Proceedings of the 2007 IEEE Swarm Intelligence Symposium (SIS 2007)

Differential Evolution Based Particle Swarm Optimization

Mahamed G.H. Omran
Department of Computer Science
Gulf University of Science and Technology
Kuwait
mjomran@gmail.com

Andries P. Engelbrecht
Department of Computer Science
University of Pretoria
South Africa
engel@cs.up.ac.za

Ayed Salman
Computer Engineering Department
Kuwait University
Kuwait
ayed@eng.kuniv.edu.kw

***Abstract*— A new, almost parameter-free optimization algorithm is developed in this paper as a hybrid of the barebones particle swarm optimizer (PSO) and differential evolution (DE). The DE is used to mutate, for each particle, the attractor associated with that particle, defined as a weighted average of its personal and neighborhood best positions. Results of this algorithm are compared to that of the barebones PSO, Von Neumann PSO, a DE PSO, and DE/rand/1/bin. These results show that the new algorithm provides excellent results with the added advantage that no parameter tuning is needed.**

I. Introduction

Particle swarm optimization (PSO) [4], [10] and differential evolution (DE) [21] are two stochastic, population-based optimization methods, which have been applied successfully to a wide range of problems as summarized in [5], [11], [16]. It is, however, the case that the performance of these methods are greatly influenced by their control parameters. Empirical and theoretical studies have shown that the convergence behavior of PSO is strongly dependent on the values of the inertia weight and the acceleration coefficients [3], [24], [25]. Wrong choices of values for these parameters may result in divergent or cyclic particle trajectories. The performance of DE, on the other hand, is influenced mainly by the scale parameter and the probability of recombination. Although recommendations for values of these parameters have been made in the literature (based on empirical studies) [22], these values are not universally applicable. The best values for DE control parameters remain problem dependent, and need to be fine tuned for each problem.

A number of variations of both PSO and DE have been developed in the past decade to improve the performance of these algorithms [5], [11], [16]. One class of variations includes hybrids between PSO and DE, where the advantages of the two approaches are combined. This paper presents another PSO-DE hybrid algortihm, which combines concepts from the barebones PSO [9] and the recombination operator of DE. The resulting algorithm, referred to as the barebones DE (BBDE) eliminates the control parameters of PSO and replaces the static DE control parameters with dynamically changing parameters to produce an almost parameter-free optimization algorithm.

The rest of the paper is organized as follows: Section II provides a compact overview of PSO and variants used in this paper, while Section III summarizes DE. The new barebones DE is presented in Section V. Results are provided in Section VI.

II. Particle Swarm Optimization

Particle swarm optimization (PSO) [4], [10] is a stochastic, population-based search method, modeled after the behavior of bird flocks. A PSO algorithm maintains a swarm of individuals (called particles), where each individual (particle) represents a candidate solution. Particles follow a very simple behavior: emulate the success of neighboring particles, and own successes achieved. The position of a particle is therefore influenced by the best particle in a neighborhood, as well as the best solution found by the particle. Particle position, $\mathbf{x}_i$, are adjusted using

$$\mathbf{x}_i(t+1) = \mathbf{x}_i(t) + \mathbf{v}_i(t+1) \tag{1}$$

where the velocity component, $\mathbf{v}_i$, represents the step size. For the basic PSO,

$$\begin{aligned} v_{ij}(t+1) &= wv_{ij}(t) + c_1 r_{1j}(t)(y_{ij}(t) - x_{ij}(t)) \\ &\quad c_2 r_{2j}(t)(\hat{y}_{ij}(t) - x_{ij}(t)) \end{aligned} \tag{2}$$

where w is the inertia weight [19], c_1 and c_2 are the acceleration coefficients, $r_{1j}, r_{2j} \sim U(0,1)$, $\mathbf{y}_i$ is the personal best position of particle i, and $\hat{\mathbf{y}}_i$ is the neighborhood best position of particle i.

The neighborhood best position $\hat{\mathbf{y}}_i$, of particle i depends on the neighborhood topology used [8], [12]. If a star topology is used, then $\hat{\mathbf{y}}_i$ refers to the best position found by the entire swarm. That is,

$$\begin{aligned} \hat{\mathbf{y}}_i &\in \{\mathbf{y}_0(t)), \mathbf{y}_1(t), \ldots, \mathbf{y}_s(t)\} \\ &= \min\{f(\mathbf{y}_0(t)), f(\mathbf{y}_1(t)), \ldots, f(\mathbf{y}_s(t))\} \end{aligned} \tag{3}$$

where s is the swarm size. The resulting algorithm is referred to as the global best (gbest) PSO. For the ring topology, the swarm is divided into overlapping neighborhoods of particles. In this case, $\hat{\mathbf{y}}_i$ is the best position found by the neighborhood of particle i. The resulting algorithm is referred to as the local best (lbest) PSO. The Von Neumann topology defines neighborhoods by organizing particles in a lattice structure. A number of empirical studies have shown that the Von Neumann topology outperforms other neighborhood topologies [12], [15]. It is important to note that neighborhoods are determined using particle indices, and are not based on any spatial information.

