A Comparative Study of Differential Evolution, Particle Swarm Optimization, and Evolutionary Algorithms on Numerical Benchmark Problems

Jakob Vesterstrøm
BiRC - Bioinformatics Research Center
University of Aarhus, Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark
Email: jve@birc.dk

René Thomsen
BiRC - Bioinformatics Research Center
University of Aarhus, Ny Munkegade, Bldg. 540
DK-8000 Aarhus C, Denmark
Email: thomsen@birc.dk

***Abstract*— Several extensions to evolutionary algorithms (EAs) and particle swarm optimization (PSO) have been suggested during the last decades offering improved performance on selected benchmark problems. Recently, another search heuristic termed differential evolution (DE) has shown superior performance in several real-world applications. In this paper we evaluate the performance of DE, PSO, and EAs regarding their general applicability as numerical optimization techniques. The comparison is performed on a suite of 34 widely used benchmark problems. The results from our study show that DE generally outperforms the other algorithms. However, on two noisy functions, both DE and PSO were outperformed by the EA.**

I. Introduction

The evolutionary computation (EC) community has shown a significant interest in optimization for many years. In particular, there has been a focus on global optimization of numerical, real-valued 'black-box' problems for which exact and analytical methods do not apply. Since the mid-sixties many general-purpose optimization algorithms have been proposed for finding near-optimal solutions to this class of problems; most notably: evolution strategies (ES) [8], evolutionary programming (EP) [3], and genetic algorithms (GA) [6].

Many efforts have also been devoted to compare these algorithms to each other. Typically, such comparisons have been based on artificial numerical benchmark problems. The goal of many studies was to verify that one algorithm outperformed another on a given set of problems. In general, it has been possible to improve a given standard method within a restricted set of benchmark problems by making minor modifications to it.

Recently, particle swarm optimization (PSO) [7] and differential evolution (DE) [11] have been introduced and particularly PSO has received increased interest from the EC community. Both techniques have shown great promise in several real-world applications [4], [5], [12], [14]. However, to our knowledge, a comparative study of DE, PSO, and GAs on a large and diverse set of problems has never been made.

In this study, we investigated the performance of DE, PSO, and an evolutionary algorithm (EA)[1] on a selection of 34 numerical benchmark problems. The main objective was to examine whether one of the tested algorithms would outperform all others on a majority of the problems. Additionally, since we used a rather large number of benchmark problems, the experiments would also reveal whether the algorithms would have any particular difficulties or preferences.

Overall, the experimental results show that DE was far more efficient and robust (with respect to reproducing the results in several runs) compared to PSO and the EA. This suggests that more emphasis should be put on DE when solving numerical problems with real-valued parameters. However, on two noisy test problems, DE was outperformed by the other algorithms.

The paper is organized as follows. In Section II we introduce the methods used in the study: DE, PSO, and the EA. Further, Section III outlines the experimental setup, parameter settings, and benchmark problems used. The experimental results are presented in Section IV. Finally, Section V contains a discussion of the experimental results.

II. Methods

A. Differential Evolution

The DE algorithm was introduced by Storn and Price in 1995 [11]. It resembles the structure of an EA, but differs from traditional EAs in its generation of new candidate solutions and by its use of a 'greedy' selection scheme. DE works as follows: First, all individuals are randomly initialized and evaluated using the fitness function provided. Afterwards, the following process will be executed as long as the termination condition is not fulfilled: For each individual in the population, an offspring is created using the weighted difference of parent solutions. In this study we used the $DE/rand/1/exp$ scheme shown in Figure 1. The offspring replaces the parent if it is fitter. Otherwise, the parent survives and is passed on to the next iteration of the algorithm.

[1]The EA used in this study resembled a real-valued GA.

0-7803-8515-2/04/$20.00 ©2004 IEEE

```
procedure create offspring O[i] from parent P[i] {
  O[i] = P[i] // copy parent genotype to offspring
  randomly select parents P[i1], P[i2], P[i3],
    where i1 ≠ i2 ≠ i3 ≠ i
  n = U(0, dim)
  for (j = 0; j < dim ∧ U(0,1) < CR; j++) {
      O[i][n] = P[i1][n] + F · (P[i2][n] − P[i3][n])
      n = (n + 1) mod dim
  }
}
```

Fig. 1. Pseudo-code for creating an offspring in DE. $U(0;x)$ is a uniformly distributed number between 0 and x. CR is the probability of crossover, F is the scaling factor, and dim is the number of problem parameters (problem dimensionality).

B. Particle Swarm Optimization

PSO was introduced by Kennedy and Eberhart in 1995. It was inspired by the swarming behavior as is displayed by a flock of birds, a school of fish, or even human social behavior being influenced by other individuals [7].

PSO consists of a swarm of particles moving in an n-dimensional, real-valued search space of possible problem solutions. Every particle has a position vector $\vec{x}$ encoding a candidate solution to the problem (similar to the genotype in EAs) and a velocity vector $\vec{v}$. Moreover, each particle contains a small memory that stores its own best position seen so far $\vec{p}$ and a global best position $\vec{g}$ obtained through communication with its neighbor particles. In this study we used the fully connected network topology for passing on information (see [15] for more details).

