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. MEMPSODE 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,
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 adap-MEMPSODE is performed in four parts: (a) the Unified PSO algorithm that serves as the global exploration part, (b) the memetic strategy that determines which local search to choose and (c) the adaptive selection strategy that determines the global exploration part, 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:

$$\min_{x \in \mathbb{R}^n} f(x),$$

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:

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

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

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

and they store the best position they have found,

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

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

$$v_{ij}^{(t+1)} = \chi \left[ v_{ij}^{(t)} + c_1 r_1 \left( p_{ij}^{(t)} - x_{ij}^{(t)} \right) \right] + c_2 r_2 \left( p_{gij}^{(t)} - x_{ij}^{(t)} \right), \quad (2)$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + v_{ij}^{(t+1)}, \quad (3)$$

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 $\chi$ 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 $R_1$, $R_2$, 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 $N_i$. Thus, the parameter $g_i$ is defined as:

$$g_i = \arg \min_{j \in N_i} f(p_j).$$

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 $N_i = I$ for all $i \in \{1, 2, \ldots, N \}$, is called the gbest (global) PSO model, while all other cases with $N_i \subseteq I$ define lbest (local) PSO models.

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

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

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:

$$g_{ij}^{(t+1)} = \chi \left[ x_{ij}^{(t)} + c_1 r_1 \left( p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left( p_{gij}^{(t)} - x_{ij}^{(t)} \right) \right],$$

$$l_{ij}^{(t+1)} = \chi \left[ x_{ij}^{(t)} + c_1 r_1 \left( p_{ij}^{(t)} - x_{ij}^{(t)} \right) + c_2 r_2 \left( p_{gij}^{(t)} - x_{ij}^{(t)} \right) \right],$$

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]:

$$u_{ij}^{(t+1)} = u g_{ij}^{(t+1)} + (1 - u) l_{ij}^{(t+1)},$$

$$x_{ij}^{(t+1)} = x_{ij}^{(t)} + u_{ij}^{(t+1)} + x_{ij}^{(t)}, \quad (5),$$

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 (trade-off) 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

<table>
<thead>
<tr>
<th>Scheme 1</th>
<th>Point of local search application</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scheme 1</td>
<td>( p_b ) (overall best position)</td>
</tr>
<tr>
<td>Scheme 2</td>
<td>Each ( p_i, i \in {1, 2, \ldots, N} ), with fixed probability ( \rho \in (0, 1] )</td>
</tr>
<tr>
<td>Scheme 3</td>
<td>( p_b ) and some randomly selected ( p_i, i \in {1, 2, \ldots, N} )</td>
</tr>
</tbody>
</table>

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 \( L \) be a local search pool:

\[
L = \{LS^{(1)}, LS^{(2)}, \ldots, LS^{(k)}\},
\]

where local search \( LS^{(i)} \) is applied with probability \( p^{(i)} \), \( i = 1, \ldots, k \). 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 \( e^{(i)}_j \) as:

\[
\text{score}^{(i)}_j = \frac{|f^{(i)}_j - f^{(j)}_0|}{\#\text{fevals}^{(i)}_j}, \quad j = 1, \ldots, c^{(i)}, \quad i = 1, \ldots, k,
\]

where \( f^{(j)}_0 \) is the objective function value before the application of \( LS^{(i)} \), \( f^{(j)}_0 \) is the approximation of the local minimum and \( \#\text{fevals}^{(i)}_j \) is the number of function evaluations spent during the local search.

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

\[
S^{(i)} = \frac{\sum_{j=1}^{c^{(i)}} \text{score}^{(i)}_j}{c^{(i)}}, \quad p^{(i)} = \frac{S^{(i)}}{\sum_{i=1}^{k} S^{(i)}}, \quad i = 1, \ldots, k.
\]

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, \( f : X \subseteq \mathbb{R}^d \rightarrow \mathbb{R} \); swarm size: \( N \); unification factor: \( UF \); probability for local search: \( \rho \); Local search pool: \( L = \{LS^{(1)}, LS^{(2)}, \ldots, LS^{(k)}\} \);
Training period: \( K \);
Output: Best detected solution: \( x^* \), \( f(x^*) \).

