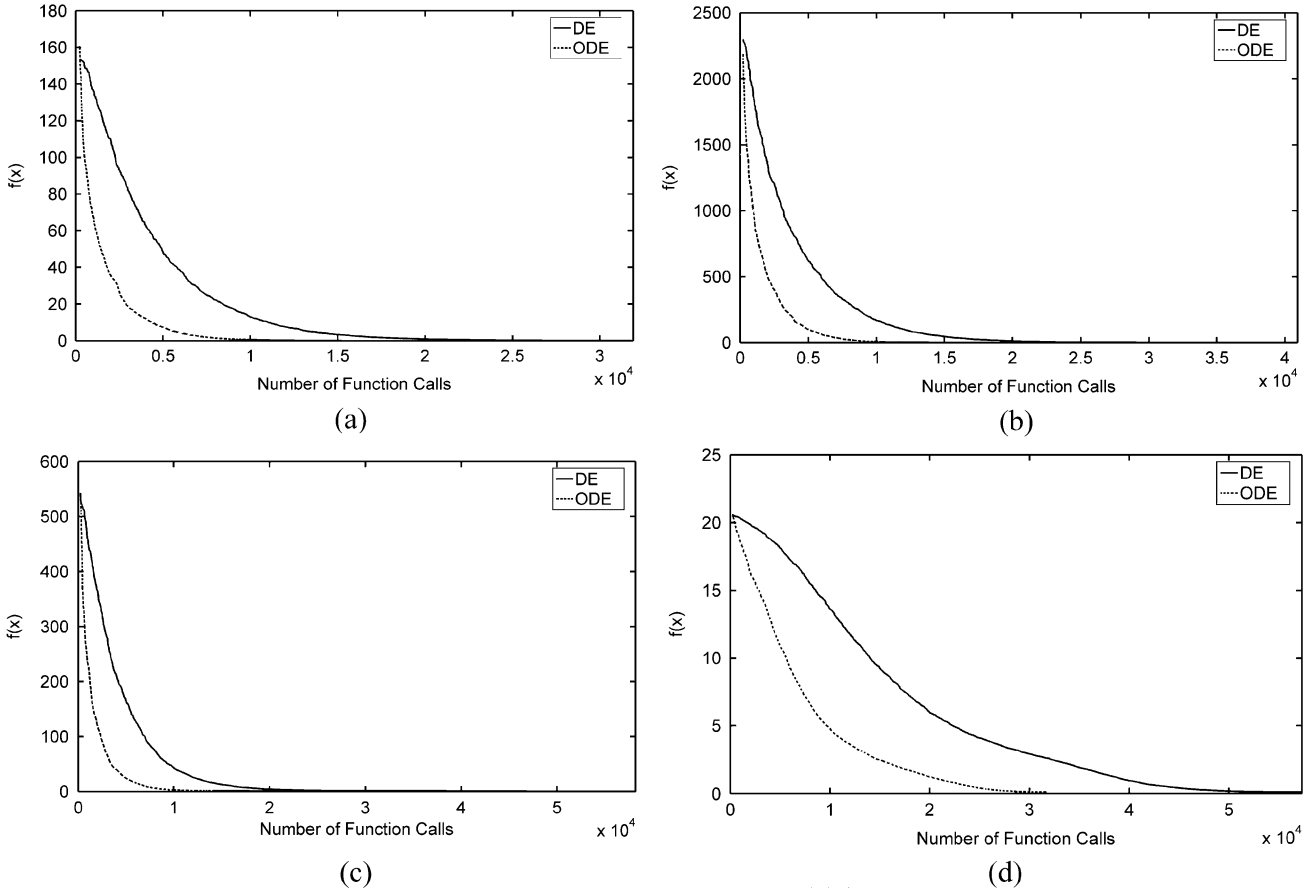

Fig. 1. Sample graphs (best solution versus NFCs) for performance comparison between DE and ODE. (a) f_1, ODE is 1.83 times faster. (b) f_2, ODE is 1.81 times faster. (c) f_6, ODE is 1.64 times faster. (d) f_8, ODE is 1.72 times faster.

In the same manner, line 30 should be replaced with

$$RP_{i,j} = \mathrm{MIN}_j^p + \left(\mathrm{MAX}_j^p - \mathrm{MIN}_j^p\right) \times \mathrm{rand}(0,1).$$

As mentioned before, for generation jumping, the current boundaries of variables ($[\mathrm{MAX}_j^p, \mathrm{MIN}_j^p]$) are used to generate random numbers. Finally, in order to have the same selection method, lines 5 and 31 in Table III are substituted with

$$\text{Select } N_p \text{ fittest individuals from set the } \{P, RP\} \text{ as currentpopulation } P.$$

After these modifications, the random version of ODE (called RDE) is introduced. Now, we are ready to apply this algorithm to solve our test problems. All control parameters are kept the same to ensure a fair comparison. Results for the current algorithm are presented in Table VI; also the results of DE and ODE (from Table IV) are repeated in this table to ease the comparison among these three competitors (DE, ODE, and RDE). Two ARs are reported in this table. The first column compares DE and ODE and the second one compares DE and RDE [$\mathrm{AR_{ODE}} = (\mathrm{NFC_{DE}})/(\mathrm{NFC_{ODE}})$ and $\mathrm{AR_{RDE}} = (\mathrm{NFC_{DE}})/(\mathrm{NFC_{RDE}})$].

As seen, ODE outperforms DE and RDE on 40 of the functions. DE performs better than ODE and RDE on 15 functions. The RDE can outperform DE (but not ODE) on just three functions f_{17}, f_{22}, and f_{35} (emphasized in boldface under the $\mathrm{AR_{RDE}}$ column). The average AR (DE versus RDE) is 0.87, which means the RDE is 13% slower than its parent algorithm. The average SR is almost the same for all of them (0.86, 0.86, and 0.87 for DE, ODE, and RDE, respectively).

Results analysis—Just by replacing the opposite numbers with additional random numbers—while the random numbers are generated uniformly in the variables dynamic intervals and the rest of the proposed algorithm is kept untouched—the average AR drops from 1.44 ($\mathrm{AR_{ODE}}$) to 0.87 ($\mathrm{AR_{RDE}}$), which is a 57% reduction in speed. This clearly demonstrates that the achieved improvements are due to usage of opposite points, and that the same level of improvement cannot be achieved via additional random sampling.

D. Experiment Series 4: Effect of Population Size

In order to investigate the effect of the population size, the same experiments (conducted in Section V-A for $N_p = 100$) are repeated for $N_p' = N_p/2$ and $N_p' = 2N_p$. The results for $N_p = 50$ and $N_p = 200$ are given in Tables XII and XIII, respectively. In order to discuss the population size, the overall results of three tables (Tables IV, XII, and XIII) are summarized in Table VII.

For $N_p = 50$, the average SR for DE and ODE is 0.79 and 0.77, respectively (DE performs marginally better than ODE). However, DE fails to solve nine functions, while ODE fails on seven. ODE outperforms DE on 35 functions; this number is 15 for DE. The average AR is 1.05 for this case ($\mathrm{AR} = 60.16$ for

TABLE V
COMPARISON OF DE AND ODE FOR DIMENSION SIZES $D/2$ AND $2D$ FOR ALL SCALABLE FUNCTIONS OF THE TEST SUITE. IN THE BOTTOM OF THE TABLE, THE AVERAGE SUCCESS RATES AND THE AVERAGE ACCELERATION RATES FOR FUNCTIONS WITH $D/2$, $2D$, AND FOR BOTH (OVERALL) ARE PRESENTED. THE BEST RESULT OF NFC AND SR FOR EACH CASE ARE HIGHLIGHTED IN BOLDFACE