Intuitively, the information about good solutions spreads through the swarm, and thus the particles tend to move to good areas in the search space. At each time step t, the velocity is updated and the particle is moved to a new position. This new position is calculated as the sum of the previous position and the new velocity:

$$\vec{x}(t+1) = \vec{x}(t) + \vec{v}(t+1)$$

The update of the velocity from the previous velocity to the new velocity is determined by the following equation:

$$\vec{v}(t{+}1) = \omega{\cdot}\vec{v}(t){+}U(0,\phi_1)(\vec{p}(t){-}\vec{x}(t)){+}U(0,\phi_2)(\vec{g}(t){-}\vec{x}(t)),$$

where $U(a,b)$ is a uniformly distributed number between a and b. The parameter ω is called the inertia weight [10] and controls the magnitude of the old velocity $\vec{v}(t)$ in the calculation of the new velocity, whereas ϕ_1 and ϕ_2 determine the significance of $\vec{p}(t)$ and $\vec{g}(t)$, respectively. Furthermore, at any time step of the algorithm v_i is constrained by the parameter v_{max}.

The swarm in PSO is initialized by assigning each particle to a uniformly and randomly chosen position in the search space. Velocities are initialized randomly in the range $[-v_{max}, v_{max}]$.

C. Attractive and Repulsive PSO

The attractive and repulsive PSO (arPSO) [9], [15] was introduced by Vesterstrøm and Riget to overcome the problems of premature convergence [1]. The modification of the basic PSO scheme is to modify the velocity update formula when the swarm diversity becomes less than a value d_{low}. This modification corresponds to repulsion of the particles instead of the usual attraction scheme. Thus, the velocity is updated according to:

$$\vec{v}(t{+}1) = \omega{\cdot}\vec{v}(t){-}U(0,\phi_1)(\vec{p}(t){-}\vec{x}(t)){-}U(0,\phi_2)(\vec{g}(t){-}\vec{x}(t))$$

This will increase the diversity over some iterations, and eventually when another value d_{high} is reached, the commonly used velocity update formula will be used again. Thus, arPSO is able to zoom out when an optimum has been reached, followed by zooming in on another hot spot, possibly discovering a new optimum in the vicinity of the old one. Previously, arPSO was shown to be more robust than the basic PSO on problems with many optima [9].

The arPSO algorithm was included in this study as a representative for the large number of algorithmic extensions to PSO that try to avoid the problem of premature convergence. Other PSO extensions could have been chosen but we selected this particular one, since the performance of arPSO was as good (or better) as many other extensions [15].

D. Evolutionary Algorithm

In this study we used a simple EA (SEA) that was previously found to work well on real-world problems [13]. The SEA works as follows: First, all individuals are randomly initialized and evaluated according to a given fitness function. Afterwards, the following process will be executed as long as the termination condition is not fulfilled: Each individual has a probability of being exposed to either mutation or recombination (or both). Mutation and recombination operators are applied with probability p_m and p_c, respectively. The mutation and recombination operators used are Cauchy mutation using an annealing scheme and arithmetic crossover, respectively. Finally, tournament selection [2] comparing pairs of individuals is applied to weed out the least fit individuals.

The *Cauchy mutation* operator is similar to the well-known Gaussian mutation operator, but the Cauchy distribution has thick tails that enable it to generate considerable changes more frequently than the Gaussian distribution. The Cauchy distribution has the form:

$$C(x,\alpha,\beta) = \frac{1}{\beta\pi\left(1+\left(\frac{x-\alpha}{\beta}\right)^2\right)}$$

where $\alpha \geq 0, \beta > 0, -\infty < x < \infty$ (α and β are parameters that affect the mean and spread of the distribution). All of the solution parameters are subject to mutation and the variance is scaled with 0.1 $\times$ the range of the specific parameter in question.

Moreover, an annealing scheme was applied to decrease the value of β as a function of the elapsed number of generations t.

α was fixed to 0. In this study we used the following annealing function:

$$\beta(t) = \frac{1}{1+t}$$

In arithmetic crossover the offspring is generated as a weighted mean of each gene of the two parents, i.e.,

$$offspring_i = r \cdot parent1_i + (1 - r) \cdot parent2_i,$$

where $offspring_i$ is the i'th gene of the offspring and $parent1_i$ and $parent2_i$ refer to the i'th gene of the two parents, respectively. The weight r is determined by a random value between 0 and 1.

E. Additional Remarks

All methods described above used truncation of infeasible solutions to the nearest boundary of the search space. Further, the termination criterion for all methods was to stop the search process, when the current number of fitness evaluations exceeded a maximum number of evaluations allowed.

III. Experiments

A. Experimental Setup and Data Sampling

The algorithms used for comparison were DE, PSO, arPSO, and the SEA. For all algorithms the parameter settings were manually tuned, based on a few preliminary experiments. The specific settings for each of the algorithms are described below. Each algorithm was tested with all of the numerical benchmarks shown in Table I. In addition, we tested the algorithms on f_1–f_{13} in 100 dimensions, yielding a total of 34 numerical benchmarks. For each algorithm, the maximum number of evaluations allowed was set to 500,000 for the 30-dimensional (or less) benchmarks and to 5,000,000 for the 100-dimensional benchmarks. Each of the experiments was repeated 30 times with different random seeds, and the average fitness of the best solutions (e.g. individuals or particles) throughout the optimization run was recorded.

B. DE Settings

DE has three parameters: The size of the population (*popsize*), the crossover constant (*CR*), and the scaling factor (*F*). In all experiments they were set to the following values: $popsize = 100, CR = 0.9, F = 0.5$

C. PSO Settings

PSO has several parameters: The number of particles in the swarm ($swarmsize$), maximum velocity (v_{max}), the parameters for attraction towards personal best and the neighborhoods best found solutions (ϕ_1 and ϕ_2), and the inertia weight (ω). For PSO we used these settings: $swarmsize = 25$, $v_{max} =$ 15% of the longest axis-parallel interval in the search space, $\phi_1 = 1.8$, $\phi_2 = 1.8$, and $\omega = 0.6$.