A large number of PSO variations have been developed, mainly to improve the accuracy of solutions, diversity, and convergence behavior [5], [11]. This section reviews those

1-4244-0708-7/07/$20.00 ©2007 IEEE

variations used in this study, from which concepts have been borrowed to develop a new, parameter-free PSO algorithm.

Van den Bergh and Engelbrecht [24], [25], and Clerc and Kennedy [3], formally proved that each particle converges to a weighted average of its personal best and neighborhood best positions. That is,

$$\lim_{t \to +\infty} x_{ij}(t) = \frac{c_1 y_{ij} + c_2 \hat{y}_{ij}}{c_1 + c_2} \tag{4}$$

This theoretically derived behavior provides support for the barebones PSO developed by Kennedy [9], where the velocity vector is replaced with a vector of random numbers sampled from a Gaussian distribution with the mean defined by equation (4), assuming that $c_1 = c_2$, and deviation,

$$\sigma = |y_{ij}(t) - \hat{y}_{ij}(t)| \tag{5}$$

The velocity equation changes to

$$v_{ij}(t+1) \sim N\left(\frac{y_{ij}(t) + \hat{y}_{ij}(t)}{2}, \sigma\right) \tag{6}$$

The position update then changes to

$$\mathbf{x}_i(t+1) = \mathbf{v}_i(t+1) \tag{7}$$

Kennedy [9] also proposed an alternative version of the barebones PSO, where

$$v_{ij}(t+1) = \begin{cases} y_{ij}(t) & \text{if } U(0,1) < 0.5 \\ N(\frac{y_{ij}(t)+\hat{y}_{ij}(t)}{2}, \sigma) & \text{otherwise} \end{cases} \tag{8}$$

Based on equation (8), there is a 50% chance that the j-th dimension of the particle dimension changes to the corresponding personal best position. This version of the barebones PSO biases towards exploiting personal best positions.

III. Differential Evolution

Differential evolution (DE) is an evolutionary algorithm proposed by Storn and Price [16], [21]. While DE shares similarities with other evolutionary algorithms (EA), it differs significantly in the sense that distance and direction information from the current population is used to guide the search process. In DE, a target vector is mutated using a difference vector (obtained as a weighted difference between randomly selected individuals) to obtain a trial vector. This trial vector is then stochastically recombined with the parent to produce an offspring, which is only accepted if the fitness of the offspring is better than that of the parent. A number of DE strategies have been proposed, differing in the way that the target vector is selected, the number of difference vectors used, and the way in which recombination is applied [20], [22]. This paper uses the DE/rand/1/bin strategy, described as follows.

For each parent, $\mathbf{x}_i(t)$, select $i_1, i_2, i_3 \sim U(1, \ldots, s)$, where s is the population size, and $i_1 \neq i_2 \neq i_3 \neq i$. A trial vector, $\mathbf{u}_i(t)$, is created as

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta(\mathbf{x}_{i_2}(t) - \mathbf{x}_{i_3}(t)) \tag{9}$$

where $\beta \in (0, \infty)$ is the scale factor, controlling the amplification of the differential variation.

The DE cross-over operator implements a discrete recombination of the trial vector, $\mathbf{u}_i(t)$, and the parent vector, $\mathbf{x}_i(t)$, to produce offspring, $\mathbf{x}_i^{'}(t)$. Cross-over is implemented as follows:

$$x_{ij}^{'}(t) = \begin{cases} u_{ij}(t) & \text{if } j \in \mathcal{J} \\ x_{ij}(t) & \text{otherwise} \end{cases} \tag{10}$$

where $x_{ij}(t)$ refers to the j-th element of the vector $\mathbf{x}_i(t)$, and $\mathcal{J}$ is the set of element indices that will undergo perturbation (i.e. the set of cross-over points). For binomial cross-over, cross-over points are randomly selected from the set of possible cross-over points, $\{1, 2, \ldots, n_d\}$, where n_d is the problem dimension. Algorithm 1 summarizes this process. In this algorithm, p_r is the probability that the considered cross-over point will be included.

$j^* \sim U(1, n_d)$;
$\mathcal{J} \leftarrow \mathcal{J} \cup \{j^*\}$;
for *each* $j \in \{1, \ldots, n_d\}$ **do**
  if $U(0,1) < p_r$ *and* $j \neq j^*$ **then**
    $\mathcal{J} \leftarrow \mathcal{J} \cup \{j\}$;
  end
end

Algorithm 1: Differential Evolution Binomial Cross-over for Selecting Cross-over Points

Although empirical studies have shown that DE convergence is relatively insensitive to control parameter values, performance can be greatly improved if parameter values are optimized. A number of DE strategies have been developed where values for control parameters adapt dynamically. Abbass and Sarker [2] proposed an approach where a new value is sampled for the scale factor for each application of the mutation operator. The scale factor is sampled from a Gaussian distribution, $\beta \sim N(0,1)$. This approach is also used in [18], [17]. In [17], the mean of the distribution was changed to 0.5 and the deviation to 0.3 (i.e. $\beta \sim N(0.5, 0.3)$), due to the empirical results that suggest that $\beta = 0.5$ provides on average good results. Abbass [1] extends this to the probability of recombination, i.e. $p_r \sim N(0,1)$.