		DE		*ODE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	15	39920	1	**27860**	1	1.43
	60	156552	1	**93720**	1	1.67
f_2	15	43268	1	**30956**	1	1.40
	60	178308	1	**108960**	1	1.64
f_3	10	49188	1	**45284**	1	1.09
	40	835316	1	**795476**	1	1.05
f_4	15	**135452**	**1**	–	0	–
	60	–	0	–	0	–
f_5	5	46964	**1**	**20252**	1	2.32
	20	817000	0.08	**412240**	**0.16**	1.98
f_6	15	108360	**0.96**	**75018**	0.88	1.44
	60	194612	**1**	**127490**	0.72	1.53
f_7	15	13776	1	**7264**	1	1.90
	60	45276	1	**8620**	1	5.25
f_8	15	80520	1	**58300**	1	1.38
	60	295172	1	**188332**	1	1.57
f_{15}	15	45726	1	**39236**	1	1.17
	60	180260	**0.84**	**121750**	0.60	1.48
f_{16}	50	3628	1	**3296**	1	1.10
	200	3680	1	**3264**	1	1.13
f_{18}	5	**18575**	0.96	20067	0.96	0.93
	20	288300	0.35	**253910**	**0.55**	1.14
f_{19}	15	87856	1	**86372**	1	1.02
	60	–	0	–	0	–
f_{21}	15	83324	1	**78916**	1	1.06
	60	330172	1	**246552**	1	1.34
f_{22}	15	129276	1	**51932**	1	2.49
	60	–	0	–	0	–
f_{23}	15	18248	1	**13252**	1	1.38
	60	73756	1	**56708**	1	1.30
f_{24}	15	372000	1	**258650**	1	1.44
	60	–	0	**795575**	**0.35**	–
f_{31}	15	265872	1	**216880**	1	1.23
	60	440320	1	**398956**	1	1.10
f_{41}	5	8060	1	**7220**	1	1.12
	20	46216	1	**27736**	1	1.55
f_{51}	5	45237	0.76	**6776**	**1**	6.68
	20	–	0	–	0	–
f_{56}	5	11524	1	**9848**	1	1.17
	20	179040	1	**55616**	1	3.22
Ave. D/2			**0.98**		*0.94*	*1.67*
Ave. 2D			*0.66*		**0.67**	**1.81**
Ave.			**0.82**		*0.81*	*1.73*

f_{33} is excluded as an exceptional case in order to prevent from dominating the statistic). By carefully looking at the results, we can recognize that when the population size is reduced from 100 to 50, four functions (namely, f_3, f_5, f_{19}, and f_{31}) for which ODE has outperformed DE, are now (for population size 50) solved faster by DE. However, this was predictable because the dimension of those functions are 20, 10, 30, and 30, respectively, and $N_p = 50$ is a small population size to solve these functions. Many authors have proposed $10D$ as a proper value for the population size [13], [14]. On the other hand, as we know, ODE reduces the population diversity by its selection method and so for small population size, we need to reduce the jumping rate to control the diversity reduction. Here, the jumping rate was kept the same for all population sizes and equal to 0.3.

For $N_p = 100$, DE and ODE show an equal average success rate ($\mathrm{SR} = 0.86$), and they fail to solve an equal number of functions ($n_{(\mathrm{SR}=0)} = 4$). However, ODE outperforms DE on 40 functions, whereas DE outperforms ODE on 15 functions. The average AR is 1.44 for this case.

For $N_p = 200$, ODE outperforms DE on all mentioned measures ($\langle 0.86, 6, 39\rangle$ versus $\langle 0.82, 7, 15\rangle$, respectively), and the average AR is 1.86. As shown in the last two rows of Table VII, ODE performs better than DE in terms of all calculated measures.

Results analysis—According to the results of Sections V-A and V-B, for the majority of functions, ODE performs better when the dimension of the problems increases. On the other hand, for higher dimensional problems a larger population size should be employed (e.g., $N_p = 10D$). According to the results of this section, ODE performs better for larger population sizes.

E. Experiments Series 5: Effect of Various Mutation Operators

More than ten mutation strategies have been developed for DE [2], [3]. Although many researchers report results for one of these mutation strategies, most works [2], [15], [16] use the standard one, namely, DE/rand/1/bin, as we did. In this work, three other well-known mutation strategies, namely, DE/rand/1/exp, DE/rand/2/exp, and DE/rand/2/bin are selected to investigate the effect of the mutation operator. The results are presented in Table I (for $f_1 - f_{30}$) and Table II (for $f_{31} - f_{58}$). The overall results of these two tables are summarized in compact form in Table VIII to ease the comparison.

For all mutation strategies, ODE performs better than DE by looking at the total NFCs, average SR, number of solved functions, number of functions for which ODE outperforms DE, and the average AR.

Results analysis—According to Table VIII, the best mutation strategy for DE and also for ODE is the DE/rand/1/bin. This confirms choosing mutation strategy DE/rand/1/bin as a standard operator by other researchers [2], [15], [16].

F. Experiment Series 6: Proper Setting of Jumping Rate J_r

In the proposed algorithm, a new control parameter J_r is added to DE's parameters (F, C_r, and N_p). Although this parameter was fixed for all experiments, the performance of ODE can vary for different J_r values. The jumping rate for our current study was set to $J_r = 0.3$ without any effort to find an optimal value. In some trials, we observed that a jumping rate higher than 0.6 is not suitable for many functions and causes a premature convergence or an unusual growing of the number of function evaluations. On the other hand, $J_r = 0$ means no jumping. By this way, simply the mean value of 0 and 0.6, ($J_r = 0.3$), was selected for all conducted experiments as a default value.

In this section, we try to find an optimal jumping rate ($J_{r\mathrm{op}}$) for each test function from a discrete set of jumping rate values to answer the question whether a general recommendation for J_r setting can be offered. Now, we are faced with a fundamental question: In order to find optimal jumping rate, should we look for the minimum NFCs or the maximum SR? Both measures are important factors in an optimization process. So, two individual objectives should be considered simultaneously. In order

TABLE VI
COMPARISON OF DE, ODE, AND RDE. THE BEST RESULT FOR EACH CASE IS HIGHLIGHTED IN **BOLDFACE**

		DE		*ODE*			*RDE*		
F	D	*NFC*	*SR*	*NFC*	*SR*	AR_{ODE}	*NFC*	*SR*	AR_{RDE}
f_1	30	87748	1	**47716**	1	1.83	115096	1	0.76
f_2	30	96488	1	**53304**	1	1.81	126780	1	0.76
f_3	20	177880	1	**168680**	1	1.05	231152	1	0.77
f_4	30	**403112**	1	–	0	–	506596	1	0.80
f_5	10	328844	**1**	**70389**	0.76	4.67	501875	0.96	0.66
f_6	30	113428	1	**69342**	0.96	1.64	149744	1	0.76
f_7	30	25140	1	**8328**	1	3.01	29096	1	0.86
f_8	30	169152	1	**98296**	1	1.72	222784	1	0.76
f_9	2	**4324**	1	4776	1	0.90	4904	1	0.88
f_{10}	4	**16600**	1	19144	1	0.87	21712	1	0.76
f_{11}	2	8016	1	**6608**	1	1.21	8248	1	0.97
f_{12}	3	**3376**	1	3580	1	0.94	3652	1	0.92
f_{13}	6	–	0	–	0	–	–	0	–
f_{14}	2	5352	1	**4468**	1	1.20	6292	1	0.85
f_{15}	30	101460	1	**70408**	1	1.44	138308	1	0.73
f_{16}	100	3608	1	**3288**	1	1.09	3924	1	0.92
f_{17}	4	549850	0.04	**311800**	0.12	1.76	381840	**0.16**	**1.44**
f_{18}	10	**191340**	**0.76**	213330	0.56	0.90	306900	0.60	0.62
f_{19}	30	385192	1	**369104**	1	1.04	498200	1	0.77
f_{20}	2	**4884**	1	5748	1	0.85	6192	1	0.79
f_{21}	30	187300	1	**155636**	1	1.20	244396	1	0.76
f_{22}	30	570290	0.28	**72250**	0.88	7.89	285108	**1**	**2.00**
f_{23}	30	41588	1	**23124**	1	1.80	54316	1	0.77
f_{24}	30	818425	0.12	**199810**	**1**	1.78	–	0	–
f_{25}	4	**13925**	0.80	15280	0.60	0.91	17038	**0.96**	0.82
f_{26}	4	–	0	–	0	–	–	0	–
f_{27}	4	–	0	–	0	–	–	0	–
f_{28}	4	4576	1	**4500**	1	1.02	5948	1	0.77
f_{29}	2	**10788**	1	11148	1	0.97	11588	1	0.93
f_{30}	2	1016	1	**996**	1	1.02	1084	1	0.94
f_{31}	30	411164	1	**337532**	1	1.22	927230	0.24	0.44
f_{32}	2	7976	1	**5092**	1	1.56	8332	1	0.96
f_{33}	5	2163	0.88	**2024**	**1**	1.07	2904	0.96	0.74
f_{34}	5	38532	1	**16340**	1	2.36	46080	1	0.84
f_{35}	2	2052	1	**1856**	1	1.11	2016	1	**1.02**
f_{36}	2	8412	1	**5772**	1	1.46	10860	1	0.77
f_{37}	2	5284	1	**4728**	1	1.12	5448	1	0.97
f_{38}	2	5280	1	**4804**	1	1.10	5540	1	0.95
f_{39}	2	3780	1	**3396**	1	1.11	3820	1	0.98
f_{40}	2	2424	1	**2152**	1	1.13	2400	1	1.00
f_{41}	10	19528	1	**15704**	1	1.24	23156	1	0.84
f_{42}	2	4780	1	**4684**	1	1.02	4924	1	0.97
f_{43}	3	**6852**	1	8484	1	0.81	8488	1	0.81
f_{44}	3	7036	1	**6172**	1	1.14	7396	1	0.95
f_{45}	2	3256	1	**3120**	1	1.04	3308	1	0.98
f_{46}	3	6184	1	**5472**	1	1.13	6676	1	0.93
f_{47}	2	2976	1	**2872**	1	1.03	3200	1	0.93
f_{48}	4	**1108**	1	1232	1	0.90	1388	1	0.80
f_{49}	2	**5232**	**1**	5956	0.92	0.88	5768	1	0.91
f_{50}	4	379900	**1**	**250260**	0.20	1.52	528632	1	0.72
f_{51}	10	–	0	**16681**	**0.84**	–	–	0	–
f_{52}	10	**14968**	1	15104	1	0.99	18328	1	0.82
f_{53}	2	7888	1	**4272**	1	1.85	9148	1	0.86
f_{54}	4	8856	1	**7504**	1	1.18	10632	1	0.83
f_{55}	9	**78567**	0.24	97536	0.56	0.81	103330	0.56	0.76
f_{56}	10	37824	1	**24260**	1	1.56	46800	1	0.80
f_{57}	2	**30704**	1	55980	0.76	0.55	37696	1	0.81
f_{58}	4	**16600**	1	19144	1	0.87	21344	1	0.78
Ave.			*0.86*		*0.86*	**1.44**		**0.87**	0.87