// Initialization
1. for \( i = 1, 2, \ldots, N \) do
   2. Initialize \( x_i \) and \( u_i \);
   3. Set \( p_i \leftarrow x_i \) // Initialize best position
   4. \( f_i \leftarrow f(x_i) \) // Evaluate particle
   5. \( f_{p_i} \leftarrow f_i \) // Best position value
   6. \( act_i \leftarrow 0 \) // by default all particles perform FE
   7. end
12. while (termination criterion does not hold) do
   13. // Determine which particles will apply LS on their best position
   14. for \( i = 1, 2, \ldots, N \) do
      15. if \( rand() < \rho \) then
         16. \( act_i \leftarrow 1 \) // The i-th particles will perform LSs
      17. else
         18. \( act_i \leftarrow 0 \) // The i-th particles will perform FE
      19. end
   20. end
   21. // Main Iteration Loop
   22. Set \( t \leftarrow 0 \) \( ls \leftarrow 0 \)
12. while (termination criterion does not hold) do
   23. // Determine which particles will apply LS on their best position
   24. for \( i = 1, 2, \ldots, N \) do
      25. \( f_i \leftarrow f(x_i) \)
      26. \( p_i \leftarrow p_i \)
      27. \( f_{p_i} \leftarrow f_{p_i} \)
   28. end
   29. if \( f_i < f_{p_i} \) then
      30. \( p_i \leftarrow x_i \)
      31. \( f_{p_i} \leftarrow f_i \)
   32. end
   33. // Main Iteration Loop
   34. // Evaluate Population or Apply Local search
   35. for \( i = 1, 2, \ldots, N \) do
      36. if \( act_i = 0 \) then
         37. \( f_i \leftarrow f(x_i) \) // Perform FE
      38. else
         39. \( \leftarrow \text{RouletteSelection}(P) \)
         40. \( [p_b^*, f_{p_b^*}, \text{fevals}] \leftarrow LS^{(j)}(p_i) \) // Perform LS
         41. score\(_{(i)} \leftarrow \left| f_{p_b^*} - f_{p_b}^* \right| / \text{fevals} \)
         42. \( ls \leftarrow ls + 1 \)
         43. if \( train = 1 \) and \( mod(ls, K) = 0 \) then
            44. // Entering adaptive phase
            45. \( \text{train} \leftarrow 0 \) \( mod(ls, 3 * K) = 0 \) then
               46. // Entering training phase
               47. \( \text{train} \leftarrow 1 \)
               48. // Average scores and probabilities are reset
               49. for \( k = 1, 2, \ldots, k \) do
                  50. \( \text{score}(x) \leftarrow 0 \) \( \text{pre}(x) \leftarrow 0 \)
               end
            end
            51. end
         end
      else if \( train = 0 \) then
         52. // Entering training phase
         53. \( \text{train} \leftarrow 1 \)
         54. // Average scores and probabilities are reset
         55. for \( k = 1, 2, \ldots, k \) do
            56. \( \text{score}(x) \leftarrow 0 \) \( \text{pre}(x) \leftarrow 0 \)
         end
      end
   57. end
58. end
59. end
60.

Training phase takes place until \( K \) local searches are per-
formed and the adaptive phase follows for an integer multiple of $K$. 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

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

**Output:**  
- New score: $S$; New probability: $P$; New count: $c$  

1. \[ S^{(i)} \leftarrow \frac{c^{(i)} S^{(i)} + \text{score}^{(i)}}{c^{(i)} + 1} \]  
2. \[ c^{(i)} \leftarrow c^{(i)} + 1 \]

// Not in training phase, update the probabilities

3. **if** $\text{train} = 0$ **then**

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

5. \[ P^{(\kappa)} \leftarrow \frac{S^{(\kappa)}}{\sum_{k}^{k} S^{(\kappa)}} \]

6. **end**

7. **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. noc stands for the maximum number of function evaluations.

#### 2.4.1 BFGS (noc 2000)


#### 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 \( p_l = 0.05 \). 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 \( 10^{-8} \) 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.

6. REFERENCES


Figure 1: Expected number of $f$-evaluations (ERT, lines) to reach $f_{\text{opt}} + \Delta f$; 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 ($\times$); interquartile range with median (notched boxes) of simulated runlengths to reach $f_{\text{opt}} + \Delta f$; 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 $\Delta f = 10^{-5}$. 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 $f_{\text{opt}} + \Delta f$ with $\Delta f = 10^k$, where $k$ is the first value in the legend. Right subplots: ECDF of the best achieved $\Delta f$ 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 $f_{\text{opt}} + 10^{-8}$. 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.


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 $\Delta t$ values for functions $f_{1-24}$. The median number of conducted function evaluations is additionally given in italics, if ERT($10^{-7}$) = $\infty$. #succ is the number of trials that reached the final target $f_{\text{opt}} + 10^{-8}$.