For the self-adaptive Pareto DE, Abbass [1] adapted the probability of recombination dynamically as

$$p_{r,i}(t) = p_{r,i_1}(t) + N(0,1)[p_{r,i_2}(t) - p_{r,i_3}(t)] \tag{11}$$

where $i_1 \neq i_2 \neq i_3 \neq i \sim U(1, \ldots, n_s)$, while sampling the scale factor from $N(0,1)$. Note that equation (11) implies that each individual has its onw, learned probability of recombination. Omran *et al.* [13] proposed a self-adaptive DE strategy which makes use of the approach in equation (11) to dynamically adapt the scale factor. That is, for each individual,

$$\beta_i(t) = \beta_{i_4}(t) + N(0, 0.5)[\beta_{i_5}(t) - \beta_{i_6}(t)] \tag{12}$$

where $i_4 \neg i_5 \neq i_6 \neq i \sim U(1, \ldots, n_s)$. The mutation operator as given in equation (9) changes to

$$\mathbf{u}_i(t) = \mathbf{x}_{i_1}(t) + \beta_i(t)[\mathbf{x}_{i_2}(t) + \mathbf{x}_{i_3}(t)]$$

The cross-over probability can be sampled from a Gaussian distribution as discussed above, or adapted according to equation (11).

IV. PSO AND DE HYBRIDS

Hendtlass [6] used the DE perturbation approach in equation (10) and Algorithm 1 to adapt particle positions. Particle positions are updated only if their offspring have better fitnesses. That is, $\mathbf{x}_i(t+1) = \mathbf{x}_i^{'}(t+1)$ only if $f(\mathbf{x}_i^{'}(t+1)) < f(\mathbf{x}_i(t))$, otherwise $\mathbf{x}_i(t+1) = \mathbf{x}_i(t)$ (assuming a minimization problem). The DE reproduction process is applied to the particles in a PSO swarm at specified intervals. At the specified intervals, the PSO swarm serves as the population for a DE algorithm, and the DE is executed for a number of generations. After execution of the DE, the evolved population is then further optimized using PSO. Kannan *et al.* [7] apply DE to each particle for a number of iterations, and replaces the particle with the best individual obtained from the DE process.

Zhang and Xie [26], and Talbi and Batouche [23] follow a somewhat different approach. Only the personal best positions are changed using

$$y_{ij}^{'}(t+1) = \begin{cases} \hat{y}_{ij}(t) + \delta_j & \text{if } j \in \mathcal{J}_i(t) \\ y_{ij}(t) & \text{otherwise} \end{cases},$$

where δ is the general difference vector defined as

$$\delta_j = \frac{y_{1j}(t) - y_{2y}(t)}{2}$$

with $\mathbf{y}_1(t)$ and $\mathbf{y}_2(t)$ randomly selected personal best positions; the notations $\mathbf{y}_i(t)$ and $\hat{\mathbf{y}}_i(t)$ are used to indicate a personal best and neighborhood best respectively. The offspring, $\mathbf{y}_i^{'}(t+1)$, replaces the current personal best, $\mathbf{y}_i(t)$, only of the offspring has a better fitness.

Due to the sensitivity of DE performance on the values of control parameters, this paper proposes a variation of the DEPSO of Hendtlass, to use adaptive parameters similar to [14], [13]. The position update is changed to

$$\mathbf{x}_i(t) = \begin{cases} \mathbf{x}_i^{PSO} & \text{if } U(0,1) \leq N(0.8, 0.1) \\ \mathbf{x}_i^{DE} & \text{otherwise} \end{cases} \quad (13)$$

where $\mathbf{x}_i^{PSO}$ refers to using the PSO position update equation, while for $\mathbf{x}_i^{DE}$, the new position of particle i is calculated as follows: if $U(0,1) \leq N(0.5, 0.3)$, then

$$x_{ij}^{DE}(t) = x_{i_1 j}(t) + N(0.7, 0.3) \times (x_{i_2 j}(t) - x_{i_3 j}(t)) \quad (14)$$

otherwise $x_{ij}^{DE}(t) = x_{ij}(t)$.

V. BAREBONES DIFFERENTIAL EVOLUTION

This paper combines the barebones PSO with the DE to produce a new, parameter-free optimization algorithm. For the barebones DE, position updates are done as follows: if $U(0,1) > p_r$, then

$$x_{ij}(t) = p_{ij}(t) + r_{2j}(t) \times (x_{i_1 j}(t) - x_{i_2 j}(t)) \quad (15)$$

otherwise $x_{ij}(t) = y_{i_3 j}(t)$, where from equation (4),

$$p_{ij}(t) = r_{1j}(t) y_{ij}(t) + (1 - r_{1j}(t)) \hat{y}_{ij} \quad (16)$$

with $i_1, i_2, i_3 \sim U(1, \ldots, s)$, $i_1 \neq i_2 \neq i_3 \neq i$, $r_{1j}, r_{2j} \sim U(0,1)$, and p_r is the probability of recombination.

Using the position update in equation (16), for a proportion of $(1 - p_r)$ of the updates, information from a randomly selected personal best, $\mathbf{y}_{i_3}$, is used (facilitating exploitation), while for a proportion of p_r of the updates step sizes are mutations of the attractor point, $\mathbf{p}_i$ (facilitating exploration). Mutation step sizes are based on the difference vector between randomly selected particles, $\mathbf{x}_{i_1}$ and $\mathbf{x}_{i_2}$. Using the above, the BBDE achieves a good balance between exploration and exploitation. It should also be noted that the exploitation of personal best positions does not focus on a specific position. The personal best position, $\mathbf{y}_{i_3}$, is randomly selected for each application of the position update.