Often the inertia weight is decreased linearly over time. The setting for PSO in this study is a bit unusual, because the inertia weight was held constant during the run. However, it was found that for the easier problems, the chosen settings outperformed the setting where ω was annealed. The setting with constant ω, on the other hand, performed poorly on multimodal problems. To be fair to PSO, we therefore included the arPSO algorithm, which was known to outperform PSO (even with annealed ω) on multi modal problems [9].

D. arPSO Settings

In addition to the basic PSO settings, arPSO used the following parameter settings: $d_{low} = 0.000005$ and $d_{high} = 0.25$. The two parameters were only marginally dependent on the problem, and these settings were consistent with the settings found in previous studies [9].

E. SEA Settings

The SEA used a population size of 100. The probability of mutating and crossing over individuals was fixed at $p_m = 0.9$ and $p_c = 0.7$, respectively. Tournament selection with a tournament size of two was used to select the individuals for the next generation. Further, elitism with an elite size of one was used to keep the overall best solution found in the population.

F. Numerical Benchmark Functions

For evaluating the four algorithms, we used a test suite of benchmark functions previously introduced by Yao and Liu [16]. The suite contained a diverse set of problems, including unimodal as well as multimodal functions, and functions with correlated and uncorrelated variables. Additionally, two noisy problems and a single problem with plateaus were included. The dimensionality of the problems originally varied from 2 to 30, but we extended the set with 100-dimensional variants to allow for comparison on more difficult problem instances. Table I lists the benchmark problems, the ranges of their search spaces, their dimensionalities, and their global minimum fitnesses. We omitted f_{19} and f_{20} from Yao and Liu's study [16] because of difficulties in obtaining the definitions of the constants used in these functions (they were not provided in [16]).

IV. Results

A. Problems of Dimensionality 30 or Less

The results for the benchmark problems f_1–f_{23} are shown in Table II and III. Moreover, Figure 2 shows the convergence graphs for selected benchmark problems. All results below 10^{-25} were reported as '0.0000000e+00'.

For functions f_1–f_4 there is a consistent performance pattern across all algorithms: PSO is the best, and DE is almost as good. They both converge exponentially fast toward the fitness optimum (resulting in a straight line when plotted using a logarithmic y-axis). The SEA is many times slower than the two other methods, and even though it eventually might converge toward the optimum, it requires several hours to find solutions that PSO and DE can reach in a few seconds. This analysis is illustrated in Figure 2 (a).

On function f_5, DE is superior to both PSO and arPSO and the SEA. Only DE converges toward the optimum. After 300,000 evaluations, it commences to fine-tune around the

TABLE I

NUMERICAL BENCHMARK FUNCTIONS WITH A VARYING NUMBER OF DIMENSIONS (DIM). REMARKS: 1) FUNCTIONS SINE AND COSINE TAKE ARGUMENTS IN RADIANS. 2) THE NOTATION $(a)^T(b)$ DENOTES THE DOT PRODUCT BETWEEN VECTORS a AND b. 3) THE FUNCTION u AND THE VALUES y_i REFERRED TO IN f_{12} AND f_{13} ARE GIVEN BY $u(x,a,b,c) = b(x-a)^c$ IF $x > a$, $u(x,a,b,c) = b(-x-a)^c$ IF $x < a$, $u(x,a,b,c) = 0$ IF $-a \leq x \leq a$, AND FINALLY $y_i = 1 + \frac{1}{4}(x_i + 1)$. THE MATRIX a USED IN f_{14}, THE VECTORS a AND b USED IN f_{15}, AND THE MATRIX a AND THE VECTOR c USED IN f_{21}–f_{23}, ARE ALL DEFINED IN THE APPENDIX.

