Adapt-MEMPSODE: A Memetic Algorithm with Adaptive Selection of Local Searches

Costas Voglis
Department of Computer Science
University of Ioannina
GR–45110, Ioannina, Greece
voglis@cs.uoi.gr

ABSTRACT

MEMPSODE global optimization software tool integrates Particle Swarm Optimization, a prominent population–based stochastic algorithm, with well established efficient local search procedures. In the original description of the algorithm [17] a single local search with specific parameters was applied at selected best position vectors. In this work we present an adaptive variant of MEMPSODE where the local search is selected from a predefined pool of different algorithms. The choice of each local search is based on a probabilistic strategy that uses a simple metric to score the efficiency of the local search. This new version of the algorithm, Adapt-MEMPSODE, is benchmarked against BBOB 2013 test bed. The results show great improvement with respect to the static version that was also benchmarked in earlier workshop.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: Optimization—*global optimization, unconstrained optimization*; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Benchmarking, Black-box optimization, Memetic PSO, Adaptive selection, Local Search

1. INTRODUCTION

It is well established in the literature that two competing goals govern the design of a global search strategy: *exploration* ensures that every part of the domain will be covered and *exploitation* concentrates the search effort in a close neighbourhood of the so far best detected positions. Modern optimization algorithms achieve the goals by global and local optimization components. Such hybrid schemes defined within the broad family of Evolutionary Algorithms (e.g Genetic Algorithms) and Swarm Intelligence methods (Particle Swarm, Differential Evolution), are called *memetic algorithms*(MAs) [7, 5, 9].

Preliminary MA schemes applied single local optimization components throughout the global search [6]. Recently, the focus is concentrated on algorithms that take advantage of more than one local search. MAs that adaptively select from a pool of local search algorithms, are usually called *Adaptive MAs* [12]. Algorithms of this category can be divided according to the adaptation type as *static*, *adaptive* and *self-adaptive*. A static algorithm is not considering any feedback during the search to modify the selection mechanism. On the other hand, an adaptive algorithm uses on-line feedback to govern the selection of a local optimization algorithms. Self-adaptive approaches usually apply evolutionary operators to co-evolve the local searches. Besides the aforementioned classification, the adaptation may be *qualitative* if each local optimization algorithm can be simply characterized as *good* or *bad*, or *quantitative* if the exact value of the feedback is important to score the local optimization algorithm. According to their adaptation level, MAs can be further characterized as *external* if a problem-specific knowledge from past experience of the practitioner is exploited, *local* when only a part of the historical trace of the algorithm is used to adapt the decision, and *global* when the complete historical knowledge is used.

The present work extends MEMPSODE optimization software [17] by allowing adaptive selection of local searches. MEMSPODE software combines Unified Particle Swarm Optimization (UPSO) [14] algorithm with local search algorithms provided by the Merlin optimization environment [13]. We introduce a simple scoring scheme based on the performance of each individual local search and use a roulette-wheel algorithm to select, with higher probability, the most *effective* one. The probabilities for each local search are adapted during the search using information from past invocations. At regular intervals we reset the probabilities so that short term information is considered. The new variant of MEMPSODE falls in the category of *local-quantitative*, *adaptive* memetic schemes. Results from the BBOB 2013 test bed reveal that the new adaptive MEMPSODE scheme outperforms the original static version of MEMPSODE that uses only one local search at a time [19, 18].

The rest of the paper is organized as follows: in Section 2 we describe in detail our adaptive MA scheme. In Section 3,

GECCO'13 Companion, July 6–10, 2013, Amsterdam, The Netherlands.

we present the experimental setup , and Section 4 we present and comment the results on BBOB 2013 test bed.

2. ADAPTIVE MEMPSODE ALGORITHM

In this section we will present the adaptive modification to the original MEMPSODE algorithm [17, 15]. The original algorithm is extended by having multiple local searches that are stochastically selected following an adaptive scheme. This scheme is based on scoring the performance of each local search. The description of adapt-MEMPSODE is performed in four parts: (a) the Unified PSO algorithm that serves as the global exploration part, (b) the memetic strategy that determines *when* to apply a local search, (c) the adaptive selection strategy that determines *which* local search to choose and (d) the short description of the local optimization algorithms used in this study.

2.1 Unified PSO

Let the n-dimensional continuous optimization problem:

$$\min_{x \in X \subset \mathbb{R}^n} f(x), \tag{1}$$

where the search space X is an orthogonal hyperbox defined as:

$$X \equiv [l_1, r_1] \times [l_2, r_2] \times \cdots \times [l_n, r_n] \ \subset \ \mathbb{R}^n.$$

A *swarm* of N *particles* is defined as a set of search points:

$$S = \{x_1, x_2, \ldots, x_N\},$$

where each particle is an n-dimensional vector:

$$x_i = (x_{i1}, x_{i2}, \ldots, x_{in})^\top \in X, \qquad i \in \{1, 2, \ldots, N\}..$$

The particles move by assuming an adaptable position shift, called *velocity*, denoted as:

$$v_i = (v_{i1}, v_{i2}, \ldots, v_{in})^\top, \qquad i \in \{1, 2, \ldots, N\},$$

and they store the best position they have found,

$$p_i = (p_{i1}, p_{i2}, \ldots, p_{in})^\top \in X, \qquad i \in \{1, 2, \ldots, N\}.$$

in memory. If t denotes the iteration counter, the particles' positions and velocities are updated at each iteration as follows [1]:

$$\begin{aligned} v_{ij}^{(t+1)} &= \chi \left[v_{ij}^{(t)} + c_1 \mathcal{R}_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) \right. \\ &\quad \left. + c_2 \mathcal{R}_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right) \right], \qquad (2) \\ x_{ij}^{(t+1)} &= x_{ij}^{(t)} + v_{ij}^{(t+1)}, \qquad (3) \end{aligned}$$

where $i \in I \equiv \{1, 2, \ldots, N\}$ and $j = 1, 2, \ldots, n$. This is the *constriction coefficient* variant of PSO, named after the parameter χ in Eq. (2), which is used to restrict the magnitude of the velocities and it was derived through the stability analysis of PSO [1]. The rest of the parameters are the positive constants c_1 and c_2, also called *cognitive* and *social* parameter, respectively; and $\mathcal{R}_1$, $\mathcal{R}_2$, which are random numbers that differ for each i and j, drawn from a uniform distribution in the range $[0, 1]$.