to combine these two measures, a new measure, called success performance (SP), has been introduced as follows [35]:

$$\mathrm{SP} = \frac{\text{mean (NFC for successful runs)}}{\mathrm{SR}}. \tag{12}$$

By this definition, the two following algorithms have equal performances (SP = 100).

Algorithm A : mean (NFC for successful runs) = 50 and SR = 0.5.

Algorithm B : mean (NFC for successful runs) = 100 and SR = 1.

Now, we repeat the conducted experiments in Section V-A for $J_r \in (0, 0.6]$ with step size of 0.1 (i.e., 50 trials per function per jumping rate value $J_r \in \{0.1, 0.2, 0.3, 0.4, 0.5, 0.6\}$). Due to space limitations, we do not show all the results, only the

TABLE VII
THE SUMMARIZED RESULTS FROM TABLES IV, XII, AND XIII. n_{DE} AND n_{ODE} ARE THE NUMBER OF FUNCTIONS FOR WHICH DE OUTPERFORMS ODE AND *VICE VERSA*. $n_{\mathrm{SR}=0}$ IS THE NUMBER OF UNSOLVED FUNCTIONS BY THE ALGORITHM ($\mathrm{SR} = 0$)

	DE			*ODE*			
N_p	SR_{ave}	$n_{SR=0}$	n_{DE}	SR_{ave}	$n_{SR=0}$	n_{ODE}	AR_{ave}
50	**0.79**	9	16	0.77	**7**	**35**	**1.05**
100	0.86	4	15	0.86	4	**40**	**1.44**
200	0.82	7	15	**0.86**	**6**	**39**	**1.86**
$\sum$		20	46		**17**	**114**	
Ave.	0.82			**0.83**			**1.45**

obtained optimal value for the jumping rate with respect to the success performance as given in Table IX.

As seen, the optimal values for the jumping rate are distributed over the discrete interval $(0, 0.6]$. However, jumping rates of 0.3 and 0.6 are repeated more than other values in this table. Higher jumping rates mostly belong to the low-dimensional functions and lower ones to the high-dimensional functions. The average value of the obtained optimal jumping rates ($\bar{J}_{r\,\mathrm{op}}$) is equal to 0.37 for our test functions.

Some sample graphs (SP versus J_r) are shown in Fig. 2 to illustrate the effect of the jumping rate on success performance. The point specified by $J_r = 0$ indicates the success performance of the DE; the rest of points $(0.1, 0.2, 0.3, 0.4, 0.5, 0.6)$ show the success performance of ODE. As mentioned before, we can observe a sharp increase in the SP for hard functions (e.g., f_5, f_8, f_{22}, f_{31}, and f_{51}) on higher jumping rates. Also, the SP decreases for easy functions by increasing the jumping rate (see f_7, f_{41}, and f_{56}). Almost a smooth behavior for all functions is recognizable for $J_r \in \{0.1, 0.2, 0.3, 0.4\}$ (it was observed even for many functions which their graphs are not presented here). Hence, working in this interval ($[0.1, 0.4]$) could be more reliable for unknown optimization problems.

Results analysis—Like DE's other control parameters, the optimal jumping rate should have a problem-oriented value. Our limited experiments suggest the range of $[0.1, 0.4]$ for an unknown optimization problem. A first attempt can be conducted with $J_r = 0.37$ ($\bar{J}_{r\,\mathrm{op}}$). Furthermore, for high-dimensional problems, a smaller jumping rate is suggested.

G. Experiment Series 7: ODE Versus FADE

The primary purpose of this work is to introduce the notion of opposition into the design and implementation of DE and demonstrate its benefits. Many other extensions of DE, if not all, can also be reconsidered to incorporate the opposition concept. In this sense, ODE should be regarded as an example and not as competitor to other DE versions. However, in order to assess the performance of ODE, a comparison with at least one other algorithm may be beneficial.

We have compared ODE with the FADE method of Liu and Lampinen [13]. They tested FADE for ten well-known benchmark functions, of which we have nine of them in our testbed. The comparison strategy is different for this experiment. The algorithms are run 100 times. Subsequently, for equal (fixed) NFCs, the average and standard deviation of the best solutions are calculated for the purpose of comparison. The same settings for parameters [13] have been used in the current experiment to assure a fair comparison. The population size is equal to $10D$ and instead of using the generation number for DE and ODE an equal number of function evaluations have been used ($N_p \times \#\mathrm{Gen.}$) as the termination criteria (since in each generation NFCs for DE and ODE is not equal). The dimension of the functions, corresponding generation numbers, and obtained results (best mean and standard deviation of 100 runs) for DE, ODE, and FADE are given in Table X. Results for FADE are taken from [13,Table VI, p. 459]. A t-test at a 0.05 significance level has been used to compare results of ODE against those of DE and FADE.

Results analysis—According to the t-test, ODE performs better than DE on 9 functions (out of 16). There is no difference for the rest of functions. ODE surpasses FADE on 12 functions, and they perform the same on the rest. So, DE and FADE cannot show better results than ODE even on one function. Although, the comparison of a nonadaptive algorithm (ODE) to an adaptive one (FADE) is not fair, interestingly, the results confirm that ODE performs outstandingly, it is not only better than DE but also better than FADE. It is important to consider here that ODE performs better than the original DE according to the current comparison strategy as well.

VI. CONCLUSION

In this paper, the concept of OBL has been employed to accelerate DE. OBL was utilized to introduce opposition-based population initialization and opposition-based generation jumping. By embedding these two steps within DE, ODE was proposed.

We can summarize the experimental results as follows.

- DE and ODE were compared for different problem dimensions ($D/2$, D, and $2D$); the results confirm that ODE performs better over high-dimensional problems. For these kind of problems, a large population size is required. On the other hand, ODE performs well with larger population sizes. These two facts support each other and make ODE more suitable for higher dimensional problems. Further study is required to solidify this expectation.
- By replacing opposite points with uniformly generated random points in the same variables' range, the AR was reduced by 57%. Therefore, the contribution of opposite points to the acceleration process was confirmed and was not reproducible by additional random sampling.
- DE and ODE were compared for different population sizes ($N_p/2$, N_p, and $2N_p$). ODE performed better for larger population sizes, which is usually required for more complex problems. In order to achieve better results for smaller population sizes, small jumping rates are suggested.
- Comparison of DE and ODE over various mutation strategies was also conducted. For all cases, ODE performed the same or better than DE with respect to the NFCs, average SR, and other performance measures.
- The influence of the jumping rate was studied and the range $[0.1, 0.4]$ was recommended for an unknown optimization problem. Most of the functions presented a reliable acceleration improvement and almost a smooth behavior in this interval. Although, the optimal jumping rate can be somewhere out of this range, higher jumping rates are generally not recommended.