Function		Dim	Ranges	Minimum value
$f_1(\vec{x}) =$	$\sum_{i=0}^{n-1} x_i^2$	30/100	$-5.12 \leq x_i \leq 5.12$	$f_1(\vec{0}) = 0$
$f_2(\vec{x}) =$	$\sum_{i=0}^{n-1} \lvert x_i \rvert + \prod_{i=0}^{n-1} x_i$	30/100	$-10 \leq x_i \leq 10$	$f_2(\vec{0}) = 0$
$f_3(\vec{x}) =$	$\sum_{i=0}^{n-1} \left(\sum_{j=0}^{i} x_i \right)^2$	30/100	$-100 \leq x_i \leq 100$	$f_3(\vec{0}) = 0$
$f_4(\vec{x}) =$	$\max \lvert x_i \rvert, 0 \leq i < n$	30/100	$-100 \leq x_i \leq 100$	$f_4(\vec{0}) = 0$
$f_5(\vec{x}) =$	$\sum_{i=0}^{n-1} \left(100 \cdot (x_{i+1} - (x_i)^2)^2 + (x_i - 1)^2 \right)$	30/100	$-30 \leq x_i \leq 30$	$f_5(\vec{1}) = 0$
$f_6(\vec{x}) =$	$\sum_{i=0}^{n-1} \left(\lfloor x_i + \frac{1}{2} \rfloor \right)^2$	30/100	$-100 \leq x_i \leq 100$	$f_6(\vec{p}) = 0$, $-\frac{1}{2} \leq p_i < \frac{1}{2}$
$f_7(\vec{x}) =$	$\left(\sum_{i=0}^{n-1} (i+1) \cdot x_i^4 \right) + rand[0,1[$	30/100	$-1.28 \leq x_i \leq 1.28$	$f_7(\vec{0}) = 0$
$f_8(\vec{x}) =$	$\sum_{i=0}^{n-1} -x_i \cdot \sin(\sqrt{\lvert x_i \rvert})$	30/100	$-500 \leq x_i \leq 500$	$f_8(420.\vec{9}7) =$ 12569.5/41898.3
$f_9(\vec{x}) =$	$\sum_{i=0}^{n-1} (x_i^2 - 10\cos(2\pi x_i) + 10)$	30/100	$-5.12 \leq x_i \leq 5.12$	$f_9(\vec{0}) = 0$
$f_{10}(\vec{x}) =$	$-20\exp\left(-0.2\sqrt{\frac{1}{n}\sum_{i=0}^{n-1} x_i^2}\right) -$ $\exp\left(\frac{1}{n}\sum_{i=0}^{n-1}\cos(2\pi x_i)\right) + 20 + e$	30/100	$-32 \leq x_i \leq 32$	$f_{10}(\vec{0}) = 0$
$f_{11}(\vec{x}) =$	$\frac{1}{4000}\left(\sum_{i=0}^{n-1} x_i^2\right) + \left(\prod_{i=0}^{n-1} \cos(\frac{x_i}{\sqrt{i+1}})\right) + 1$	30/100	$-600 \leq x_i \leq 600$	$f_{11}(\vec{0}) = 0$
$f_{12}(\vec{x}) =$	$\frac{\pi}{n}\{10(\sin(\pi y_1))^2 +$ $\sum_{i=0}^{n-2}\left((y_i-1)^2(1+10(sin(\pi y_{i+1}))^2)\right) +$ $(y_n-1)^2\} + \sum_{i=0}^{n-1} u(x_i,10,100,4)$	30/100	$-50 \leq x_i \leq 50$	$f_{12}(\vec{-1}) = 0$
$f_{13}(\vec{x}) =$	$0.1\{(\sin(3\pi x_1))^2 +$ $\sum_{i=0}^{n-2}\left((x_i-1)^2(1+(sin(3\pi x_{i+1}))^2)\right) + (x_n-1)$ $(1+(\sin(2\pi x_n))^2)\} + \sum_{i=0}^{n-1} u(x_i,5,100,4)$	30/100	$-50 \leq x_i \leq 50$	$f_{13}(1,\ldots,1,-4.76)$ $= -1.1428$
$f_{14}(\vec{x}) =$	$\left(\frac{1}{500} + \sum_{j=0}^{24}(j+1+\sum_{i=0}^{1}(x_i-a_{ij})^6)^{-1}\right)^{-1}$	2	$-65.54 \leq x_i \leq 65.54$	$f_{14}(-3\vec{1}.95)$ $= 0.998$
$f_{15}(\vec{x}) =$	$\sum_{i=0}^{10}\left(a_i - \frac{x_0(b_i^2+b_i x_1)}{b_i^2+b_i x_2+x_3}\right)^2$	4	$-5 \leq x_i \leq 5$	$f_{15}(0.19,0.19,0.12,0.14)$ $= 0.0003075$
$f_{16}(\vec{x}) =$	$4x_0^2 - 2.1x_0^4 + \frac{1}{3}x_0^6 + x_0x_1 - 4x_1^2 + 4x_1^4$	2	$-5 \leq x_i \leq 5$	$f_{16}(-0.09,0.71)$ $= -1.0316$
$f_{17}(\vec{x}) =$	$(x_1 - \frac{5.1}{4\pi^2}x_0^2 + \frac{5}{\pi}x_0 - 6)^2 +$ $10(1-\frac{1}{8\pi})\cos(x_0) + 10$	2	$-5 \leq x_i \leq 15$	$f_{17}(9.42,2.47) = 0.398$
$f_{18}(\vec{x}) =$	$\{1+(x_0+x_1+1)^2(19-14x_0+3x_0^2-14x_1+$ $6x_0x_1+3x_1^2)\}\{30+(2x_0-3x_1)^2$ $(18-32x_0+12x_0^2+48x_1-36x_0x_1+27x_1^2)\}$	2	$-2 \leq x_i \leq 2$	$f_{18}(1.49\text{e-}05, 1.00) = 3$
$f_{21}(\vec{x}) =$	$-\sum_{i=0}^{4}\left((x-a_i)^T(x-a_i)+c_i\right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{21}(\approx \vec{4}) = -10.2$
$f_{22}(\vec{x}) =$	$-\sum_{i=0}^{6}\left((x-a_i)^T(x-a_i)+c_i\right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{22}(\approx \vec{4}) = -10.4$
$f_{23}(\vec{x}) =$	$-\sum_{i=0}^{9}\left((x-a_i)^T(x-a_i)+c_i\right)^{-1}$	4	$0 \leq x_i \leq 10$	$f_{23}(\approx \vec{4}) = -10.5$

optimum at an exponentially progressing rate. We may note that PSO performs moderately better than the SEA.

DE and the SEA easily find the optimum for the f_6 function, whereas both PSOs fail. This test function consists of plateaus, and apparently both PSO methods have difficulties with functions of this kind. The average best fitness value of 0.04 for basic PSO comes from failing twice in 30 runs.

Function f_7 is a noisy problem. All algorithms seem to converge in a similar pattern, see Figure 2 (d). The SEA had the best convergence speed, followed by arPSO, PSO, and finally DE.

