Experimental Comparison of Six Population-Based Algorithms for Continuous Black Box Optimization

Petr Pošík posik@labe.felk.cvut.cz
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Jiří Kubalík kubalik@labe.felk.cvut.cz
Faculty of Electrical Engineering, Czech Technical University in Prague, Czech Republic

Abstract

Six population-based methods for real-valued black box optimization are thoroughly compared in this article. One of them, Nelder-Mead simplex search, is rather old, but still a popular technique of direct search. The remaining five (POEMS, G3PCX, Cauchy EDA, BIPOP-CMA-ES, and CMA-ES) are more recent and came from the evolutionary computation community. The recently proposed "comparing continuous optimizers" (COCO) methodology was adopted as the basis for the comparison. The results show that BIPOP-CMA-ES reaches the highest success rates and is often also quite fast. The results of the remaining algorithms are mixed, but Cauchy EDA and POEMS are usually slow.

Keywords

Benchmarking, real-valued black box optimization, population-based algorithms, estimation of distribution algorithm, evolutionary strategy, covariance matrix adaptation.

1 Introduction

Population-based optimization algorithms (genetic and evolutionary algorithms, particle swarm optimization, etc.) are often employed to solve optimization problems in the continuous domain. They are derivative-free, thus applicable in the black box scenario, and they are said to be less prone to getting stuck in local optima. In this article we have chosen six population-based techniques for a comparison. Despite the fact that they each use different principles in taking advantage of the information hidden in the population, their instances used in this comparison express one common trait: all of them search in the neighborhood of a single point, and can thus be viewed as population-based *local* optimizers. The population serves in them either as a probe to explore the local neighborhood of the main point, and/or as a source of genetic material from which they construct the neighbors of the main point. The algorithms chosen for the comparison are introduced in the next paragraphs.

The iterative prototype optimization with evolved improvement steps, POEMS (Kubalík, 2009a), is a local search (LS) technique hybridized with an evolutionary algorithm (EA). Note that it is relatively common to see an LS inside an EA, but POEMS is the opposite—an EA inside an LS. The inner EA is used to evolve a sequence of

modifications, which—applied to the current prototype—create a better solution. Thanks to the EA operating on the current prototype, the search for the improvement can be rather global. See Section 2.1 for details.

The second algorithm is an estimation of distribution algorithm (EDA) (Larrañaga and Lozano, 2002). EDAs are a class of EAs that do not use the crossover and mutation operators to create the offspring population. Instead, they build a probabilistic model describing the distribution of the promising individuals and create offspring by sampling from the model. The particular EDA in this article uses the Cauchy distribution as its probabilistic model, hence its name—Cauchy EDA (Pošík, 2008), denoted as CEDA. For the details on the evolutionary model, learning, and sampling algorithms, see Section 2.2.

The third competitor is the generalized generation gap with parent-centric crossover, G3PCX (Deb, 2005). It is a steady-state evolutionary algorithm; with the parameterization used in this article, it is also very local-search intensive—in each generation it generates only two candidates in the neighborhood of the best so far solution. The algorithm and its crossover operator are described in Section 2.3.

As the reference algorithms for the above-mentioned optimizers, we chose three other techniques, which are shortly described in Section 2.4. The well-known Nelder-Mead simplex search method, NMSS (Nelder and Mead, 1965), is a deterministic algorithm. It maintains a population with a strictly defined size of $D + 1$ members (where D is the problem dimension) and adapts them using a set of if-then rules. Despite being old, this algorithm is still very popular.

The last two algorithms in the comparison are two restarted variants of the evolutionary strategy with covariance matrix adaptation, CMA-ES (Hansen and Ostermeier, 2001). The first one, BIPOP-CMA-ES, denoted as BI-CMA in this work, combines two modes of parameter settings for each restart. The algorithm was one of the most successful algorithms according to Hansen et al. (2010a), and is an example of the state of the art method. The other variant, which we simply denote as CMA-ES, with a slight abuse of notation, is a multi-start version of the basic algorithm (it uses the same algorithm parameters in all independent restarts).

The COCO (comparing continuous optimizers) methodology (Hansen, Auger, et al., 2009) was chosen as the tool for the comparison. The framework is able to show the differences among the algorithms at all stages of the search, not just after a certain number of evaluations, as is the usual practice. It was used as the basis of the black box optimization benchmarking (BBOB) workshops of the GECCO '09 and GECCO '10 conferences. The testbed consists of 24 carefully chosen noiseless benchmark functions (Hansen et al., 2009a) which represent various types of difficulties observed in real-world problems (ill-conditioning, multimodality, etc.). The dimension of the search space varied from two to 40 during the experiments. Note that the comparison takes into account the particular *algorithm instances*, not the algorithms in general. Our view of an algorithm includes not only the general algorithm itself, but also a particular set of parameter settings (parameter values, or parameter setting procedures).