TABLE VIII
THE SUMMARIZED RESULTS FROM TABLES I AND II. $\sum \mathrm{NFC}_i$ IS THE SUMMATION OF THE NUMBER OF FUNCTION CALLS (JUST FOR THE FUNCTIONS WHICH ALL EIGHT COMPETITORS COULD SOLVE). $n_{\mathrm{SR}=0}$ IS THE NUMBER OF UNSOLVED FUNCTIONS ($\mathrm{SR} = 0$). n_{DE} AND n_{ODE} ARE THE NUMBER OF FUNCTIONS FOR WHICH DE OUTPERFORMS ODE AND *VICE VERSA*. N IS THE NUMBER OF FUNCTIONS FOR WHICH THE ALGORITHM COULD OUTPERFORMS OTHER ALGORITHMS. $\mathrm{AR}_{\mathrm{ave}}$ IS THE AVERAGE ACCELERATION RATE

	DE					*ODE*					
Mutation Strategy	$\sum NFC_i$	SR_{ave}	$n_{SR=0}$	n_{DE}	N	$\sum NFC_i$	SR_{ave}	$n_{SR=0}$	n_{ODE}	N	AR_{ave}
$DE/rand/1/bin$	853,979	0.86	4	15	7	**656,274**	0.86	4	**40**	**14**	1.42
$DE/rand/1/exp$	857,672	**0.86**	**5**	17	8	**743,584**	0.85	6	**37**	**16**	1.51
$DE/rand/2/exp$	4,865,187	0.71	15	7	0	**1,207,478**	**0.83**	**7**	**44**	**7**	2.78
$DE/rand/2/bin$	4,070,189	0.69	17	8	1	**1,119,580**	**0.81**	**9**	**41**	**2**	3.03

TABLE IX
OPTIMAL JUMPING RATE J_{rop} FOR ALL TEST FUNCTIONS WITH RESPECT TO THE SUCCESS PERFORMANCE (SP) ON INTERVAL $(0, 0.6]$ WITH STEP SIZE OF 0.1

F	D	J_{rop}	F	D	J_{rop}	F	D	J_{rop}	F	D	J_{rop}
f_1	30	0.3	f_{16}	100	0.5	f_{31}	30	0.1	f_{46}	3	0.6
f_2	30	0.3	f_{17}	4	0.3	f_{32}	2	0.4	f_{47}	2	0.6
f_3	20	0.3	f_{18}	10	0.2	f_{33}	5	0.6	f_{48}	4	0.6
f_4	30	0.1	f_{19}	30	0.2	f_{34}	5	0.6	f_{49}	2	0.1
f_5	10	0.2	f_{20}	2	0.1	f_{35}	2	0.5	f_{50}	4	0.1
f_6	30	0.3	f_{21}	30	0.5	f_{36}	2	0.6	f_{51}	10	0.2
f_7	30	0.6	f_{22}	30	0.2	f_{37}	2	0.6	f_{52}	10	0.3
f_8	30	0.3	f_{23}	30	0.3	f_{38}	2	0.6	f_{53}	2	0.6
f_9	2	0.1	f_{24}	30	0.3	f_{39}	2	0.5	f_{54}	4	0.6
f_{10}	4	0.2	f_{25}	4	0.3	f_{40}	2	0.5	f_{55}	9	0.6
f_{11}	2	0.4	f_{26}	4	–	f_{41}	10	0.6	f_{56}	10	0.6
f_{12}	3	0.1	f_{27}	4	–	f_{42}	2	0.4	f_{57}	2	0.5
f_{13}	6	–	f_{28}	4	0.3	f_{43}	3	0.1	f_{58}	4	0.2
f_{14}	2	0.5	f_{29}	2	0.1	f_{44}	3	0.5			
f_{15}	30	0.3	f_{30}	2	0.6	f_{45}	2	0.4			

The main motivation for the current work was utilizing the notion of opposition to accelerate DE. In order to have a comparison with other methods (other than the original DE), ODE was also compared with FADE. The results clearly confirmed that ODE performs better than FADE in terms of convergence rate and solution accuracy on the utilized 58 benchmark functions.

Utilizing opposite numbers to accelerate an optimization method is a new concept. Further studies are still required to investigate its benefits, weaknesses, and limitations. This work can be considered as a first step in this direction. The main claim is not defeating DE or any of its numerous versions but to introduce a new notion into optimization via metaheuristics—the notion of opposition.

Possible directions for future work include the adaptive setting of the jumping rate, proposing other possibilities to implement ODE (e.g., opposition-based mutation strategies), and applying the same or similar scheme to accelerate other population-based methods (e.g., GA and PSO).

APPENDIX A
LIST OF BENCHMARK FUNCTIONS

- Sphere model
$$f_1(x) = \sum_{i=1}^{n} {x_i}^2, \quad \text{with } -5.12 \le x_i \le 5.12$$
$$\min(f_1) = f_1(0, \ldots, 0) = 0.$$
- Axis parallel hyperellipsoid
$$f_2(x) = \sum_{i=1}^{n} i{x_i}^2, \quad \text{with } -5.12 \le x_i \le 5.12$$
$$\min(f_2) = f_2(0, \ldots, 0) = 0.$$
- Schwefel's problem 1.2
$$f_3(x) = \sum_{i=1}^{n} \left(\sum_{j=1}^{i} x_j \right)^2, \quad \text{with } -65 \le x_i \le 65$$
$$\min(f_3) = f_3(0, \ldots, 0) = 0.$$
- Rosenbrock's valley
$$f_4(x) = \sum_{i=1}^{n-1} [100(x_{i+1} - x_i^2)^2 + (1 - x_i)^2], \quad \text{with } -2 \le x_i \le 2$$
$$\min(f_4) = f_4(1, \ldots, 1) = 0.$$
- Rastrigin's function
$$f_5(x) = 10n + \sum_{i=1}^{n} \left(x_i^2 - 10\cos(2\pi x_i) \right), \quad \text{with } -5.12 \le x_i \le 5.12$$
$$\min(f_5) = f_5(0, \ldots, 0) = 0.$$
- Griewangk's function
$$f_6(x) = \sum_{i=1}^{n} \frac{x_i^2}{4000} - \prod_{i=1}^{n} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1, \quad \text{with } -600 \le x_i \le 600$$
$$\min(f_6) = f_6(0, \ldots, 0) = 0.$$
- Sum of different power
$$f_7(x) = \sum_{i=1}^{n} |x_i|^{(i+1)}, \quad \text{with } -1 \le x_i \le 1$$
$$\min(f_7) = f_7(0, \ldots, 0) = 0.$$
- Ackley's path function
$$f_8(x) = -20\exp\left(-0.2\sqrt{\frac{\sum_{i=1}^{n} x_i^2}{n}} \right) - \exp\left(\frac{\sum_{i=1}^{n} \cos(2\pi x_i)}{n} \right) + 20 + e, \quad \text{with } -32 \le x_i \le 32$$
$$\min(f_8) = f_8(0, \ldots, 0) = 0.$$
- Beale function
$$f_9(x) = [1.5 - x_1(1 - x_2)]^2 + [2.25 - x_1(1 - x_2^2)]^2 + \left[2.625 - x_1\left(1 - x_2^3\right)\right]^2, \quad \text{with } -4.5 \le x_i \le 4.5$$
$$\min(f_9) = f_9(3, 0.5) = 0.$$
- Colville function
$$f_{10}(x) = 100\left(x_2 - x_1^2\right)^2 + (1 - x_1)^2 + 90\left(x_4 - x_3^2\right)^2 + (1 - x_3)^2 + 10.1((x_2 - 1)^2 + (x_4 - 1)^2) + 19.8(x_2 - 1)(x_4 - 1), \quad \text{with } -10 \le x_i \le 10$$
$$\min(f_{10}) = f_{10}(1, 1, 1, 1) = 0.$$

TABLE X
COMPARISON OF DE, ODE, AND FUZZY ADAPTIVE DE (FADE). MEAN BEST AND STANDARD DEVIATION (STD DEV) OF 100 RUNS ARE REPORTED. FOR THE DE AND ODE, THE EQUAL NUMBER OF FUNCTION CALLS ARE USED INSTEAD OF GENERATION NUMBERS ($N_p \times \#\mathrm{Gen.}$). T-TEST IS USED TO COMPARE ODE AGAINST DE AND FADE. "†" INDICATES THE t -VALUE OF 99 DEGREE OF FREEDOM IS SIGNIFICANT AT A 0.05 LEVEL OF SIGNIFICANCE BY TWO-TAILED t-TEST. $f_{\min}$ INDICATES OPTIMAL MINIMUM OF THE FUNCTION