Functions f_8–f_{13} are highly multimodal functions. On all of them, DE clearly performs best and it finds the global optimum in all cases. Neither the SEA nor the PSOs find the global optimum for these functions in any of the runs. Further, the SEA consistently outperforms both PSOs, with f_{10} being an exception.

Both DE, the SEA, and arPSO come very close to the global optimum on f_{14} in all runs, but only DE hits the exact optimum every time, making it the best algorithm on this problem. PSO occasionally stagnates at a local optimum, which is the reason for its poor average best fitness.

On f_{15} both PSOs perform worse than DE and SEA. DE converges very fast to good values near the optimum, but seems to stagnate suboptimally. The SEA converges slowly, but outperforms DE after 500,000 fitness evaluations. To investigate if the SEA would continue its convergence and ultimately reach the optimum, we tried to let it run for 2 million evaluations. In the majority of runs, it actually found the optimum after approximately 1 million evaluations (data not shown).

Functions f_{16}–f_{18} are all easy problems, and all algorithms are able to find near optimum solutions quickly. Both PSO methods have particularly fast convergence. For some reason, the SEA seems to be able to fine-tune its results slightly better than the other algorithms.

The last three problems are f_{21}–f_{23}. Again, DE is superior compared to the other algorithms. It finds optimum in all cases. DE, the SEA, and PSO all converge quickly, but the SEA stagnates before finding the optimum, and both PSO methods converge even earlier. arPSO performs better than PSO, and is almost as good as the SEA.

B. Problems of Dimensionality 100

On f_1, PSO and DE have good exponential convergence to optimum (similar to the results in 30 dimensions) and the SEA is much slower. However, on f_2 the picture has changed. DE still has exponential convergence to optimum, but both PSOs fail to find the optimum – they are now performing worse than the SEA. The same pattern occurs for f_3 and f_4. This contrasts with the 30 dimensional cases, where PSO performed exceptionally well for f_1–f_4.

On the difficult f_5 problem, DE is superior and finds optimum after 3.5 million evaluations. The other algorithms fail to find the optimum. However, the SEA is slightly better than the basic PSO.

Both DE and the SEA quickly find the optimum of f_6 in all runs. PSO only finds the optimum in 9 out of 30 runs. This is the reason for the average of 2.1 for PSO on this problem.

The results for the 100 dimensional version of f_7 are similar to the results in the 30 dimensional case. The SEA has the best convergence, arPSO is slightly slower, followed by PSO, and finally DE.

For problems f_8–f_{13} the results also resemble those from 30 dimensions. One exception is that arPSO is now marginally worse than the SEA on f_{10}. Thus, in 100 dimensions the SEA is consistently better than both PSOs on these six problems.

V. Discussion

Overall, DE is clearly the best performing algorithm in this study. It finds the lowest fitness value for most of the problems. The only exceptions are: 1) The noisy f_7 problem, where the nature of the convergence of DE is similar, but still slower than the other algorithms. Apparently, DE faces difficulties on noisy problems. 2) On f_{15} DE stagnates on a suboptimal value (and both PSOs even earlier). Only the SEA is able to find the optimum on this problem, which only has four problem parameters, but still appears to be very difficult.

We have not tried to tune DE to these problems. Most likely, one could improve the performance of DE by altering the crossover scheme, varying the parameters CR and F, or using a more 'greedy' offspring generation strategy (e.g. $DE/best/1/exp$).

DE is robust; it is able to reproduce the same results consistently over many trials, whereas the performance of PSO and arPSO is far more dependent on the randomized initialization of the individuals. This difference is profound on the 100 dimensional benchmark problems. As a result, both versions of PSO must be executed several times to ensure good results, whereas one run of DE and the SEA usually suffices.

PSO is more sensitive to parameter changes than the other algorithms. When changing the problem, one probably need to change parameters as well to sustain optimal performance. This is not the case for the SEA and DE. The 100 dimensional problems illustrate this point. Thus, the settings for both PSOs do not generalize to 100 dimensions, whereas DE and the SEA can be used with the same settings and still give the same type of convergence.

In general, DE shows great fine-tuning abilities, but on f_{16} and f_{17} it fails in comparison to the SEA. We have not determined why DE fails to fine tune these particular problems, but it would be interesting to investigate why.

Regarding convergence speed, PSO is always the fastest, whereas the SEA or arPSO are always the slowest. However, the SEA could be further improved with a more 'greedy' selection scheme similar to DE. Especially on the very easy functions f_{16}–f_{18}, PSO has a very fast convergence (3-4 times faster than DE). This may be of practical relevance for some real-world problems where the evaluation is computationally expensive and the search space is relatively simple and of low dimensionality.

TABLE II

RESULTS FOR ALL ALGORITHMS ON BENCHMARK PROBLEMS OF DIMENSIONALITY 30 OR LESS (MEAN OF 30 RUNS AND STANDARD DEVIATIONS (STDDEV)). FOR EACH PROBLEM, THE BEST PERFORMING ALGORITHM(S) IS EMPHASIZED IN BOLDFACE.