The parameter g_i controls the information-sharing between the i-th particle and the rest. A neighborhood of the i-th particle is defined in the form of a set of indices of other particles $\mathcal{N}_i$. Thus, the parameter g_i is defined as:

$$g_i = \arg \min_{j \in \mathcal{N}_i} f(p_j).$$

Obviously, the topologies influence the flow of information among the particles and, hence, may affect the algorithm's efficiency and effectiveness. The special case where $\mathcal{N}_i = I$ for all $i \in \{1, 2, \ldots, N\}$, is called the *gbest* (global) PSO model, while all other cases with $\mathcal{N}_i \subset I$ define *lbest* (local) PSO models.

The best position of each particle is updated at each iteration as follows:

$$p_i^{(t+1)} = \begin{cases} x_i^{(t+1)}, & \text{if } f\left(x_i^{(t+1)}\right) < f\left(p_i^{(t)}\right), \\ p_i^{(t)}, & \text{otherwise.} \end{cases} \tag{4}$$

The *Unified PSO* (UPSO) algorithm generalizes the original PSO model by combining lbest and gbest velocity updates. UPSO stemmed from the speculation (supported by experimental evidence) that harnessing search directions with different exploration / exploitation properties can produce more efficient schemes. Let:

$$\begin{aligned} \mathcal{G}_{ij}^{(t+1)} &= \chi \left[v_{ij}^{(t)} + c_1 r_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left(p_{gj}^{(t)} - x_{ij}^{(t)} \right) \right], \\ \mathcal{L}_{ij}^{(t+1)} &= \chi \left[v_{ij}^{(t)} + c_1 r_1 \left(p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left(p_{g_i j}^{(t)} - x_{ij}^{(t)} \right) \right], \end{aligned}$$

denote the velocity update of x_i in the gbest and lbest PSO models, respectively, where g is the index of the overall best particle, i.e.:

$$g = \arg \min_{j \in \{1, 2, \ldots, N\}} f(p_j).$$

Then, UPSO's update equations are defined as in [14]:

$$\begin{aligned} \mathcal{U}_{ij}^{(t+1)} &= u\, \mathcal{G}_{ij}^{(t+1)} + (1 - u)\, \mathcal{L}_{ij}^{(t+1)}, \qquad (5) \\ x_{ij}^{(t+1)} &= x_{ij}^{(t)} + \mathcal{U}_{ij}^{(t+1)}, \qquad (6) \end{aligned}$$

where $i \in I$ and $j = 1, 2, \ldots, n$. The parameter $u \in [0, 1]$ is called *unification factor* and it balances the influence (tradeoff) of the gbest and lbest velocity update.

2.2 Memetic Strategy

The design process of a Memetic algorithm involves a number of parameters that control the *point (where)* and *frequency (when)* of application of the local search. Although additional information on the objective function may offer some guidance, in the general case these parameters are empirically determined based on a trial-and-error procedure. These parameters control a probability distribution that stochastically instructs the local search to begin from specific best positions.

Both MEMPSODE and the present adaptive variant are based on the scheme proposed for the Memetic PSO (MPSO) in [15] called *memetic strategies* and reported in Table 1. These schemes can be applied either at each iteration or whenever a specific number of consecutive iterations has been completed. To avoid unnecessary function evaluations, a local search is applied on a best position only if it has not been previously selected for local search or it has changed from the last application of local search. Note here that in all memetic strategies of Table 1 the local search is the same throughout the algorithm. In this work we extend the strategy by controlling *which* local search is going to be applied.

2.3 Adaptive selection strategy

In the present variant of MEMPSODE, we introduce a new adaptive selection strategy based on scoring the performance of each local search. Our selection algorithm is

Table 1: The memetic strategies of MEMPSODE

Scheme	Point of local search application
Scheme 1	p_g (overall best position)
Scheme 2	Each p_i, $i \in \{1, 2, \ldots, N\}$, with fixed probability $\rho \in (0, 1]$
Scheme 3	p_g and some randomly selected p_i, $i \in \{1, 2, \ldots, N\}$

based on *roulette selection* [12, 11] where the probabilities change during the algorithm's execution. The probabilities are identical for all particles (global context).

Roulette selection ensures that the number of applications of a specific local search is stochastic and proportional to a probability. More formally let $\mathcal{L}$ be a local search pool:

$$\mathcal{L} = \{LS^{(1)}, LS^{(2)}, \ldots, LS^{(k)}\},$$

where local search $LS^{(i)}$ is applied with probability $\mathcal{P}^{(i)}$, $i = 1, \ldots, k$. The probability is based on a scoring mechanism that is updated after the application of the local search. Also let $c^{(i)}$, $i = 1, \ldots, k$ count the number of applications of the $i-$th local search. In the $j-$th application of the local search $LS^{(i)}$, we define the $score_j^{(i)}$ as:

$$score_j^{(i)} = \frac{\frac{\left|\tilde{f}_j^{(i)} - \hat{f}_j^{(i)}\right|}{\left|\hat{f}_j^{(i)}\right|}}{\#evals_j^{(i)}}, \quad j = 1, \ldots, c^{(i)}, \quad i = 1, \ldots, k,$$

where $\tilde{f}_j^{(i)}$ is the objective function value before the application of $LS^{(i)}$, $\hat{f}_j^{(i)}$ is the approximation of the local minimum and $\#evals_j^{(i)}$ is the number of function evaluations spent during the local search.

The *average score*, $\mathcal{S}^{(i)}$, and the *probability*, $\mathcal{P}^{(i)}$ are then defined as:

$$\mathcal{S}^{(i)} = \frac{\sum_{j=1}^{c^{(i)}} score_j^{(i)}}{c^{(i)}}, \quad \mathcal{P}^{(i)} = \frac{\mathcal{S}^{(i)}}{\sum_{i=1}^{k} \mathcal{S}^{(i)}} \quad i = 1, \ldots, k,$$

The above probability is assigned to each local search subject to the roulette selection during the adaptive random scheme.