The results for some of the above-mentioned algorithms were already presented as the workshop articles (Kubalík, 2009a; Pošík, 2009a, b; Hansen, 2009a, b). The results for the multi-start CMA-ES are presented for the first time, and constitute one of the original contributions of this article.

The goal of this article is to collect the results of all the above-mentioned algorithms, compare them conveniently in one place, provide a discussion of the pros and cons of the algorithms compared to each other, and formulate some hypotheses about the reasons

Algorithm 1 Prototype optimization with evolved improvement steps (POEMS)

```
1  i ← 0
2  Prototype^(i) ← generatePrototype()
3  while POEMS termination condition not satisfied do
4      i ← i + 1
5      BestSequence ← runEA(Prototype^(i-1))
6      Cand ← apply(BestSequence, Prototype^(i-1))
7      if Cand is better than or equal to Prototype^(i-1) then
8          Prototype^(i) ← Cand
9      else
10         Prototype^(i) ← Prototype^(i-1)
11 return Prototype^(i)
```

why certain algorithms work for a particular function class while others do not. We also discuss the influence of the algorithm restarting strategy. In the above-mentioned original articles, the discussion (if any) was based solely on the results of the respective algorithm and no comparison was made. We also discuss the results in more detail than the summary article of **?**.

The rest of the article is organized as follows. After describing the algorithms in Section 2, the COCO experimental framework in Section 3, and the experiment and algorithm parameter settings in Section 4, the article continues with the presentation of the benchmarking results in Section 5 and discusses them in Sections 6 and 7. The discussion is broken down by the individual function groups and by the algorithms, respectively. The article is summarized and concluded in Section 8.

2 Algorithms

2.1 Iterative Prototype Optimization with Evolved Improvement Steps

The POEMS approach is an iterative improvement method. It maintains the current best solution, the *prototype*, and replaces it if it finds a better one. The candidates that challenge the current champion are built by an evolutionary algorithm that searches for the best modification of the current prototype. The modifications evolved by the EA have the form of sequences of actions. An outline of the whole algorithm is shown as Algorithm 1.

In Algorithm 1, first, an initial prototype is generated (line 2). The solution to the problem is a vector in $\mathcal{R}^D$ and the initial prototype is generated by uniformly sampling the space $[l_i, u_i]^D$, where l_i and u_i are the lower and upper bounds defining the domain of each variable $1 \leq i \leq D$. Then the main loop, in which an improving neighbor of the current solution is sought by an EA, is repeated until the stopping condition is fulfilled (line 3). After each EA run (line 5), the best sequence of modifying actions found by the EA is applied to the current prototype (line 6) to build a neighbor candidate which competes with the current prototype. If the neighbor candidate is equally fit or better than the current prototype (line 7), it is accepted as the new prototype for the next iteration (line 8). Note that it is important to also accept neighbors of the same quality, since this provides a means for traversing plateaus in the search space.

2.1.1 Representation of Action Sequences

The EA evolves action sequences, which are fixed-length chromosomes composed of $MaxGenes$ genes, where each gene represents an instance of a certain action chosen from the set of elementary actions defined for the given problem. Each action is represented by an *action_type* attribute followed by the parameters of the action. Apart from actions that truly modify the prototype, there is also a special type of action called *nop* (no operation)—a void action with no effect on the prototype, regardless of the values of its parameters. Chromosomes can contain any number of instances of the *nop* operation.

The only active *action_type* used in this article is denoted as $changeVariable(i, v)$. This action changes the value of variable i by adding the value v. The parameter v can be a positive or a negative real number. For each variable i, its maximal change is limited to $|v| \leq \sigma_i$, with the constraint that $l_i \leq prototype[i] + v \leq u_i$ must always be satisfied. The values v of the actions generated to the starting population of action sequences are initialized to be close to the maximal available value according to the stochastic rule $v = \pm\sigma_i \times (0.95 + |r| \times 0.05)$, where $r \sim N(0, 1/3)$, $r \in (-1, 1)$. The values of v can later be modified during the EA run by a mutation operator, which will be discussed in Section 2.1.2.

The parameters σ_i are initialized to $\sigma_i = 0.25 \times (u_i - l_i)$ at the beginning of the POEMS run. During the course of the run, all the σ_i values are adapted in every iteration k as

$$\sigma_i^{(k)} = \sigma_i^{(k-1)} \times (1 - \alpha) + \delta_i \times \alpha,$$

where $\delta_i = |prototype[i]^{(k)} - prototype[i]^{(k-1)}|$ and $\alpha \in (0, 1)$ is a weighting factor ($\alpha = 0.2$ was used here). Thus, if the ith variable of the prototype does not change from iteration $k - 1$ to iteration k, the corresponding σ_i decreases with the maximal factor. In the opposite case, σ_i decreases to a lesser extent, or it may even increase. This can be interpreted so that if for the given value of σ_i an improving action sequence that includes a modification of the variable i has been found, there is perhaps no need to decrease the value of σ_i. In fact, the absence of an action modifying the variable i in the improving action sequence (or if no improving action sequence has been found in the current iteration) can indicate that the interval determined by the σ_i value is too wide. Thus, the search should focus on a closer neighborhood of the current prototype's value.