				DE	**ODE**	**FADE**
F	f_{min}	D	# *Gen.*	Mean Best (Std Dev)	Mean Best (Std Dev)	Mean Best (Std Dev)
f_1	0	3	50	3.49×10^{-7} $(5.12 \times 10^{-7})^{\dagger}$	4.79×10^{-8} (5.81×10^{-8})	1.18×10^{-5} $(1.88 \times 10^{-10})^{\dagger}$
		50	5000	1.25×10^{-13} $(4.88 \times 10^{-14})^{\dagger}$	1.73×10^{-64} (1.59×10^{-64})	2.35×10^{-10} $(2.97 \times 10^{-21})^{\dagger}$
f_4	0	2	50	8.0×10^{-3} $(2.93 \times \times 10^{-2})$	$4.8 \times \times 10^{-3}$ $(1.80 \times \times 10^{-2})$	2.2×10^{-3} (7.73×10^{-5})
		50	7000	5.30 $(9.56 \times 10^{-1})^{\dagger}$	5.01×10^{-1} (8.90×10^{-1})	$4.16 \times 10^{+1}$ $(1.82 \times 10^{-2})^{\dagger}$
f_5	0	2	3000	0 (0)	0 (0)	0 (0)
		50	10000	$3.43 \times 10^{+2}$ $(0.98 \times 10^{+1})^{\dagger}$	$6.56 \times 10^{+1}$ $(6.80 \times 10^{+1})$	$2.58 \times 10^{+2}$ $(9.17 \times 10^{+1})^{\dagger}$
f_6	0	2	180	5.19×10^{-4} (1.9×10^{-3})	1.49×10^{-4} (1.0×10^{-3})	2.4×10^{-3} $(1.20 \times 10^{-5})^{\dagger}$
		50	5000	6.13×10^{-11} $(2.02 \times 10^{-11})^{\dagger}$	1.53×10^{-62} (2.38×10^{-62})	5.78×10^{-1} $(3.5 \times 10^{-3})^{\dagger}$
f_8	0	3	50	1.07×10^{-2} $(7.4 \times 10^{-3})^{\dagger}$	4.8×10^{-3} (3.7×10^{-3})	1.56×10^{-1} $(1.20 \times 10^{-2})^{\dagger}$
		50	5000	2.33×10^{-6} $(2.33 \times 10^{-6})^{\dagger}$	7.02×10^{-15} (1.66×10^{-15})	5.9×10^{-2} $(1.23 \times 10^{-6})^{\dagger}$
f_{11}	-1	2	80	-1 (0)	-1 (0)	-0.9896 $(1.3 \times 10^{-3})^{\dagger}$
f_{23}	0	5	30	0 (0)	0 (0)	0 (0)
		50	5000	0 (0)	0 (0)	0 (0)
f_{24}	0	30	5000	5.1×10^{-3} $(9.69 \times 10^{-4})^{\dagger}$	8.70×10^{-4} (3.30×10^{-4})	9.16 $(1.50 \times 10^{-1})^{\dagger}$
		50	5000	1.64×10^{-2} $(2.4 \times 10^{-3})^{\dagger}$	9.04×10^{-4} (3.41×10^{-4})	$1.90 \times 10^{+1}$ $(2.92 \times 10^{-1})^{\dagger}$
f_{42}	3	2	50	3 (0)	3 (0)	3.0001 $(3.35 \times 10^{-7})^{\dagger}$

- Easom function

$$f_{11}(x) = -\cos(x_1)\cos(x_2)\exp((-(x_1-\pi)^2 - (x_2-\pi)^2)), \quad \text{with} -100 \le x_i \le 100$$
$$\min(f_{11}) = f_{11}(\pi,\pi) = -1.$$

- Hartmann function 1

$$f_{12}(x) = -\sum_{i=1}^{4} \alpha_i \exp(-\sum_{j=1}^{3} A_{ij}(x_j - P_{ij})^2), \quad \text{with } 0 \le x_i \le 1$$
$$\min(f_{12}) = f_{12}(0.114614, 0.555649, 0.852547) = -3.86278$$

where

$$\alpha = [1 \quad 1.2 \quad 3 \quad 3.2]$$
$$A = \begin{bmatrix} 3 & 10 & 30 \\ 0.1 & 10 & 35 \\ 3 & 10 & 30 \\ 0.1 & 10 & 35 \end{bmatrix}$$
$$P = \begin{bmatrix} 0.36890 & 0.11700 & 0.26730 \\ 0.46990 & 0.43870 & 0.74700 \\ 0.10910 & 0.87320 & 0.55470 \\ 0.03815 & 0.57430 & 0.88280 \end{bmatrix}.$$

- Hartmann function 2

$$f_{13}(x) = -\sum_{i=1}^{4} \alpha_i \exp\left(-\sum_{j=1}^{6} B_{ij}(x_j - Q_{ij})^2\right), \quad \text{with } 0 \le x_i \le 1$$
$$\min(f_{13}) = f_{13}(0.20169, 0.150011, 0.476874, 0.275332, 0.311652, 0.6573) = -3.32237$$

where

$$\alpha = [1 \quad 1.2 \quad 3 \quad 3.2]$$
$$B = \begin{bmatrix} 10 & 3 & 17 & 3.05 & 1.7 & 8 \\ 0.05 & 10 & 17 & 0.1 & 8 & 14 \\ 3 & 3.5 & 1.7 & 10 & 17 & 8 \\ 17 & 8 & 0.05 & 10 & 0.1 & 14 \end{bmatrix}$$
$$Q = \begin{bmatrix} 0.1312 & 0.1696 & 0.5569 & 0.0124 & 0.8283 & 0.5886 \\ 0.2329 & 0.4135 & 0.8307 & 0.3736 & 0.1004 & 0.9991 \\ 0.2348 & 0.1451 & 0.3522 & 0.2883 & 0.3047 & 0.6650 \\ 0.4047 & 0.8828 & 0.8732 & 0.5743 & 0.1091 & 0.0381 \end{bmatrix}$$

- Six Hump Camel back function

$$f_{14}(x) = 4x_1^2 - 2.1x_1^4 + \frac{1}{3}x_1^6 + x_1x_2 - 4x_2^2 + 4x_2^4, \quad \text{with} -5 \le x_i \le 5$$
$$\min(f_{14}) = f_{14}(0.0898, -0.7126)/(-0.0898, 0.7126) = 0.$$

- Levy function

$$f_{15}(x) = \sin^2(3\pi x_1) + \sum_{i=1}^{n-1}(x_i - 1)^2 \times (1 + \sin^2(3\pi x_{i+1})) + (x_n - 1)(1 + \sin^2(2\pi x_n)), \quad \text{with} -10 \le x_i \le 10$$
$$\min(f_{15}) = f_{15}(1, \ldots, 1) = 0.$$

- Matyas function

$$f_{16}(x) = 0.26\left(x_1^2 + x_2^2\right) - 0.48x_1x_2, \quad \text{with} -10 \le x_i \le 10$$
$$\min(f_{16}) = f_{16}(0,0) = 0.$$

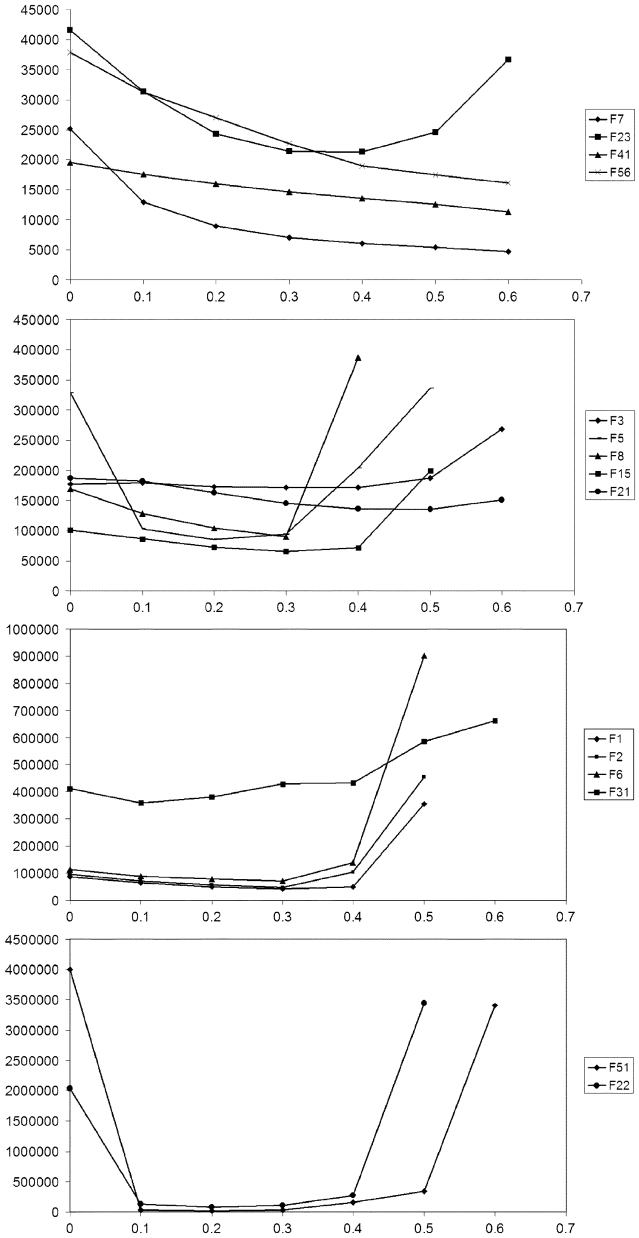

Fig. 2. Graphs of success performance (SP) versus jumping rate ($J_r \in (0, 0.6]$ with step size of 0.1) for some sample functions. The point declared by $J_r = 0$ shows the SP of the DE; the rest of points $(0.1, 0.2, 0.3, 0.4, 0.5, 0.6)$ show the SP of the ODE.