The presented adaptive calculation results in the following observations:

(a) The average score for a local search $LS^{(i)}$ is formed using global information constructed from the beginning of the algorithm. This may be restrictive since not all search areas of an objective function share the same morphology.

(b) If a local search does not perform well in early stages, then it may be assigned a low score and probability. In this case we may prematurely exclude a local search from the pool without properly assessing its performance.

The observations above guide us to split the adaptive random process into two phases, which are continuously repeated one after the other. In the first phase, called *training phase*, the algorithm collects information (average score) for the local searches but applies roulette selection with equal probabilities. After the training phase comes the *adaptive phase*, where the probabilities are adapted to the average score and also updated after every local search.

Algorithm 1: Pseudocode of Adapt-MEMPSODE.

Input: Objective function, $f : X \subset \mathbb{R}^n \to \mathbb{R}$; swarm size: N; unification factor: UF; probability for local search: ρ; Local search pool: $\mathcal{L} = \{LS^{(1)}, LS^{(2)}, \ldots, LS^{(k)}\}$; Training period : $\mathcal{K}$;
Output: Best detected solution: x^*, $f(x^*)$.

```
// Initialization
for i = 1, 2, ..., N do
    Initialize x_i and u_i
    Set p_i ← x_i // Initialize best position
    f_i ← f(x_i) // Evaluate particle
    f_{p_i} ← f_i // Best position value
    act_i ← 0// By default all particles perform FEs
end
for i = 1, 2, ..., k do
    c^(i) ← 0;  S^(i) ← 0;  P^(i) ← 1/k// Initialize
end
// Main Iteration Loop
Set t ← 0    ls ← 0
while (termination criterion does not hold) do
    // Determine which particles will apply LS on their best position
    for i = 1, 2, ..., N do
        if rand() < ρ then
            act_i ← 1 // The i-th particles will perform LSs
        else
            act_i ← 0 // The i-th particles will perform FEs
        end
    end
    // Update Best Indices
    Calculate global best index g_1 and local best index g_2
    // Update Swarm/Population
    for i = 1, 2, ..., N do
        Calculate lbest velocity update, L_i, using g_2 and u_i
        Calculate gbest velocity update, G_i, using g_1 and u_i
        u_i ← UF L_i + (1 − UF) G_i // Unified PSO
        x_i = x_i + u_i // Update particle's position
    end
    // Update Best Positions/Individuals
    for i = 1, 2, ..., N do
        if f_i < f_{p_i} then
            p_i ← x_i
            f_{p_i} ← f_i
        end
    end
    // Evaluate Population or Apply Local search
    for i = 1, 2, ..., N do
        if act_i = 0 then
            f_i ← f(x_i) // Perform FE
        else
            ← RouletteSelection(P)
            [p_i^+, f_{p_i}^+, f_evals] ← LS^(j)(p_i) // Perform LS
            score^(j) ← (|f_{p_i} − f_{p_i}^+| / |f_{p_i}^+|) / f_evals
            ls ← ls + 1;
            if train = 1 and mod(ls, K) = 0 then
                // Entering adaptive phase
                train ← 0
            else if train = 0 and mod(ls, 3 * K) = 0 then
                // Entering training phase
                train ← 1
                // Average scores and probabilities are reset
                for κ = 1, 2, ..., k do
                    c^(κ) ← 0;  S^(κ) ← 0;  P^(κ) ← 1/k
                end
            end
            UpdateProb(j, score, S, P, train)
        end
    end
end
```

Training phase takes place until $\mathcal{K}$ local searches are per-

formed and the adaptive phase follows for an integer multiple of $\mathcal{K}$. When the adaptive phase ends, all counters are reset, probabilities are equalized, and information is gathered from the start during a fresh new training phase. With this two–phase scheme, we manage to include a *short memory* in our adaptation, since very old information is now excluded. In addition, we give to all local searches the same probability to appear during the *construction* of the statistics.

The complete algorithmic scheme of our memetic algorithm is presented in Algorithm 1. The adaptation of the probabilities is presented in Procedure Updater.

Procedure Updater(j, score, $\mathcal{S}$, $\mathcal{P}$, train)

Input: Selected index: i ; Score for the selected: $score^{(i)}$; Score array: $\mathcal{S}$, Probability array: $\mathcal{P}$, Phase flag: *train*

Output: New score: $\mathcal{S}$, New probability: $\mathcal{P}$; New count: c

$\mathcal{S}^{(i)} \leftarrow \frac{c^{(i)}\mathcal{S}^{(i)} + \text{score}}{c^{(i)}+1}$ `// Update mean` $\mathcal{S}^{(i)}$

$c^{(i)} \leftarrow c^{(i)} + 1$ `// Update count` $c^{(i)}$

`// Not in training phase, update the probabilities`

if train $= 0$ **then**

  for $\kappa = 1, 2, \ldots, k$ **do**

    $\mathcal{P}^{(\kappa)} \leftarrow \frac{\mathcal{S}^{(\kappa)}}{\sum^{k} \mathcal{S}^{(\kappa)}}$

  end

end

2.4 Local Search Algorithms

In MEMPSODE [17] the PSO variant is coupled with local optimization methods from the robust Merlin optimization environment [13]. Merlin environment implements 11 local search algorithms and provides a rich variety of controlling options. For this work we chose a subset of 9 diverse local searches, all programmed either directly in the Merlin environment or via the plugin mechanism (see [13]).

Merlin also provides scripting capabilities that verify if a point is a local minimizer so that unnecessary local search applications can be avoided. In the remaining we briefly present the local optimization algorithms that constitute our pool. Henceforth, a local search approach will be denoted as $LS^{(k)}$, and we assume that it returns the approximation of the minimizer, its function value, and the required number of function evaluations.

Whenever first order derivative information (gradient) is required, it is calculated via finite differences using [16]. For the default values of the parameters of each algorithm, please refer to Merlin user manual distributed with the software. In the following description of local searches we provide specific Merlin parameters in parentheses. `nock` stands for the maximum number of function evaluations.