Even though the barebones DE still has one parameter, p_r, empirical results presented in Section VI show that performance is insensitive to its value for the optimization problems considered. Therefore the reference to this algorithm as being almost parameter free.

VI. EXPERIMENTAL RESULTS

This section compares the performance of the barebones DE (BBDE) with that of the two barebones PSO algorithms discussed in Section II, a PSO using the Von Neumann neighborhood topology, the differential evolution PSO (DEPSO) described in Section III, and the DE/rand/1/bin strategy. For the PSO algorithms, $w = 0.72$ and $c_1 = c_2 = 1.49$. These values have been shown to provide very good results [3], [24], [25]. For the DE, $\beta = 0.5$ and $p_r = 0.9$, as suggested in [22].

The following functions have been used to test the performance of the different algorithms:

Ackley: For $x_i \in [-30, 30]$, and $n_d = 30$,

$$\begin{aligned} f(\mathbf{x}) &= -20\text{exp}\left(-0.2\sqrt{\frac{1}{n_d}\sum_{i=1}^{n_d} x_i^2}\right) \\ &\quad -exp\left(\frac{1}{n_d}\sum_{i=1}^{n_d}\cos(2\pi x_i)\right) + 20 + e \end{aligned}$$

Shekel's foxholes: For $x_i \in [-65.536, 65.536]$, and $n_d = 2$,

$$f(\mathbf{x}) = \left[\frac{1}{500} + \sum_{j=1}^{25}\left(\frac{1}{j + \sum_{i=1}^{2}(x_i - a_{ij})^6}\right)\right]^{-1}$$

where (a_{ij}) is defined as

$$\begin{pmatrix} -32 & -16 & 0 & 16 & 32 & -32 & \ldots & 0 & 16 & 32 \\ -32 & -32 & -32 & -32 & -32 & -16 & \ldots & 32 & 32 & 32 \end{pmatrix}$$

Griewank: For $x_i \in [-600, 600]$, and $n_d = 30$,

$$f(\mathbf{x}) = \frac{1}{4000}\sum_{i=1}^{n_d} x_i^2 - \prod_{i=1}^{n_d}\cos(\frac{x_i}{\sqrt{i}}) + 1$$

Himmelblau: For $x_i \in [-6, 6]$, and $n_d = 2$,

$$f(\mathbf{x}) = 200 - (x_1^2 + x_2 - 11)^2 - (x_1 + x_2^2 - 7)^2$$

Hyperellipsoid: For $x_i \in [-5.12, 5.12]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d} i x_i^2$$

Michalewicz 12: For $x_i \in [0, \pi]$, and $n_d = 30$,

$$f(\mathbf{x}) = -\sum_{i=1}^{n_d} \sin(x_i) \left[\sin(i x_i^2)/\pi\right]^{20}$$

Quadric: For $x_i \in [-100, 100]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d} \left(\sum_{j=1}^{n_d} x_i \right)^2$$

Quartic: For $x_i \in [-1.28, 1.28]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d} i x_i^4$$

Rastrigin: For $x_i \in [-5.12, 5.12]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d} [x_i^2 - 10\cos(2\pi x_i) + 10]$$

Rosenbrock: For $x_i \in [-2.048, 2.048]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d - 1} [100(x_{i+1} - x_i^2)^2 + (x_i - 1)^2]$$

Shekel N: For $x_i \in [0, 10]$, $n_d = 4$, $m = 10$,

$$f(\mathbf{x}) = -\sum_{i=1}^{m} \frac{1}{(\mathbf{x} - \mathbf{A}_i)(\mathbf{x} - \mathbf{A}_i)^T + c_i}$$

where

$$\mathbf{A} = \begin{bmatrix} 4 & 4 & 4 & 4 \\ 1 & 1 & 1 & 1 \\ 8 & 8 & 8 & 8 \\ 6 & 6 & 6 & 6 \\ 3 & 7 & 3 & 7 \\ 2 & 9 & 2 & 9 \\ 5 & 5 & 3 & 3 \\ 8 & 1 & 8 & 1 \\ 6 & 2 & 6 & 2 \\ 7 & 3.6 & 7 & 3.6 \end{bmatrix} \text{ and } \mathbf{c} = \begin{bmatrix} 0.1 \\ 0.2 \\ 0.2 \\ 0.4 \\ 0.4 \\ 0.6 \\ 0.3 \\ 0.7 \\ 0.5 \\ 0.5 \end{bmatrix}$$

Spherical: For $x_i \in [-5.12, 5.12]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d} x_i^2$$

Step: For $x_i \in [-100, 100]$, and $n_d = 30$,

$$f(\mathbf{x}) = \sum_{i=1}^{n_d} (\lfloor x_i + 0.5 \rfloor)^2$$

The results reported in this section are averages and deviations over 30 independent runs. For each entry, the first row gives the average, and the second row gives the deviation. Each run used a population (swarm) of 30 indivuduals (particles), and continued for 100000 function evaluations.