Benchmark	DE		PSO		arPSO		SEA	
Problem	Mean	Stddev	Mean	Stddev	Mean	Stddev	Mean	Stddev
1	**0.0000000e+00**	0.00e+00	**0.0000000e+00**	0.00e+00	6.8081735e-13	5.30e-13	1.7894112e-03	2.77e-04
2	**0.0000000e+00**	0.00e+00	**0.0000000e+00**	0.00e+00	2.0892037e-02	1.48e-01	1.7207452e-02	1.70e-03
3	2.0200713e-09	8.26e-10	**0.0000000e+00**	0.00e+00	**0.0000000e+00**	2.13e-25	1.5891817e-02	4.25e-03
4	3.8502177e-08	9.17e-09	**2.1070152e-16**	8.01e-16	1.4183790e-05	8.27e-06	1.9827734e-02	2.07e-03
5	**0.0000000e+00**	0.00e+00	4.0263857e+00	4.99e+00	3.5509286e+02	2.15e+03	3.1318954e+01	1.74e+01
6	**0.0000000e+00**	0.00e+00	4.0000000e-02	1.98e-01	1.8980000e+01	6.30e+01	**0.0000000e+00**	0.00e+00
7	4.9390831e-03	1.13e-03	1.9082207e-03	1.14e-03	3.8866682e-04	4.78e-04	**7.1062480e-04**	3.27e-04
8	**-1.2569481e+04**	2.30e-04	-7.1874076e+03	6.72e+02	-8.5986527e+03	2.07e+03	-1.1669334e+04	2.34e+02
9	**0.0000000e+00**	0.00e+00	4.9170789e+01	1.62e+01	2.1491414e+00	4.91e+00	7.1789575e-01	9.22e-01
10	**-1.1901591e-15**	7.03e-16	1.4046895e+00	7.91e-01	1.8422773e-07	7.15e-08	1.0468180e-02	9.08e-04
11	**0.0000000e+00**	0.00e+00	2.3528934e-02	3.54e-02	9.2344555e-02	3.41e-01	4.6366988e-03	3.96e-03
12	**0.0000000e+00**	0.00e+00	3.8199611e-01	8.40e-01	8.5597888e-03	4.79e-02	4.5626102e-06	8.11e-07
13	**-1.1428244e+00**	4.45e-08	-5.9688703e-01	5.17e-01	-9.6263537e-01	5.14e-01	-1.1427420e+00	1.34e-05
14	**9.9800390e-01**	3.75e-08	1.1570484e+00	3.68e-01	9.9800393e-01	2.13e-08	9.9800400e-01	4.33e-08
15	4.1736828e-04	3.01e-04	1.3378460e-03	3.94e-03	1.2476701e-03	3.96e-03	**3.7041858e-04**	8.78e-05
16	-1.0316285e+00	1.92e-08	-1.0316284e+00	3.84e-08	-1.0316284e+00	3.84e-08	**-1.0316300e+00**	3.16e-08
17	3.9788735e-01	1.17e-08	3.9788736e-01	5.01e-09	3.9788736e-01	5.01e-09	**3.9788700e-01**	2.20e-08
18	**3.0000000e+00**	0.00e+00	**3.0000000e+00**	0.00e+00	3.5162719e+00	3.65e+00	**3.0000000e+00**	0.00e+00
21	**-1.0153201e+01**	4.60e-07	-5.3944733e+00	3.40e+00	-8.1809408e+00	2.60e+00	-8.4076288e+00	3.16e+00
22	**-1.0402943e+01**	3.58e-07	-6.9460507e+00	3.70e+00	-8.4352620e+00	2.83e+00	-8.9125580e+00	2.86e+00
23	**-1.0536412e+01**	2.09e-07	-6.7107552e+00	3.77e+00	-8.6155040e+00	2.88e+00	-9.7995696e+00	2.24e+00

TABLE III

RESULTS FOR ALL ALGORITHMS ON BENCHMARK PROBLEMS OF DIMENSIONALITY 100 (MEAN OF 30 RUNS AND STANDARD DEVIATIONS (STDDEV)). FOR EACH PROBLEM, THE BEST PERFORMING ALGORITHM(S) IS EMPHASIZED IN BOLDFACE.

Benchmark	DE		PSO		arPSO		SEA	
Problem	Mean	Stddev	Mean	Stddev	Mean	Stddev	Mean	Stddev
1	**0.0000000e+00**	0.00e+00	**0.0000000e+00**	0.00e+00	7.4869991e+02	2.31e+03	5.2291447e-04	5.18e-05
2	**0.0000000e+00**	0.00e+00	1.8045813e+01	6.52e+01	3.9637792e+01	2.45e+01	1.7371780e-02	9.43e-04
3	**5.8734789e-10**	1.83e-10	3.6666668e+03	6.94e+03	1.8174752e+01	2.50e+01	3.6846433e-02	6.06e-03
4	**1.1284972e-09**	1.42e-10	5.3121806e+00	8.63e-01	2.4367166e+00	3.80e-01	7.6708840e-03	5.71e-04
5	**0.0000000e+00**	0.00e+00	2.0203629e+02	7.66e+02	2.3609401e+02	1.25e+02	9.2492247e+01	1.29e+01
6	**0.0000000e+00**	0.00e+00	2.1000000e+00	3.52e+00	4.1183333e+02	4.21e+02	**0.0000000e+00**	0.00e+00
7	7.6640871e-03	6.58e-04	2.7845728e-02	7.31e-02	3.2324733e-03	7.87e-04	**7.0539773e-04**	9.70e-05
8	**-4.1898293e+04**	1.06e-03	-2.1579648e+04	1.73e+03	-2.1209102e+04	2.98e+03	-3.9430820e+04	5.36e+02
9	**0.0000000e+00**	0.00e+00	2.4359139e+02	4.03e+01	4.8096522e+01	9.54e+00	9.9767318e-02	3.04e-01
10	**8.0232117e-15**	1.74e-15	4.4934316e+00	1.73e+00	5.6281044e-02	3.08e-01	2.9328603e-03	1.47e-04
11	**5.4210109e-20**	0.00e+00	4.1715080e-01	6.45e-01	8.5311042e-02	2.56e-01	1.8932321e-03	4.42e-03
12	**0.0000000e+00**	0.00e+00	1.1774980e-01	1.75e-01	9.2199219e-02	4.61e-01	2.9783067e-07	2.76e-08
13	**-1.1428244e+00**	2.74e-08	-3.8604485e-01	9.47e-01	3.3010679e+02	1.72e+03	-1.1428100e+00	2.41e-08