2.4.1 *BFGS* (`noc 2000`)

A well-known quasi-Newton method with line search [10] that uses the BFGS update formula [2] to approximate the Hessian matrix. The update is performed directly on the Cholesky factors of the Hessian.

2.4.2 *Simplex* (`noc 2000 disp 0.5`)

Designed by Nelder and Mead [8] the Simplex algorithm is based on the concept of *simplex* (or polytope) in $\mathbb{R}^n$, which is a volume element defined by $(n+1)$ vertices. This volume element is stretched, expanded, reflected in pursuit of the minimizer. The initial simplex is created using a displacement of 50% with respect to the bounding box along each direction (`(disp 0.5)`).

2.4.3 *Roll* (`noc 2000`)

This method belongs to the class of pattern search methods. It proceeds by taking proper steps along each coordinate direction, in turn. Then, the method performs an one–dimensional search on a properly formed direction, in order to tackle possible correlations among the variables.

2.4.4 *Congra* (`noc 2000`)

This is an implementation of the Conjugate Gradient method for unconstrained minimization. This method generates a set of conjugate search directions and applies line search.

2.4.5 *Auto* (`noc 2000`)

AUTO is a meta–local search procedure that attempts to select the best LS algorithm among 5 already implemented in Merlin. The methods BFGS, ROLL, SIMPLEX and TRUST are invoked one after the other. For each method, a rate is calculated by dividing the relative achieved reduction of the function's value by the number of function calls spent. The method with the highest rate is then invoked again and the procedure is repeated.

2.4.6 *Rand* (`noc 2000 red 0.9`)

We added a random search component to our arsenal of local searches in order to tackle highly discontinuous problems with low or no structure. In every iteration a random step is taken from a uniform distribution inside a predefined box. If the step leads to a smaller function value, it is accepted. Subsequent rejections reduce the sampling box size, hence leading to smaller random steps. In our case, we start with random step inside a box of magnitude up to 5.0, reduced by 10% (`(red 0.9)`) every 50 consecutive rejections.

2.4.7 *Hooke and Jeeves with discrete steps* (`noc 2000 ls 0`)

The original pattern search method of Hooke and Jeeves [4] consists of a sequence of exploratory moves about a base point which, if successful, are followed by pattern move.

2.4.8 *Hooke and Jeeves with line search* (`noc 2000 ls 1`)

A modification of the original Hooke& Jeeves algorithm where discrete steps are replaced by line searches. The `ls 1` option switches to the line search version.

2.4.9 *Tolmin* (`noc 2000`)

Another implementation of the BFGS method with line search that uses the Goldfarb-Idnani factorization [3] of the Hessian.

3. EXPERIMENTAL PROCEDURE

MEMPSODE was applied for a maximum of $5 \times 10^5 \times n$ function evaluations per run. Whenever it terminated before reaching the global minimum (e.g., when LS resulted in a

local minimizer), we performed an independent restart until the maximum number of function evaluations was reached.

The memetic Scheme 2 was applied with probability of local search set to $\rho_i = 0.05$.We used a swarm size $N = 30$ particles. The unification factor u was set to 1 (gbest) and the initial velocity vector was restraint by a factor of 0.01. Each local search assumed a maximum number of 2000 function evaluations. Input options for the local searches are presented in verbatim in parentheses after each subsection title.

The experiments were conducted on an Intel I7-2600 3.4 GHz machine with 8GB RAM using GNU compiler suite v.4.4.3.

4. CPU TIMING EXPERIMENT

For the timing experiment the experimental procedure described above was run on f8 with at most 1000 function evaluations in each call to adapt-MEMPSODE and restarted until at least 30 seconds had passed. The timing experiment was performed on the same platform as the experimental procedure. The results for the UPSO variant were 3.0, 2.1, 1.5, 0.8, 0.66, and 0.53 times 10^{-4} seconds per function evaluation for dimensions 2, 3, 5, 10, 20, and 40, respectively.

5. RESULTS

Results using the BBOB post-processing scripts are presented in Figures 1, 2 and 3 and in Tables 2 and 3. Compared to the results of MEMPSODE in BBOB 2012 [19, 18] we witness a large improvement of the new adaptive scheme against the original MEMPSODE. From Figure 2 we can see that the new adaptive scheme solves, with accuracy 10^{-8}, almost 90% of the 5-dimensional and 40% of the 20-dimensional cases. Original MEMPSODE on the other hand solved 50% of the 5-dimensional and less that 10% of the 20-dimensional cases in BBOB 2012 testbed. From Figure 1 we can see that adapt-MEMPSODE achieves very good ERTs for all dimensions, close to the result from BBOB-2009. These results are confirmed from Figure 2 where we can see that in the 5-dimensional case adapt-MEMPSODE has the fourth best cumulative distribution line. In the 20-dimensional case we still score above the average.

The proposed adapt-MEMPSODE algorithm exploits the diversity provided by multiple local searches and achieves better exploration of the search space. This behaviour can be justified by the fact that different local searches may approximate different minimizers even if they begin from the same starting point. As a consequence, the stochastic selection of local search algorithm increases the coverage of search area.

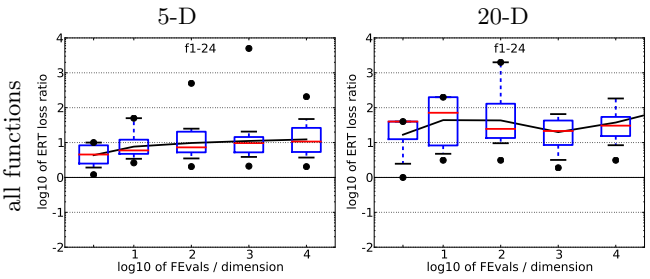

Figure 3: ERT loss ratio vs. a given budget FEvals. Each cross (+) represents a single function. The target value f_t used for a given FEvals is the smallest (best) recorded function value such that ERT(f_t) $\leq$ FEvals for the presented algorithm. Shown is FEvals divided by the respective best ERT(f_t) from BBOB-2009 for functions f_1–f_{24} in 5-D and 20-D.