Section VI-A provides an empirical analysis of the influence of parameter p_r on the performance of the BBDE, while Section VI-B compares the performance of the BBDE with other algorithms.

A. Recombination Parameter

The performance of the barebones DE has been evaluated for $p_r \in \{0.1, 0.3, 0.5, 0.7, 0.9\}$. Table I summarizes the results. Allthough the results in Table I suggest that smaller values of p_r are preferable, there are no significant difference in performance for the different values. This is also illustrated in Figure 1 for selected functions. Note that similar profiles were obtained for the other functions. Lower values of p_r makes sense, as exploration is then favored. A strategy to be explored at a later stage is to dynamically adjust p_r to increase over time.

B. Comparative Analysis

For each of the benchmark functions, the value of p_r that resulted in the best average fitness is selected for the BBDE. These results are then compared against the barebones (BB) PSO, the exploiting barebones (BBExp) PSO, Von Neumann (VN) PSO, DEPSO, and DE/rand/1/bin (DE) in Table II. No results are given for the DEPSO and DE for the himmelblau functions, due to the individuals (particles) leaving the search space (none of the algorithms used any mechanism to force individuals (particles) to remain in the search space).

Figure 2 illustrates results for selected functions. For the Ackley function, Figure 2(a) clearly shows how the DEPSO struggled, while the VN PSO achieved very good results. Interesting to note is that the barebones PSO algorithms and the BBDE achieved a faster reduction in fitness than the other algorithms. Allthough the DE obtained results similar to that of the barebones algorithms, it was slower in reducing fitness values. Figure 2(b) shows that both the DEPSO and the DE had problems with the Michalewicz function, with results significantly worse than that of the other algorithms. Figure 2(c) shows that the DEPSO obtained the best results, with a gradual decrease in average fitness. The barebones algorithms again showed the fastest reduction in average fitness values. For the step function (refer to Figure 2(d)), all algorithms (except the DEPSO and DE, for which values very close to zero were reached) obtained the global optimum for all runs. What is interesting to note is that again the barebones algorithms reached accurate solutions much faster than the other algorithms. The DEPSO, and especially the DE, took significantly longer to find good solutions.

The last row of Table II provides a performance rank for each algorithm. To calculate the performance rank, a position is assigned to each algorithm for each function, where a position of 1 indicates that the algorithm obtained the best average fitness for that problem, while a rank of 6 indicates the worst performance. The performance rank is the average position over all functions. The Von Neumann PSO is the best performer with a rank of 1.538, followed by the BBDE with a rank of 2.0. What is important to note is that the BBDE

TABLE I
INFLUENCE OF p_r ON THE PERFORMANCE OF BBDE

Function	$p_r = 0.1$	$p_r = 0.3$	$p_r = 0.5$	$p_r = 0.7$	$p_r = 0.9$
ackley	2.235287	2.210726	2.393488	2.136173	2.501318
	0.166974	0.178265	0.217434	0.159471	0.202922
foxholes	0	0	0	0.034414	0.068691
	0	0	0	0.033267	0.066402
griewank	0.043288	0.027314	0.027176	0.026950	0.029312
	0.011215	0.004971	0.006145	0.007671	0.009801
himmelblau	-832.472412	-836.045656	-825.325922	-825.325922	-818.179432
	9.947930	9.644610	10.410637	10.410637	10.701122
hyperellipsoid	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
michalewicz	-25.019098	-25.122886	-24.894284	-25.171186	-24.956519
	0.290695	0.259553	0.268938	0.305726	0.260732
quadric	0.320673e-3	0.168747e-3	0.148594e-3	0.202841e-3	0.131393e-2
	0.853671e-4	0.437480e-4	0.485624e-4	0.551421e-4	0.946728e-3
quartic	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
rastrigin	76.165639	72.185823	72.391682	75.891177	77.194900
	3.903804	3.018019	3.536751	2.985969	4.017189
rosenbrock	15.839071	16.636823	18.315201	14.295707	18.483846
	2.058175	2.093440	2.561361	0.948028	2.715411
shekeln	-6.395098	-7.251276	-6.016494	-5.595183	-6.874382
	0.632766	0.705802	0.639234	0.639550	0.692448
spherical	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0
step	0 ± 0	0 ± 0	0 ± 0	0 ± 0	0 ± 0

performs better than the two barbones PSO algorithms, the DE, and the DEPSO. It can therefore be reasoned that the BBDE combines the good characteristics of the barebones PSO and DE. It should also be noted that the performance of the Von Neumann PSO depends strongly on the values of w, c_1 and c_2. As indicated in [24], [25], wrong choices of values for these parameters may lead to divergent or cyclic trajectories. In contrast, BBDE does not require these parameters. Furthermore, the BBDE is mostly insensitive to the value of its only parameter, p_r.

VII. CONCLUSIONS

This paper presented a new population-based algorithm, as a hybrid of the barebones particle swarm optimizer (PSO) and differential evolution (DE). The particle position update is changed to probabilistically base a new position on a randomly selected personal best position, or a mutation of the particle attractor (i.e. weighted average of the personal best and neighborhood best). The BBDE does not make use of the standard PSO parameters (i.e. inertia weight, acceleration coefficients, and velocity clamping), and also removes the DE scale parameter. The only parameter is the probability of recombination, for which it was shown empirically that the BBDE is insensitive.