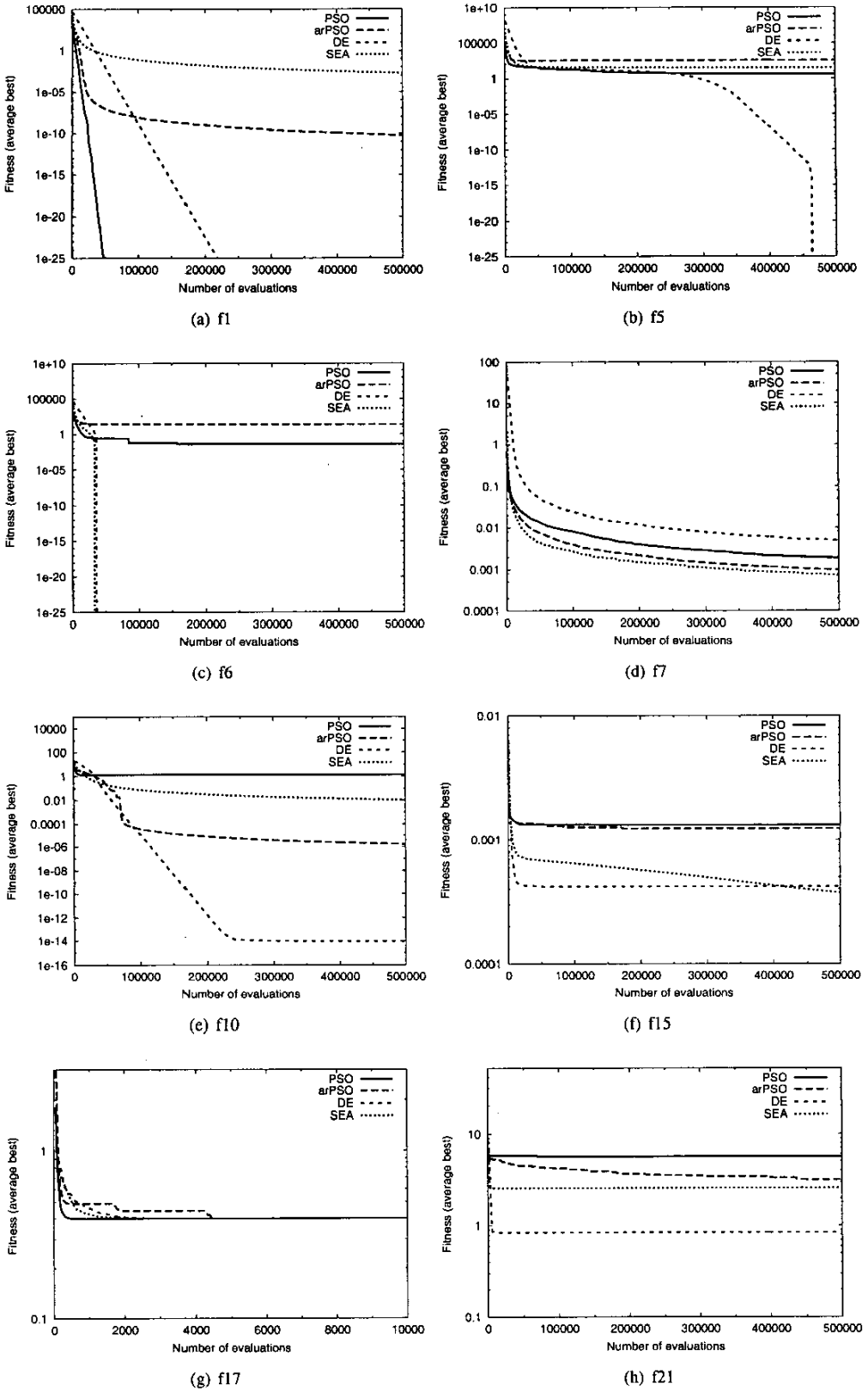

Fig. 2. Average best fitness curves for selected benchmark problems. All results are means of 30 runs.

To conclude, the performance of DE is outstanding in comparison to the other algorithms tested. It is simple, robust, converges fast, and finds the optimum in almost every run. In addition, it has few parameters to set, and the same settings can be used for many different problems. Previously, the DE has shown its worth on real-world problems, and in this study it outperformed PSO and EAs on the majority of the numerical benchmark problems as well. Among the tested algorithms, the DE can rightfully be regarded as an excellent first choice, when faced with a new optimization problem to solve. The results for the two noisy benchmark functions call for further investigations. More experiments are required to determine why and when the DE and PSO methods fail on noisy problems.

ACKNOWLEDGMENTS

The authors would like to thank Wouter Boomsma for proofreading the paper. Also, we would like to thank the colleagues at BiRC for valuable comments on early versions of the manuscript. This work has been supported by the Danish Research Council.