Table 3: ERT loss ratio compared to the respective best result from BBOB-2009 for budgets given in the first column (see also Figure 3). The ERT Loss ratio equals to one for the respective best algorithm from BBOB-2009.

	f_1–f_{24} **in 5-D**, maxFE/D=501108					
#FEs/D	best	10%	25%	**med**	75%	90%
2	1.2	1.9	2.4	4.4	8.3	10
10	2.6	3.4	4.6	5.8	11	50
100	2.0	3.5	4.7	7.2	20	26
1e3	2.1	3.9	5.2	8.7	14	23
1e4	2.0	3.7	5.3	10	23	52
1e5	2.0	4.0	5.1	11	23	63
RL_{US}/D	5e5	5e5	5e5	5e5	5e5	5e5
	f_1–f_{24} **in 20-D**, maxFE/D=500609					
#FEs/D	best	10%	25%	**med**	75%	90%
2	1.0	2.4	10	34	40	40
10	3.1	4.7	7.9	69	2.0e2	2.0e2
100	3.1	9.5	13	23	1.0e2	2.0e3
1e3	1.9	3.2	8.3	18	40	67
1e4	3.1	8.0	14	30	54	1.8e2
1e5	1.1	7.9	27	95	1.7e2	7.3e2
1e6	3.1	8.3	27	2.0e2	8.7e2	1.5e3
RL_{US}/D	5e5	5e5	5e5	5e5	5e5	5e5

6. REFERENCES

[1] M. Clerc and J. Kennedy. The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Trans. Evol. Comput.*, 6(1):58–73, 2002.

[2] R. Fletcher. A new approach to variable metric algorithms. *The Computer Journal*, 13(3):317–322, 1970.

[3] D. Goldfarb and A. Idnani. A numerically stable dual method for solving strictly convex quadratic programs. *Mathematical programming*, 27(1):1–33, 1983.

[4] R. Hooke and T. A. Jeeves. "direct search"solution of numerical and statistical problems. *Journal of the ACM (JACM)*, 8(2):212–229, 1961.

[5] N. Krasnogor and J. Smith. A tutorial for competent memetic algorithms: Model, taxonomy and design issues. *IEEE Trans. Evol. Comput.*, 9(5):474–488, 2005.

[6] P. Moscato. On evolution, search, optimization, genetic algorithms and martial arts: Towards memetic algorithms. Technical Report C3P Report 826, Caltech Concurrent Computation Program, California, USA, 1989.

[7] P. Moscato. Memetic algorithms: A short introduction. In D. Corne, M. Dorigo, and F. Glover, editors, *New Ideas in Optimization*, pages 219–235. McGraw–Hill, London, 1999.