- Perm function

$$f_{17}(x) = \sum_{k=1}^{n} \left[\sum_{i=1}^{n} (i^k + 0.5) \left(\left(\frac{1}{i} x_i \right)^k - 1 \right) \right]^2, \quad \text{with } -n \le x_i \le n$$
$$\min(f_{17}) = f_{17}(1, 2, 3, \ldots, n) = 0.$$

- Michalewicz function

$$f_{18}(x) = -\sum_{i=1}^{n} \sin(x_i)(\sin(ix_i^2/\pi))^{2m}, \quad \text{with } 0 \le x_i \le \pi, m = 10$$
$$\min(f_{18_{(n=2)}}) = -1.8013,$$
$$\min(f_{18_{(n=5)}}) = -4.687658,$$
$$\min(f_{18_{(n=10)}}) = -9.66015.$$

TABLE XI
NAME OF PROBLEMS FOR f_{35}-f_{58}

F	*Name of the Problem*
f_{35}	Aluffi-Pentini's Problem
f_{36}	Becker and Lago Problem
f_{37}	Bohachevsky 1 Problem
f_{38}	Bohachevsky 2 Problem
f_{39}	Camel Back-3 Three Hump Problem
f_{40}	Dekkers and Aarts Problem
f_{41}	Exponential Problem
f_{42}	Goldstein and Price Problem
f_{43}	Gulf Research Problem
f_{44}	Helical Valley Problem
f_{45}	Hosaki Problem
f_{46}	Levy and Montalvo 1 Problem
f_{47}	McCormick problem
f_{48}	Miele and Cantrell Problem
f_{49}	Multi-Gaussian Problem
f_{50}	Neumaier 2 Problem
f_{51}	Odd Square Problem
f_{52}	Paviani's Problem
f_{53}	Periodic Problem
f_{54}	Powell's Quadratic Problem
f_{55}	Price's Transistor Modeling Problem
f_{56}	Salomon Problem
f_{57}	Schaffer 2 Problem
f_{58}	Wood's Function

- Zakharov function

$$f_{19}(x) = \sum_{i=1}^{n} x_i^2 + \left(\sum_{i=1}^{n} 0.5ix_i \right)^2 + \left(\sum_{i=1}^{n} 0.5ix_i \right)^4, \quad \text{with } -5 \le x_i \le 10$$
$$\min(f_{19}) = f_{19}(0, \ldots, 0) = 0.$$

- Branins's function

$$f_{20}(x) = a\left(x_2 - bx_1^2 + cx_1 - d\right)^2 + e(1 - f)\cos(x_1) + e, \quad \text{with } -5 \le x_1 \le 10, \quad 0 \le x_2 \le 15$$
$$\text{where } a = 1, b = 5.1/(4\pi^2), c = 5/\pi, d = 6, e = 10, f = 1/(8\pi)$$
$$\min(f_{20}) = f_{20}(-\pi, 12.275)/(-\pi, 2.275)/(9.42478, 2.475) = 0.3979.$$

- Schwefel's problem 2.22

$$f_{21}(x) = \sum_{i=1}^{n} |x_i| + \prod_{i=1}^{n} |x_i|, \quad \text{with } -10 \le x_1 \le 10$$
$$\min(f_{21}) = f_{21}(0, \ldots, 0) = 0.$$

- Schwefel's problem 2.21

$$f_{22}(x) = \max_i \{|x_i|, 1 \le i \le n\}, \quad \text{with } -100 \le x_1 \le 100$$
$$\min(f_{22}) = f_{22}(0, \ldots, 0) = 0.$$

- Step function

$$f_{23}(x) = \sum_{i=1}^{n} (\lfloor x_i + 0.5 \rfloor)^2, \quad \text{with } -100 \le x_1 \le 100$$
$$\min(f_{23}) = f_{23}(-0.5 \le x_i < 0.5) = 0.$$

TABLE XII
COMPARISON OF DE AND ODE ($N_p = 50$)

		DE		*ODE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	32548	1	**25293**	0.92	1.29
f_2	30	36126	1	**28889**	0.92	1.25
f_3	20	**81630**	1	88272	1	0.92
f_4	30	–	0	–	0	–
f_5	10	**94627**	**0.60**	170200	0.24	0.57
f_6	30	42543	0.80	**36426**	0.76	1.17
f_7	30	8984	1	**3600**	1	2.50
f_8	30	63180	1	**52930**	0.80	1.19
f_9	2	**2248**	1	2378	1	0.95
f_{10}	4	**8456**	**0.96**	54136	0.84	0.16
f_{11}	2	3924	1	**3378**	1	1.16
f_{12}	3	1856	1	**1796**	1	1.03
f_{13}	6	–	0	–	0	–
f_{14}	2	2588	1	**2366**	1	1.09
f_{15}	30	37065	0.80	**32513**	**0.84**	1.14
f_{16}	100	1826	1	1782	1	1.02
f_{17}	4	–	0	–	0	–
f_{18}	10	**217870**	0.20	219790	0.20	0.99
f_{19}	30	**162700**	1	185882	1	0.88
f_{20}	2	**2614**	1	2804	1	0.93
f_{21}	30	64960	1	**63734**	1	1.02
f_{22}	30	–	0	–	0	–
f_{23}	30	15688	1	**12658**	1	1.24
f_{24}	30	–	0	–	0	–
f_{25}	4	**8866**	**0.72**	77539	0.52	0.11
f_{26}	4	–	0	–	0	–
f_{27}	4	–	0	–	0	–
f_{28}	4	2472	0.96	**2316**	1	1.06
f_{29}	2	**10420**	1	10884	1	0.96
f_{30}	2	628	1	**522**	1	1.20
f_{31}	30	**85136**	1	94082	0.88	0.90
f_{32}	2	3806	1	**2830**	1	1.34
f_{33}	5	64009	0.60	**1064**	1	60.16
f_{34}	5	16400	0.96	**11282**	1	1.45
f_{35}	2	1046	1	**1036**	1	1.00
f_{36}	2	3920	1	**3074**	1	1.28
f_{37}	2	2664	1	**2480**	1	1.07
f_{38}	2	2786	1	**2444**	1	1.14
f_{39}	2	1956	1	**1776**	1	1.10
f_{40}	2	1275	1	**1116**	1	1.14
f_{41}	10	9246	1	**7554**	1	1.22
f_{42}	2	2408	1	**2370**	1	1.02
f_{43}	3	**3512**	1	4018	1	0.87
f_{44}	3	3534	1	**3180**	1	1.11
f_{45}	2	1734	1	**1654**	1	1.05
f_{46}	3	3162	1	**2916**	1	1.08
f_{47}	2	1618	1	**1528**	1	1.06
f_{48}	4	**564**	1	776	1	0.73
f_{49}	2	**2809**	0.84	3308	**0.88**	0.91
f_{50}	4	–	0	**364300**	**0.08**	–
f_{51}	10	–	0	**136290**	**0.28**	–
f_{52}	10	**7160**	1	7262	1	0.99
f_{53}	2	4230	1	**2580**	1	1.64
f_{54}	4	4462	1	**3998**	1	1.12
f_{55}	9	271620	**0.40**	**182956**	0.36	1.48
f_{56}	10	18874	1	**12714**	1	1.48
f_{57}	2	**14550**	**0.92**	69472	0.60	0.21
f_{58}	4	**8456**	**0.96**	54136	0.84	0.16
Ave.			*0.79*		*0.77*	*1.05*

TABLE XIII
COMPARISON OF DE AND ODE ($N_p = 200$)