APPENDIX

f_{14} :

$$a = \begin{pmatrix} -32, -16, 0, 16, 32, \ldots, -32, -16, 0, 16, 32 \\ -32, \ldots, -16, \ldots, 0, \ldots, 16, \ldots, 32, \ldots \end{pmatrix}$$

f_{15} :

$$a = (0.1957, 0.1947, 0.1735, 0.1600, 0.0844, 0.0627, 0.0456, 0.0342, 0.0323, 0.0235, 0.0246)$$

$$b = \left(\tfrac{1}{0.25}, \tfrac{1}{0.5}, \tfrac{1}{1}, \tfrac{1}{2}, \tfrac{1}{4}, \tfrac{1}{6}, \tfrac{1}{8}, \tfrac{1}{10}, \tfrac{1}{12}, \tfrac{1}{14}, \tfrac{1}{16} \right)$$

f_{21}–f_{23} :

$$a = \begin{pmatrix} 4.0 & 4.0 & 4.0 & 4.0 \\ 1.0 & 1.0 & 1.0 & 1.0 \\ 8.0 & 8.0 & 8.0 & 8.0 \\ 6.0 & 6.0 & 6.0 & 6.0 \\ 3.0 & 7.0 & 3.0 & 7.0 \\ 2.0 & 9.0 & 2.0 & 9.0 \\ 5.0 & 5.0 & 3.0 & 3.0 \\ 8.0 & 1.0 & 8.0 & 1.0 \\ 6.0 & 2.0 & 6.0 & 2.0 \\ 7.0 & 3.6 & 7.0 & 3.6 \end{pmatrix}$$

$$c = \left(0.1, 0.2, 0.2, 0.4, 0.4, 0.6, 0.3, 0.7, 0.5, 0.5 \right)$$

REFERENCES

[1] P. J. Angeline. Evolutionary optimization versus particle swarm optimization: Philosophy and performance differences. In V. W. Porto, N. Saravanan, D. Waagen, and A. E. Eiben, editors, *Evolutionary Programming VII*, pp. 601–610. Springer, 1998.

[2] T. Bäck, D. B. Fogel, and Z. Michalewicz, editors. *Handbook of Evolutionary Computation*, chapter C2.3. Institute of Physics Publishing and Oxford University Press, 1997.

[3] L. J. Fogel, A. J. Owens, and M. J. Walsh. Artificial intelligence through a simulation of evolution. In M. Maxfield, A. Callahan, and L. J. Fogel, editors, *Biophysics and Cybernetic Systems: Proc. of the 2nd Cybernetic Sciences Symposium*, pp. 131–155. Spartan Books, 1965.

[4] Y. Fukuyama, S. Takayama, Y. Nakanishi, and H. Yoshida. A particle swarm optimization for reactive power and voltage control in electric power systems. In W. Banzhaf, J. Daida, A. E. Eiben, M. H. Garzon, V. Honavar, M. Jakiela, and R. E. Smith, editors, *Proceedings of the Genetic and Evolutionary Computation Conference*, pp. 1523–1528. Morgan Kaufmann Publishers, 1999.

[5] D. Gies and Y. Rahmat-Samii. Particle swarm optimization for reconfigurable phase-differentiated array design. *Microwave and Optical Technology Letters*, Vol. 38, No. 3, pp. 168–175, 2003.

[6] J. H. Holland. *Adpatation in Natural and Artificial Systems*. University of Michigan Press, Ann Arbor, MI, 1975.

[7] J. Kennedy and R. C. Eberhart. Particle swarm optimization. In *Proceedings of the 1995 IEEE International Conference on Neural Networks*, Vol. 4, pp. 1942–1948. IEEE Press, 1995.

[8] I. Rechenberg. *Evolution strategy: Optimization of technical systems by means of biological evolution*. Fromman-Holzboog, 1973.

[9] J. Riget and J. S. Vesterstrøm. A diversity-guided particle swarm optimizer – the arPSO. Technical report, EVALife, Dept. of Computer Science, University of Aarhus, Denmark, 2002.

[10] Y. Shi and R.C. Eberhart. A modified particle swarm optimizer. In *Proceedings of the IEEE Conference on Evolutionary Computation*, pp. 69–73. IEEE Press, 1998.

[11] R. Storn and K. Price. Differential evolution - a simple and efficient adaptive scheme for global optimization over continuous spaces. Technical report, International Computer Science Institute, Berkley, 1995.

[12] R. Thomsen. Flexible ligand docking using differential evolution. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Vol. 4, pp. 2354–2361. IEEE Press, 2003.

[13] R. Thomsen. Flexible ligand docking using evolutionary algorithms: investigating the effects of variation operators and local search hybrids. *BioSystems*, Vol. 72, No. 1-2, pp. 57–73, 2003.

[14] R. K. Ursem and P. Vadstrup. Parameter identification of induction motors using differential evolution. In *Proceedings of the 2003 Congress on Evolutionary Computation*, Vol. 2, pp. 790–796. IEEE Press, 2003.

[15] J. S. Vesterstrøm and J. Riget. Particle swarms: Extensions for improved local, multi-modal, and dynamic search in numerical optimization. Master's thesis, EVALife, Dept. of Computer Science, University of Aarhus, Denmark, 2002.

[16] X. Yao and Y. Liu. Fast evolutionary programming. In L. J. Fogel, P. J. Angeline, and T. Bäck, editors, *Proceedings of the 5th Annual Conference on Evolutionary Programming*, pp. 451–460. MIT Press, 1996.