[8] J. Nelder and R. Mead. A simplex method for

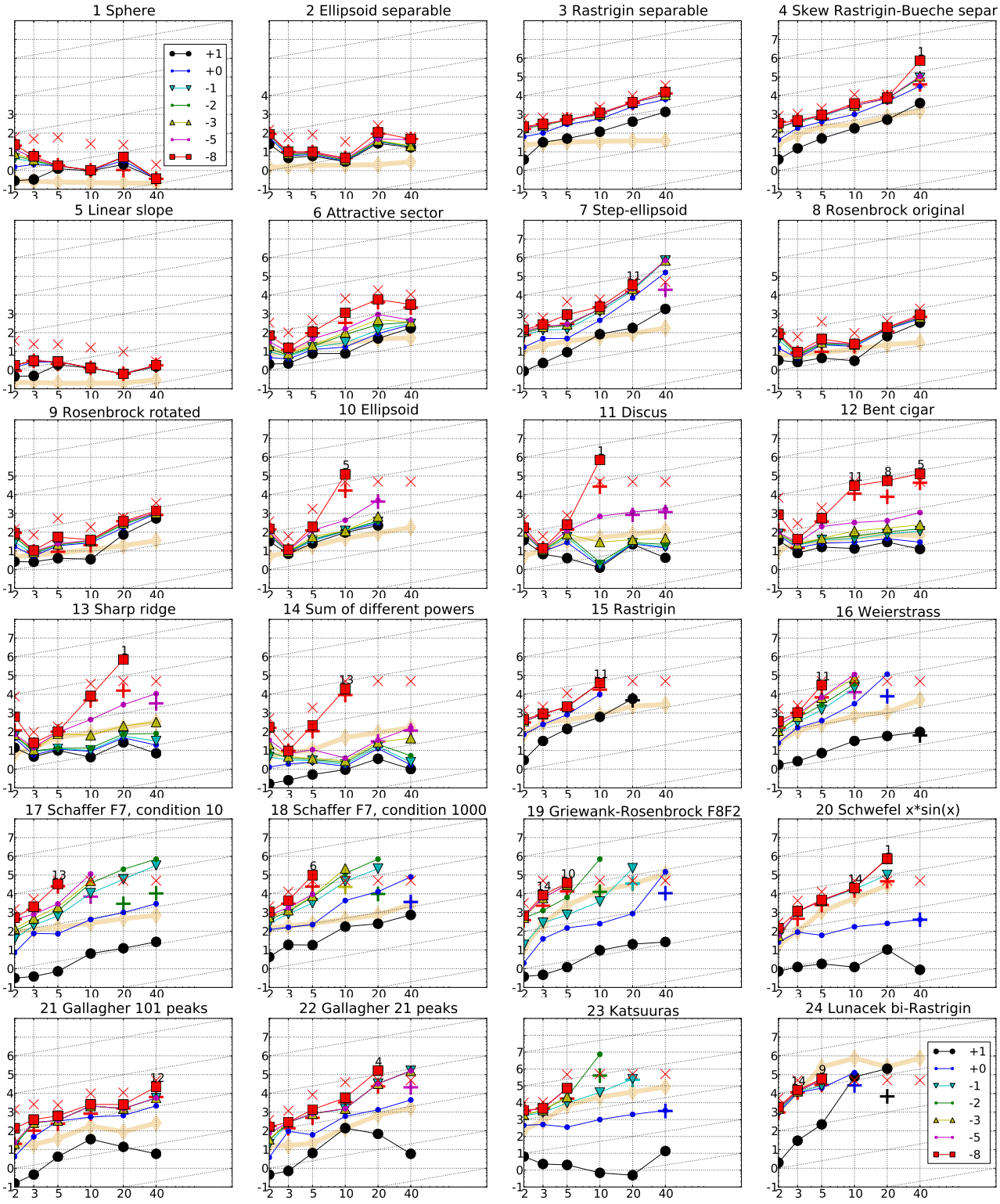

Figure 1: Expected number of f-evaluations (ERT, lines) to reach $f_{\mathrm{opt}} + \Delta f$; median number of f-evaluations (+) to reach the most difficult target that was reached not always but at least once; maximum number of f-evaluations in any trial (×); interquartile range with median (notched boxes) of simulated runlengths to reach $f_{\mathrm{opt}} + \Delta f$; all values are divided by dimension and plotted as $\log_{10}$ values versus dimension. Shown are $\Delta f = 10^{\{1,0,-1,-2,-3,-5,-8\}}$. Numbers above ERT-symbols (if appearing) indicate the number of trials reaching the respective target. The light thick line with diamonds indicates the respective best result from BBOB-2009 for $\Delta f = 10^{-8}$. Horizontal lines mean linear scaling, slanted grid lines depict quadratic scaling.

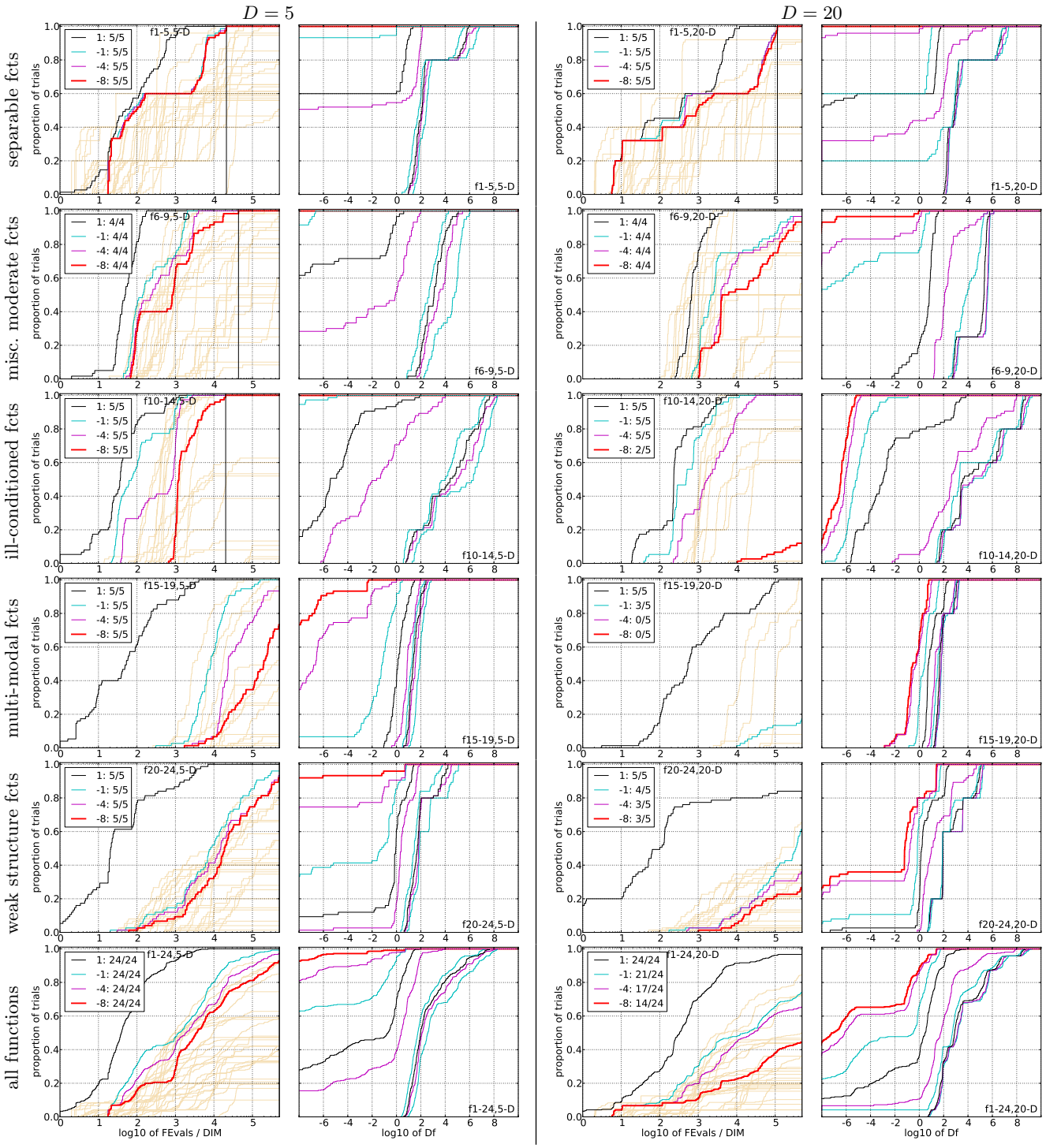

Figure 2: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials with an outcome not larger than the respective value on the x-axis. Left subplots: ECDF of number of function evaluations (FEvals) divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. Right subplots: ECDF of the best achieved Δf divided by 10^{-8} for running times of $D, 10\,D, 100\,D, \ldots$ function evaluations (from right to left cycling black-cyan-magenta). The thick red line represents the most difficult target value $f_{\mathrm{opt}} + 10^{-8}$. Legends indicate the number of functions that were solved in at least one trial. Light brown lines in the background show ECDFs for $\Delta f = 10^{-8}$ of all algorithms benchmarked during BBOB-2009.