		DE		*ODE*		
F	*D*	*NFC*	*SR*	*NFC*	*SR*	*AR*
f_1	30	234912	1	**103968**	1	2.26
f_2	30	259096	1	**116360**	1	2.23
f_3	20	488544	1	**412808**	1	1.18
f_4	30	**809424**	**1**	–	0	–
f_5	10	983900	0.04	**149800**	**1**	6.57
f_6	30	307880	**1**	**155800**	0.88	1.98
f_7	30	71864	1	**17144**	1	4.19
f_8	30	457848	1	**211768**	1	2.16
f_9	2	**8200**	1	9200	1	0.89
f_{10}	4	**34400**	1	37688	1	0.91
f_{11}	2	15800	1	**12496**	1	1.26
f_{12}	3	**6632**	1	7104	1	0.93
f_{13}	6	–	0	–	0	–
f_{14}	2	10264	1	**7648**	1	1.34
f_{15}	30	280280	1	**173000**	1	1.62
f_{16}	100	6832	1	**6304**	1	1.08
f_{17}	4	**513930**	**0.08**	–	0	–
f_{18}	10	498090	0.84	**448140**	**0.88**	1.11
f_{19}	30	–	0	**927440**	**0.84**	–
f_{20}	2	**9824**	1	10368	1	0.95
f_{21}	30	526280	1	**368048**	1	1.43
f_{22}	30	987800	0.04	**144368**	**1**	6.84
f_{23}	30	111792	1	**48304**	1	2.31
f_{24}	30	–	0	–	0	–
f_{25}	4	**25896**	0.92	30757	0.92	0.84
f_{26}	4	–	0	–	0	–
f_{27}	4	–	0	–	0	–
f_{28}	4	8856	1	**8512**	1	1.04
f_{29}	2	**20768**	1	22008	1	0.94
f_{30}	2	1808	1	**1792**	1	1.01
f_{31}	30	–	0	811075	**0.28**	–
f_{32}	2	15784	1	**9152**	1	1.72
f_{33}	5	4192	1	**3832**	1	1.09
f_{34}	5	80864	1	**27568**	1	2.93
f_{35}	2	3824	1	**3728**	1	1.03
f_{36}	2	17320	1	**10784**	1	1.60
f_{37}	2	10272	1	**9032**	1	1.14
f_{38}	2	10336	1	**9024**	1	1.15
f_{39}	2	7104	1	**6512**	1	1.09
f_{40}	2	4320	1	**3488**	1	1.24
f_{41}	10	39976	1	**31448**	1	1.27
f_{42}	2	8920	1	**8712**	1	1.02
f_{43}	3	**13664**	1	16384	1	0.83
f_{44}	3	13256	1	**12080**	1	1.10
f_{45}	2	6376	1	**6056**	1	1.05
f_{46}	3	11776	1	**10656**	1	1.11
f_{47}	2	5872	1	**5424**	1	1.08
f_{48}	4	**1904**	1	2424	1	0.79
f_{49}	2	**10184**	1	11840	1	0.86
f_{50}	4	**747848**	**1**	788810	0.88	0.95
f_{51}	10	–	0	**27220**	**1**	–
f_{52}	10	**30304**	1	31056	1	0.98
f_{53}	2	15216	1	**8392**	1	1.81
f_{54}	4	17088	1	**14760**	1	1.16
f_{55}	9	**174620**	0.36	272170	**0.44**	0.64
f_{56}	10	71760	1	**49088**	1	1.46
f_{57}	2	49816	1	**6896**	1	7.22
f_{58}	4	**33720**	1	37240	1	0.91
Ave.			*0.82*		**0.86**	**1.68**

- Quartic function, i.e., noise

$$f_{24}(x) = \sum_{i=1}^{n} i x_i^4 + \text{random}[0,1),$$

$$\text{with } -1.28 \le x_1 \le 1.28$$

$$\min(f_{24}) = f_{24}(0,\ldots,0) = 0.$$

- Kowalik's function

$$f_{25}(x) = \sum_{i=1}^{11} \left[a_i - \frac{x_1\left(b_i^2 + b_i x_2\right)}{b_i^2 + b_i x_3 + x_4} \right]^2,$$

$$\text{with } -5 \le x_i \le 5$$

$$\min(f_{25}) = f_{25}(0.19, 0.19, 0.12, 0.14) = 0.0003075$$

where a and b^{-1} is shown in the equation at the top of the next page.

$$a = [0.1957 \quad 0.1947 \quad 0.1735 \quad 0.1600 \quad 0.0844 \quad 0.0627 \quad 0.0456 \quad 0.0342 \quad 0.0323 \quad 0.0235 \quad 0.0246]$$
$$b^{-1} = [0.25 \quad 0.5 \quad 1 \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16]$$

- Shekel's Family

$$f(x) = -\sum_{i=1}^{m} [(x_i - a_i)(x_i - a_i)^T + c_i]^{-1},$$
$$\text{with } m = 5, 7, \text{and } 10 \text{ for } f_{26}(x), f_{27}(x), \text{ and } f_{28}(x), \text{respectively}, 0 \le x_j \le 10,$$
$$\min(f_{26}) = f_{26}(4,4,4,4) = -10.2,$$
$$\min(f_{27}) = f_{27}(4,4,4,4) = -10.4,$$
$$\min(f_{28}) = f_{28}(4,4,4,4) = -10.5$$

where

$$a = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix}, \quad c = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{bmatrix}.$$

- Tripod function

$$f_{29}(x) = p(x_2)(1 + p(x_1)) + |(x_1 + 50p(x_2)(1 - 2p(x_1)))| + |(x_2 + 50(1 - 2p(x_2)))|,$$
$$\text{with } -100 \le x_i \le 100,$$
$$\min(f_{29}) = f_{29}(0, -50) = 0$$

where $p(x) = 1$ for $x \ge 0$, otherwise, $p(x) = 0$.

- De Jong's function 4 (no noise)

$$f_{30}(x) = \sum_{i=1}^{n} i x_i{}^4, \quad \text{with } -1.28 \le x_i \le 1.28$$
$$\min(f_{30}) = f_{30}(0, \ldots, 0) = 0.$$

- Alpine function

$$f_{31}(x) = \sum_{i=1}^{n} |x_i \sin(x_i) + 0.1x_i|, \quad \text{with} -10 \le x_i \le 10$$
$$\min(f_{31}) = f_{31}(0, \ldots, 0) = 0.$$

- Schaffer's function 6

$$f_{32}(x) = 0.5 + \frac{\sin^2 \sqrt{(x_1^2 + x_2^2)} - 0.5}{1 + 0.01 (x_1^2 + x_2^2)^2},$$
$$\text{with } -10 \le x_i \le 10$$
$$\min(f_{32}) = f_{32}(0,0) = 0.$$

- Pathological function

$$f_{33}(x) = \sum_{i=1}^{n-1} \left(0.5 + \frac{\sin^2 \sqrt{(100x_i^2 + x_{i+1}^2)} - 0.5}{1 + 0.001 (x_i^2 - 2x_i x_{i+1} + x_{i+1}^2)^2} \right),$$
$$\text{with } -100 \le x_i \le 100$$
$$\min(f_{33}) = f_{33}(0, \ldots, 0) = 0.$$

- Inverted cosine wave function (Masters)

$$f_{34}(x) = -\sum_{i=1}^{n-1} \left(\exp\left(\frac{-(x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1})}{8} \right) \times \cos\left(4\sqrt{x_i^2 + x_{i+1}^2 + 0.5x_i x_{i+1}} \right) \right),$$
$$\text{with } -5 \le x_i \le 5$$
$$\min(f_{34}) = f_{34}(0, \ldots, 0) = -n + 1.$$

The names of functions $f_{35} - f_{58}$ are listed in Table XI. For more information about these functions, see [31].

Appendix B
Complementary Results

The results for $N_p = 50$ and $N_p = 200$ are given in Tables XII and XIII, respectively.

Acknowledgment

The authors would like to thank Prof. X. Yao and three anonymous referees for their detailed and constructive comments that helped us to increase the quality of this work.

References

[1] R. Storn and K. Price, "Differential evolution—A simple and efficient adaptive scheme for global optimization over continuous spaces," Berkeley, CA, Tech. Rep. TR-95-012, 1995.

[2] ——, "Differential evolution—A simple and efficient heuristic for global optimization over continuous spaces," in *Journal of Global Optimization 11*. Norwell, MA: Kluwer, 1997, pp. 341–359.

[3] K. Price, R. M. Storn, and J. A. Lampinen, *Differential Evolution: A Practical Approach to Global Optimization (Natural Computing Series)*, 1st ed. New York: Springer, 2005, ISBN: 3540209506.

[4] J. Vesterstroem and R. Thomsen, "A comparative study of differential evolution, particle swarm optimization, and evolutionary algorithms on numerical benchmark problems," *Proc. Congr. Evol. Comput.*, vol. 2, pp. 1980–1987, 2004.