The BBDE outperformed all the algorithms used in this study, except for the Von Neumann PSO. Future research will investigate the performance of BBDE using a Von Neumann topology instead of the star topology. The performance of the BBDE on noisy problems will be investigated, as well as its scalability.

REFERENCES

[1] H.A. Abbass. The self-adaptive Pareto differential evolution algorithm. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 831–836, 2002.

[2] H.A. Abbass, R. Sarker, and C. Newton. PDE: a Pareto-frontier differential evolution approach for multi-objective optimization problems. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 971–978, 2001.

[3] M. Clerc and J. Kennedy. The Particle Swarm-Explosion, Stability, and Convergence in a Multidimensional Complex Space. *IEEE Transactions on Evolutionary Computation*, 6(1):58–73, 2002.

[4] R.C. Eberhart and J. Kennedy. A New Optimizer using Particle Swarm Theory. In *Proceedings of the Sixth International Symposium on Micromachine and Human Science*, pages 39–43, 1995.

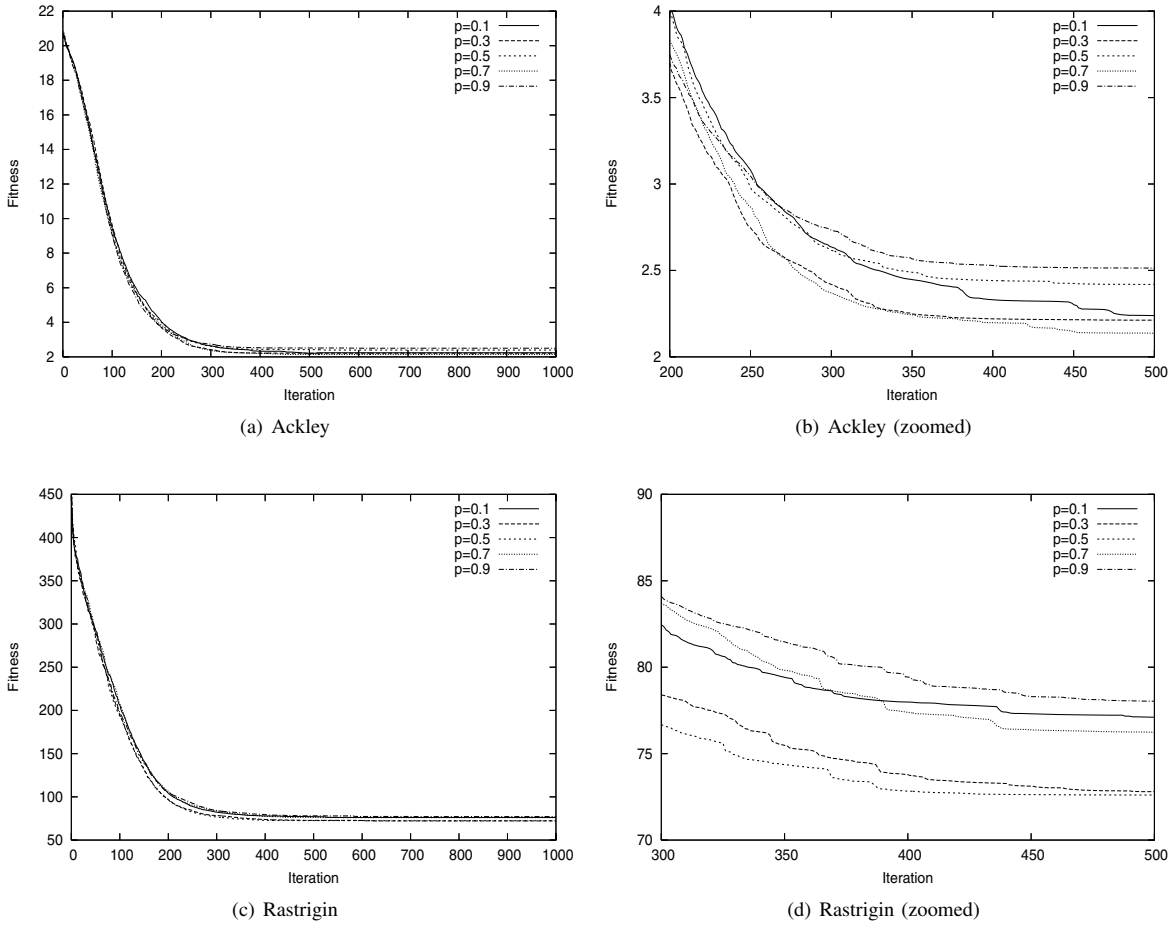
[5] A.P. Engelbrecht. *Fundamentals of Computational Swarm Intelligence*. Wiley & Sons, 2005.

[6] T. Hendtlass. A Combined Swarm Differential Evolution Algorithm for Optimization Problems. In *Proceedings of the Fourteenth International Conference on Industrial and Engineering Applications of Artificial Intelligence and Expert Systems, In: Lecture Notes in Computer Science*, volume 2070, pages 11–18. Springer-Verlag, 2001.