5-D

Δf	1e+1	1e+0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f_1	11	12	12	12	12	12	12	15/15
	5.7(4)	7.3	7.3	7.3	7.3	7.3	7.5(0.2)	15/15
f_2	83	87	88	89	90	92	94	15/15
	3.6(3)	4.1(3)	4.5(3)	4.7(3)	4.8(3)	5.0(3)	5.2(3)	15/15
f_3	716	1622	1637	1642	1646	1650	1654	15/15
	3.6(3)	9.2(6)	15(6)	15(6)	15(6)	15(6)	15(6)	15/15
f_4	809	1633	1688	1758	1817	1886	1903	15/15
	3.2(3)	12(4)	23(16)	25(22)	24(21)	23(21)	24(23)	15/15
f_5	10	10	10	10	10	10	10	15/15
	10(6)	14(5)	14(6)	14(6)	14(6)	14(6)	14(6)	15/15
f_6	114	214	281	404	580	1038	1332	15/15
	3.3(3)	2.9(2)	2.8(2)	2.3(1)	1.8(1)	2.2(2)	3.7(1)	15/15
f_7	24	324	1171	1451	1572	1572	1597	15/15
	19(14)	7.5(6)	6.5(2)	8.5(2)	9.0(2)	9.0(2)	17(12)	15/15
f_8	73	273	336	372	391	410	422	15/15
	3.0(1)	4.1(5)	3.6(4)	3.4(4)	3.8(5)	3.7(5)	3.8(5)	15/15
f_9	35	127	214	263	300	335	369	15/15
	5.9(2)	7.5(11)	4.8(7)	4.1(6)	3.8(6)	3.6(6)	3.4(5)	15/15
f_{10}	349	500	574	607	626	829	880	15/15
	3.7(7)	3.5(6)	3.2(5)	3.5(5)	4.6(8)	7.5(9)	11(8)	15/15
f_{11}	143	202	763	977	1177	1467	1673	15/15
	1.4(0.2)	7.0(11)	3.5(3)	3.7(2)	3.4(2)	4.1(0.6)	5.4(2)	15/15
f_{12}	108	268	371	413	461	1303	1494	15/15
	7.4(12)	4.8(5)	5.2(7)	5.3(6)	5.0(6)	7.7(6)	11(11)	15/15
f_{13}	132	195	250	319	1310	1752	2255	15/15
	3.8(9)	3.0(9)	2.5(7)	2.1(5)	2.9(0.5)	2.6(0.5)	2.1(0.4)	15/15
f_{14}	10	41	58	90	139	251	476	15/15
	2.6(2)	2.9(0.4)	2.4(0.2)	1.8(0.2)	1.4(0.1)	2.2(0.1)	23(2)	15/15
f_{15}	511	9310	19369	19743	20073	20769	21359	14/15
	14(15)	4.4(4)	5.6(4)	5.5(4)	5.4(4)	5.2(4)	5.1(4)	15/15
f_{16}	120	612	2662	10163	10449	11644	12095	15/15
	3.0(2)	32(31)	28(12)	13(13)	31(17)	29(15)	95(121)	11/15
f_{17}	5.2	215	899	2861	3669	6351	7934	15/15
	6.9(4)	17(20)	36(13)	22(35)	27(33)	23(19)	104(111)	13/15
f_{18}	103	378	3968	8451	9280	10905	12469	15/15
	8.9(6)	30(31)	40(87)	41(53)	41(49)	44(46)	308(347)	6/15
f_{19}	1	1	242	1.0e5	1.2e5	1.2e5	1.2e5	15/15
	60(24)	7494(7112)	154(80)	3.1(2)	13(13)	13(13)	16(20)	10/15
f_{20}	16	851	38111	51362	54470	54861	55313	14/15
	5.7(2)	3.6(5)	5.5(8)	4.1(6)	3.9(5)	3.9(5)	4.2(5)	15/15
f_{21}	41	1157	1674	1692	1705	1729	1757	14/15
	5.0(5)	14(15)	11(11)	11(10)	10(10)	10(10)	14(22)	15/15
f_{22}	71	386	938	980	1008	1040	1068	14/15
	4.6(3)	7.9(10)	42(36)	42(35)	41(34)	40(33)	52(44)	15/15
f_{23}	3.0	518	14249	27890	31654	33030	34256	15/15
	3.4(3)	3.4(3)	3.4(3)	3.8(4)	3.5(4)	11(22)	10(21)	15/15
f_{24}	1622	2.2e5	6.4e6	9.6e6	9.6e6	1.3e7	1.3e7	3/15
	6.8(7)	4.4(7)	0.20(0.3)	0.27(0.2)	0.27(0.3)	0.20(0.2)	0.23(0.2)	9/15