[5] J. Andre, P. Siarry, and T. Dognon, "An improvement of the standard genetic algorithm fighting premature convergence in continuous optimization," *Advance in Engineering Software 32*, pp. 49–60, 2001.

[6] O. Hrstka and A. Kučerová, "Improvement of real coded genetic algorithm based on differential operators preventing premature convergence," *Advance in Engineering Software 35*, pp. 237–246, 2004.

[7] H. R. Tizhoosh, "Opposition-based learning: A new scheme for machine intelligence," in *Proc. Int. Conf. Comput. Intell. Modeling Control and Autom.*, Vienna, Austria, 2005, vol. I, pp. 695–701.

[8] ——, "Reinforcement learning based on actions and opposite actions," in *Proc. ICGST Int. Conf. Artif. Intell. Mach. Learn.*, Cairo, Egypt, 2005.

[9] ——, "Opposition-based reinforcement learning," *J. Advanced Comput. Intell. Intelligent Inform.*, vol. 10, no. 3, pp. 578–585, 2006.

[10] M. Shokri, H. R. Tizhoosh, and M. Kamel, "Opposition-based Q(λ) algorithm," in *Proc. IEEE World Congr. Comput. Intell.*, Vancouver, BC, Canada, 2006, pp. 646–653.

[11] M. Ventresca and H. R. Tizhoosh, "Improving the convergence of backpropagation by opposite transfer functions," in *Proc. EEE World Congr. Comput. Intell.*, Vancouver, BC, Canada, 2006, pp. 9527–9534.

[12] S. Rahnamayan, H. R. Tizhoosh, and M. M. A Salama, "Opposition versus randomness in soft computing techniques," *Elsevier J. Applied Soft Comput.*, Aug. 2006, submitted for publication.

[13] J. Liu and J. Lampinen, "A fuzzy adaptive differential evolution algorithm," *Soft Computing-A Fusion of Foundations, Methodologies and Applications*, vol. 9, no. 6, pp. 448–462, 2005.
[14] R. Storn, "On the usage of differential evolution for function optimization," in *Proc. Biennial Conf. North Amer. Fuzzy Inf. Process. Soc.*, 1996, pp. 519–523.
[15] 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, no. 6, pp. 646–657, Dec. 2006.
[16] J. Sun, Q. Zhang, and E. P. K. Tsang, "DE/EDA: A new evolutionary algorithm for global optimization," *Inf. Sci.*, vol. 169, pp. 249–262, 2005.
[17] 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.
[18] M. M. Ali and A. Törn, "Population set-based global optimization algorithms: Some modifications and numerical studies," *Comput. Oper. Res.*, vol. 31, no. 10, pp. 1703–1725, 2004.
[19] D. K. Tasoulis, N. G. Pavlidis, V. P. Plagianakos, and M. N. Vrahatis, "Parallel differential evolution." [Online]. Available: http://citeseer.ist.psu.edu/661975.html
[20] Y.-J. Shi, H.-F. Teng, and Z.-Q. Li, "Cooperative Co-evolutionary differential evolution for function optimization," in *Proc. 1st Int. Conf. Advances in Natural Comput.*, Changsha, China, Aug. 27–29, 2005, pp. 1080–1088.
[21] J. Sun, Q. Zhang, and E. Tsang, "DE/EDA: A new evolutionary algorithm for global optimization," *Inf. Sci.*, vol. 169, pp. 249–262, 2004.
[22] N. Noman and H. Iba, "Enhancing differential evolution performance with local search for high dimensional function optimization," in *Proc. Conf. Genetic Evol. Comput.*, Washington, DC, 2005, pp. 967–974.
[23] H.-Y. Fan and J. Lampinen, "A trigonometric mutation operation to differential evolution," *J. Global Optim.*, vol. 27, no. 1, pp. 105–129, 2003.
[24] P. Kaelo and M. M. Ali, "Probabilistic adaptation of point generation schemes in some global optimization algorithms," *J. Optim. Methods Softw.*, vol. 21, no. 3, pp. 343–357, 2006.
[25] K. Price, *An Introduction to Differential Evolution*, D. Corne, M. Dorigo, and F. Glover, Eds. London, U.K.: McGraw-Hill, 1999, pp. 79–108, ISBN:007-709506-5, New Ideas in Optimization.
[26] G. C. Onwubolu and B. V. Babu, *New Optimization Techniques in Engineering*. New York: Springer, 2004.
[27] S. Das, A. Konar, and U. Chakraborty, "Improved differential evolution algorithms for handling noisy optimization problems," in *Proc. IEEE Congr. Evol. Comput.*, 2005, vol. 2, pp. 1691–1698.
[28] S. Rahnamayan, H. R. Tizhoosh, and M. M. A. Salama, "Opposition-based differential evolution algorithms," in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Canada, 2006, pp. 7363–7370, (held as a part of IEEE World Congr. Comput. Intell.).
[29] ——, "Opposition-based differential evolution for optimization of noisy problems," in *Proc. IEEE Congr. Evol. Comput.*, Vancouver, BC, Canada, 2006, pp. 6756–6763, (held as a part of IEEE World Congr. Comput. Intell..
[30] V. K. Koumousis and C. P. Katsaras, "A sawtooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 19–28, Feb. 2006.
[31] M. M. Ali, C. Khompatraporn, and Z. B. Zabinsky, "A numerical evaluation of several stochastic algorithms on selected continuous global optimization test problems," *J. Global Optim.*, vol. 31, pp. 635–672, 2005.
[32] V. K. Koumousis and C. P. Katsaras, "A sawtooth genetic algorithm combining the effects of variable population size and reinitialization to enhance performance," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 19–28, 2006.
[33] X. Yao, Y. Liu, and G. Lin, "Evolutionary programming made faster," *IEEE Trans. Evol. Comput.*, vol. 3, no. 2, p. 82, July 1999.
[34] C. Y. Lee and X. Yao, "Evolutionary programming using mutations based on the Lévy probability distribution," *IEEE Trans. Evol. Comput.*, vol. 8, no. 1, pp. 1–13, Feb. 2004.
[35] P. N. Suganthan, N. Hansen, J. J. Liang, K. Deb, Y.-P. Chen, A. Auger, and S. Tiwari, "Problem definitions and evaluation criteria for the CEC 2005 special session on real-parameter optimization," Nanyang Tech. Univ., Singapore and KanGAL, Kanpur Genetic Algorithms Lab., IIT, Kanpur, India, Tech. Rep., Rep. No. 2005005, May 2005.

Shahryar Rahnamayan (S'06–M'06) received the B.Sc. and M.S. degrees both (with Honors) in software engineering from the Shahid Beheshti University, Tehran, Iran. He is currently working towards the Ph.D. degree in the field of evolutionary algorithms at the University of Waterloo, Waterloo, ON, Canada.

His current research interests are evolutionary algorithms, machine intelligence, and computer vision. During the Ph.D. program, he has published 15 conference papers and 4 journals.

Mr. Rahnamayan is a member of the Association for Computing Machinery (ACM). He was awarded the Ontario Graduate Scholarship (OGS), the President's Graduate Scholarship (PGS), and the Canadian Institute of Health Research (CIHR) Fellowship.

Hamid R. Tizhoosh received the M.S. degree in electrical engineering from the University of Technology, Aachen, Germany, in 1995 and the Ph.D. degree in computer vision from the University of Magdeburg, Magdeburg, Germany, in 2000.

From 1993 to 1996, he worked at Management of Intelligent Technologies, Ltd. (M.I.T. GmbH), Aachen, Germany, in the area of industrial image processing. He was active as the Scientist in the Engineering Department of Photon Dynamics, Markham, Canada, until 2001. He also conducted research in the Knowledge/Intelligence Systems Laboratory, University of Toronto, ON, Canada, in 2001. Since September 2001, he has been a Faculty Member at the Department of Systems Design Engineering, University of Waterloo, Waterloo, ON, Canada. His research encompasses machine intelligence, computer vision, and opposition-based learning.

Magdy M. A. Salama (S'75–M'77–SM'98–F'02) received the B.Sc. and M.S. degrees in electrical engineering from Cairo University, Cairo, Egypt, in 1971 and 1973, respectively, and the Ph.D. degree in electrical engineering from the University of Waterloo, Waterloo, ON, Canada, in 1977.

Currently, he is a Professor in the Electrical and Computer Engineering Department, University of Waterloo. His research interests include the operation and control of electric distribution systems and biomedical engineering applications. He has consulted widely with government agencies and the electrical authority.

Dr. Salama is a registered Professional Engineer in the Province of Ontario.