[7] S. Kannan, S.M.R. Slochanal, P. Subbaraj, and N.P. Padhy. Application of Particle Swarm Optimization Technique and its Variants to Generation Expansion Planning. *Electric Power Systems Research*, 70(3):203–210, 2004.

[8] J. Kennedy. Small Worlds and Mega-Minds: Effects of Neighborhood Topology on Particle Swarm Performance. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 3, pages 1931–1938, July 1999.

[9] J. Kennedy. Bare Bones Particle Swarms. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 80–87, April 2003.



(a) Ackley (b) Ackley (zoomed)

(c) Rastrigin (d) Rastrigin (zoomed)

Fig. 1. Performance of BBDE for Different Values of p_r

[10] J. Kennedy and R.C. Eberhart. Particle Swarm Optimization. In *Proceedings of the IEEE International Joint Conference on Neural Networks*, pages 1942–1948. IEEE Press, 1995.

[11] J. Kennedy, R.C. Eberhart, and Y. Shi. *Swarm Intelligence*. Morgan Kaufmann, 2001.

[12] J. Kennedy and R. Mendes. Population Structure and Particle Performance. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 1671–1676. IEEE Press, 2002.

[13] M.G.H Omran, A.P. Engelbrecht, and A. Salman. Empirical Analysis of Self-Adaptive Differential Evolution. *accepted for European Journal of Operational Research*, 2006.

[14] M.G.H. Omran, A. Salman, and A.P. Engelbrecht. Self-Adaptive Differential Evolution. In *Lecture Notes in Artificial Intelligence*, volume 3801, pages 192–199. Springer, 2005.

[15] E.S. Peer, F. van den Bergh, and A.P. Engelbrecht. Using Neighborhoods with the Guaranteed Convergence PSO. In *Proceedings of the IEEE Swarm Intelligence Symposium*, pages 235–242. IEEE Press, 2003.

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

[17] A.K. Qin and P.N. Suganthan. Self-adaptive differential evolution algorithm for numerical optimization. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 2, pages 1785–1791, 2005.

[18] J. Rönkkönen, S. Kukkonen, and K.V. Price. Real-Parameter Optimization with Differential Evolution. In *Proceedings of the IEEE Congress on Evolutionary Computation*, volume 1, pages 506–513, 2005.

[19] Y. Shi and R.C. Eberhart. A Modified Particle Swarm Optimizer. In *Proceedings of the IEEE Congress on Evolutionary Computation*, pages 69–73, May 1998.

[20] R. Storn. On the Usage of Differential Evolution for Function Optimization. In *Proceedings of the Biennieal Conference of the North American Fuzzy Information Processing Society*, pages 519–523, 1996.

[21] R. Storn and K. Price. Differential evolution - A Simple and efficient adaptive scheme for global optimization over continuous spaces. Technical Report TR-95-012, International Computer Science Institute, 1995.

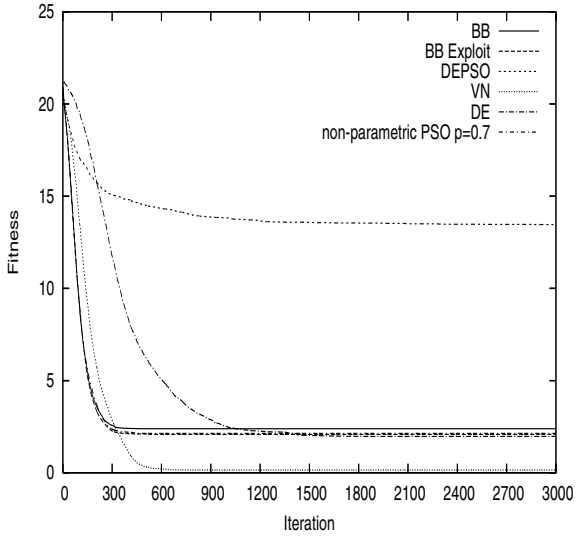
[22] R. Storn and K. Price. Differential Evolution – A Simple and Efficient Heuristic for global Optimization over Continuous Spaces. *Journal of Global Optimization*, 11(4):431–359, 1997.

[23] H. Talbi and M. Batouche. Hybrid particle swarm with differential evolution for multimodal image registration. In *Proceedings of the IEEE International Conference on Industrial Technology*, volume 3, pages 1567–1573, 2004.

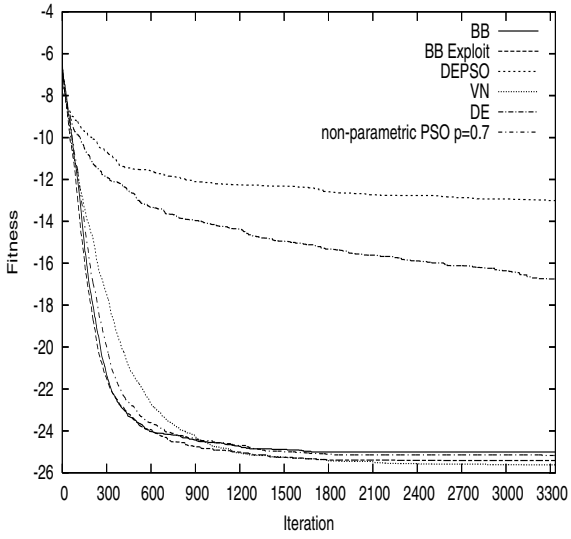
[24] F. van den Bergh. *An Analysis of Particle Swarm Optimizers*. PhD thesis, Department of Computer Science, University of Pretoria, Pretoria, South Africa, 2002.