20-D

Δf	1e+1	1e+0	1e-1	1e-2	1e-3	1e-5	1e-7	#succ
f_1	43	43	43	43	43	43	43	15/15
	10(7)	16(14)	23(23)	25(25)	25(25)	25(25)	25(25)	15/15
f_2	385	386	387	388	390	391	393	15/15
	15(11)	18(10)	18(10)	19(11)	21(11)	25(8)	45(27)	15/15
f_3	5066	7626	7635	7637	7643	7646	7651	15/15
	16(11)	68(31)	110(49)	110(48)	110(49)	110(48)	116(48)	15/15
f_4	4722	7628	7666	7686	7700	7758	1.4e5	9/15
	22(12)	127(96)	183(101)	183(100)	182(100)	181(99)	10(5)	15/15
f_5	41	41	41	41	41	41	41	15/15
	3.1(0.3)	3.1(0.3)	3.1(0.3)	3.1(0.3)	3.1(0.3)	3.1(0.3)	3.1(0.3)	15/15
f_6	1296	2343	3413	4255	5220	6728	8409	15/15
	7.6(3)	7.6(3)	7.8(4)	13(8)	19(8)	28(16)	63(58)	15/15
f_7	1351	4274	9503	16523	16524	16524	16969	15/15
	26(21)	340(370)	369(537)	247(333)	266(317)	266(309)	417(446)	11/15
f_8	2039	3871	4040	4148	4219	4371	4484	15/15
	6.4(3)	7.8(5)	8.2(5)	8.2(5)	8.6(5)	8.7(6)	8.5(6)	15/15
f_9	1716	3102	3277	3379	3455	3594	3727	15/15
	8.8(3)	11(5)	15(6)	16(8)	17(8)	19(4)	20(2)	15/15
f_{10}	7413	8661	10735	13641	14920	17073	17476	15/15
	6.2(4)	6.8(3)	6.8(5)	6.0(4)	9.2(7)	59(61)	∞*1.0e7*	0/15
f_{11}	1002	2228	6278	8586	9762	12285	14831	15/15
	4.5(1.0)	2.2(0.2)	0.81(0.1)	**0.64**$(0.1)^{\downarrow 4}$	0.81(0.2)	20(16)	9880(10790)	0/15
f_{12}	1042	1938	2740	3156	4140	12407	13827	15/15
	5.7(4)	4.6(3)	5.8(3)	6.5(5)	8.0(5)	6.6(7)	124(224)	8/15
f_{13}	652	2021	2751	3507	18749	24455	30201	15/15
	8.2(2)	4.7(2)	4.2(2)	4.3(0.4)	2.2(1)	23(21)	367(415)	1/15
f_{14}	75	239	304	451	932	1648	15661	15/15
	10(14)	11(6)	11(6)	10(3)	5.9(1)	4.3(0.7)	∞*1.0e7*	0/15
f_{15}	30378	1.5e5	3.1e5	3.2e5	3.2e5	4.5e5	4.6e5	15/15
	38(23)	∞	∞	∞	∞	∞	∞*1.0e7*	0/15
f_{16}	1384	27265	77015	1.4e5	1.9e5	2.0e5	2.2e5	15/15
	8.5(9)	869(1023)	∞	∞	∞	∞	∞*1.0e7*	0/15
f_{17}	63	1030	4005	12242	30677	56288	80472	15/15
	39(22)	194(137)	2971(3814)	3341(4122)	∞	∞	∞*1.0e7*	0/15
f_{18}	621	3972	19561	28555	67569	1.3e5	1.5e5	15/15
	81(61)	659(1260)	2243(2302)	4974(5344)	∞	∞	∞*1.0e7*	0/15
f_{19}	1	1	3.4e5	4.7e6	6.2e6	6.7e6	6.7e6	15/15
	4084(4842)	1.8e5(1e5)	134(135)	∞	∞	∞	∞*1.0e7*	0/15
f_{20}	82	46150	3.1e6	5.5e6	5.5e6	5.6e6	5.6e6	14/15
	26(15)	1.1(0.8)	6.7(7)	27(31)	27(28)	27(29)	27(27)	1/15
f_{21}	561	6541	14103	14318	14643	15567	17589	15/15
	4.9(7)	20(33)	22(25)	21(24)	21(24)	20(22)	19(20)	15/15
f_{22}	467	5580	23491	24163	24948	26847	1.3e5	12/15
	29(47)	47(93)	279(374)	272(364)	263(352)	245(331)	75(97)	4/15
f_{23}	3.2	1614	67457	3.7e5	4.9e5	8.1e5	8.4e5	15/15
	3.1(3)	25(20)	75(60)	∞	∞	∞	∞*1.0e7*	0/15
f_{24}	1.3e6	7.5e6	5.2e7	5.2e7	5.2e7	5.2e7	5.2e7	3/15
	31(34)	∞	∞	∞	∞	∞	∞*1.0e7*	0/15

Table 2: Expected running time (ERT in number of function evaluations) divided by the best ERT measured during BBOB-2009 (given in the respective first row) for different Δf values for functions f_1–f_{24}. The median number of conducted function evaluations is additionally given in *italics*, if $\mathrm{ERT}(10^{-7}) = \infty$. #succ is the number of trials that reached the final target $f_{\mathrm{opt}} + 10^{-8}$.

function minimization. *The computer journal*, 7(4):308–313, 1965.

[9] F. Neri, C. Cotta, and P. Moscato, editors. *Handbook of Memetic Algorithms*. Springer, 2011.

[10] J. Nocedal and S. Wright. *Numerical optimization*. Springer verlag, 1999.

[11] Y.-S. Ong and A. J. Keane. Meta-lamarkian learning in memetic algorithms. *IEEE Transactions on Evolutionary Computation*, 8(2):99–110, 2004.

[12] Y.-S. Ong, M.-H. Lim, N. Zhu, and K.-K. Wong. Classification of adaptive memetic algorithms: A comparative study. *IEEE Transactions on systems, man, and cybernetics*, 36(1):141–152, 2006.

[13] D. G. Papageorgiou, I. N. Demetropoulos, and I. E. Lagaris. Merlin-3.1. 1. a new version of the Merlin optimization environment. *Computer Physics Communications*, 159(1):70–71, 2004.

[14] K. E. Parsopoulos and M. N. Vrahatis. UPSO: A unified particle swarm optimization scheme. In *Lecture Series on Computer and Computational Sciences, Vol. 1, Proceedings of the International Conference of Computational Methods in Sciences and Engineering (ICCMSE 2004)*, pages 868–873. VSP International Science Publishers, Zeist, The Netherlands, 2004.

[15] Y. G. Petalas, K. E. Parsopoulos, and M. N. Vrahatis. Memetic particle swarm optimization. *Annals of Operations Research*, 156(1):99–127, 2007.

[16] C. Voglis, P. Hadjidoukas, I. Lagaris, and D. Papageorgiou. A numerical differentiation library exploiting parallel architectures. *Computer Physics Communications*, 180(8):1404–1415, 2009.

[17] C. Voglis, K. Parsopoulos, D. Papageorgiou, I. Lagaris, and M. Vrahatis. Mempsode: A global optimization software based on hybridization of population-based algorithms and local searches. *Computer Physics Communications*, 183(5):1139–1154, 2012.

[18] C. Voglis, G. S. Piperagkas, K. E. Parsopoulos, D. G. Papageorgiou, and I. E. Lagaris. MEMPSODE: An empirical assessment of local search algorithm impact on a memetic algorithm using noiseless testbed. In *GECCO'12(Companion)*, pages 245–252, Philadelphia (PA), USA, 2012.

[19] C. Voglis, G. S. Piperagkas, K. E. Parsopoulos, D. G. Papageorgiou, and I. E. Lagaris. MEMPSODE: Comparing particle swarm optimization and differential evolution on a hybrid memetic global optimization framework. In *GECCO'12(Companion)*, pages 253–260, Philadelphia (PA), USA, 2012.