[25] F. van den Bergh and A.P. Engelbrecht. A Study of Particle Swarm Optimization Particle Trajectories. *Information Sciences*, 176(8):937–971, 2006.

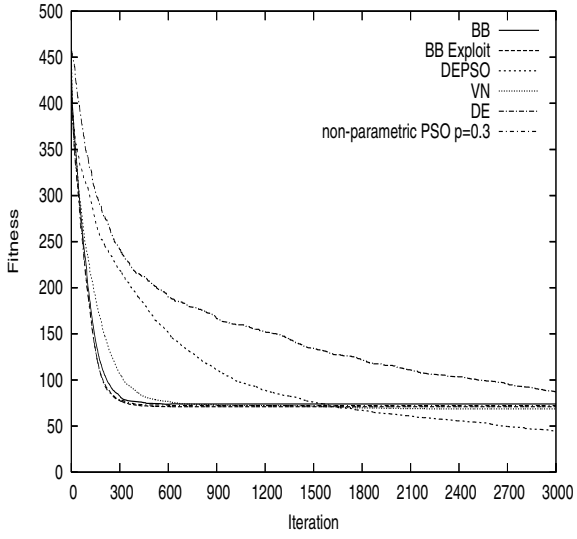
[26] W-J. Zhang and X-F. Xie. DEPSO: hybrid particle swarm with differential evolution operator. In *IEEE International Conference on Systems, Man, and Cybernetics*, volume 4, pages 3816–3821, 2003.



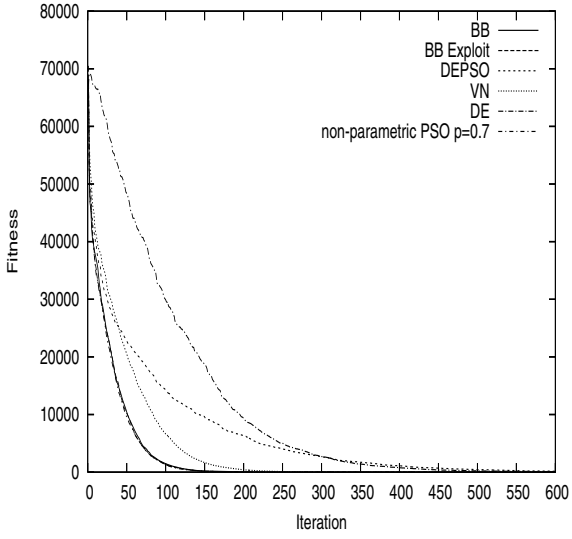
(a) Ackley



(b) Michalewicz



(c) Rastrigin



(d) Step

Fig. 2. Performance Comparison

TABLE II
COMPARATIVE RESULTS

Function	**BB**	**BBExp**	**VN**	**DEPSO**	**DE**	**BBDE**
ackley	2.389906	2.081037	0.150282	13.435463	1.971628	2.136173
	0.198375	0.222534	0.069756	0.550129	0.276186	0.159471
foxholes	0	0	0	0	0.520673	0
	0	0	0	0	0.362131	0
griewank	0.393245e-1	0.474688e-1	0.729644e-2	0.244013e-1	0.284551e-1	0.269504e-1
	0.645749e-2	0.219540e-1	0.185556e-2	0.402870e-2	0.102846e-1	0.767095e-2
himmelblau	-828.899167	-821.752677	-850.338636	N/A	N/A	-836.045656
	10.202072	10.576323	7.812089			9.644610
hyperellipsoid	0	0	0	0.379939e-9	0.168737e-11	0
	0	0	0	0.232345e-9	0.935678e-12	0
michalewicz	-25.006655	-25.417601	-25.623863	-13.009471	-16.752122	-25.171186
	0.280002	0.276912	0.251935	0.700428	0.271555	0.305726
quadric	0.125863e-3	0.457733e-3	0.772398	8153.745542	30788.550654	0.148594e-3
	0.357456e-4	0.237904e-3	0.113289	743.527784	3229.185538	0.485624e-4
quartic	0	0	0	0.436603e-14	0.728774e-5	0
	0	0	0	0.348727e-12	0.665595e-5	0
rastrigin	73.969877	71.019324	68.549094	40.970572	79.786811	72.185823
	3.841866	2.846686	3.323843	2.021865	5.293505	3.018019
rosenbrock	16.195936	17.103387	28.719351	30.243866	34.434192	14.295707
	0.437205	2.079606	3.473769	3.118939	3.686825	0.948028
shekeln	-7.006751	-6.557710	-10.265707	-7.024528	-6.992243	-7.251276
	0.671463	0.657450	0.353107	0.664893	0.705697	0.705802
spherical	0	0	0	0.339409e-10	0.762325e-11	0
	0	0	0	0.255639e-10	0.462353e-11	0
step	0	0	0	0.692243e-9	0.216447e-3	0
	0	0	0	0.278905e-9	0.209224e-3	0
Ranking	2.615	2.692	1.538	3.250	3.830	2.000