If the values of σ_i for all $1 \leq i \leq D$ fall below 10^{-11}, they are reinitialized to the original values $0.25 \times (u_i - l_i)$ while the current prototype remains unchanged. This can be considered a restart of the search strategy with the neighborhood size set to the maximal extent and the current prototype used as the initial prototype. The idea behind this action is to boost the exploration when all σ_i values have become negligible.

2.1.2 EA Inside POEMS

The `runEA()` function contains a complete EA. The EA works in an incremental mode: just two new action sequences are generated in each generation by means of crossover and mutation operators. In each generation, the parents are either crossed, with a probability P_{cross}, or they are mutated so that every action within the action sequence is mutated with a probability P_{mutate}. Typically, some variants of a simple action-swapping crossover and mutation operators that change action parameters are used. Here, the crossover operator works in two steps: first, a subset of active actions of both parents is chosen for the offspring, and then the offspring action sequence is completed with *nop* actions. This guarantees that the new action sequences will have at least one active action. The mutation operator is applied to the individual actions within the mutated action sequence. It changes the value v of an action so that it can be either increased or

Algorithm 2 Simple EDA used in this article

1 Initialize the parameters $\mu^0 = (\mu_1^0, \ldots, \mu_D^0)$, $\sigma^0 = (\sigma_1^0, \ldots, \sigma_D^0)$, and $R^0 \in \mathcal{R}^{D \times D}$, R is orthonormal, D is the dimensionality of the search space. Set the generation counter $t = 0$.
2 **while** termination condition is not met **do**
3 Sample N offspring from the search distribution (use R^t as a rotation matrix containing the base vectors, σ^t as relative scaling factors of individual components, and μ^t as the distribution center).
4 Evaluate the individuals.
5 Select the τN best solutions (truncation selection).
6 Estimate new μ^{t+1}, σ^{t+1}, and R^{t+1} using the selected individuals.
7 Enlarge the σ^{t+1} by a constant factor k (global step size).
8 Advance the generation counter: $t = t + 1$.

decreased as follows

$$v' = v \times (1 + r \times MaxChange),$$

where $MaxChange$ is a control parameter of the mutation operator chosen from the interval $(-1,1)$ once for the whole run and r is a normally distributed random value.

Once the prototype has reached a good quality, it is very likely that all action sequences in the randomly initialized population will worsen the prototype. Naturally, action sequences with a smaller number of active actions will worsen the prototype to a lesser extent. To prevent the EA from converging toward action sequences with a minimal number of active actions, a niching tournament selection and a steady state niching replacement strategy were proposed (Kubalík, 2009b). These niching strategies ensure that the population contains a diverse set of action sequences with respect to the number of active actions at any stage of the EA run.

2.1.3 Fitness Evaluation

The evolved action sequences are assessed based on how well/badly they modify the current prototype, which is passed as an input parameter to the EA. The action sequences that change the prototype by only a negligible factor, or do not change the prototype at all,[1] are fatally penalized to avoid the convergence to useless trivial modifications. This is implemented so that if an action sequence applied to the prototype produces a solution s such that $\forall i (i \in 1, \ldots, D) : |s[i] - prototype[i]| \leq 10^{-12}$, then this action sequence receives the worst possible fitness value.

2.2 Estimation of Distribution Algorithm with the Cauchy Model

As already stated, EDAs create and sample a probabilistic model to generate new candidate solutions. In real-valued spaces, such an algorithm can have a simple structure as shown in Algorithm 2.

If the Gaussian distribution is employed as the model of promising individuals and the parameters of the distribution, μ, σ, and R, are learned by the maximum likelihood (ML) estimation, the algorithm is very prone to premature convergence (i.e., the population converges on the slope of the fitness function) as recognized by many authors (e.g., Bosman and Thierens, 2000; Očenášek et al., 2004).

[1] For instance, action sequences that are composed solely of *nop* actions.

Algorithm 3 Generalized generation gap (G3, with PCX)

1 Initialize and evaluate the population $\mathcal{B}$ of size N.
2 **while** termination condition is not met **do**
3 Select the set $\mathcal{P}$ of parents from the population $\mathcal{B}$, that is, select the best member and $\mu - 1$ other individuals uniformly.
4 Generate the set $\mathcal{C}$ of λ offspring from the μ selected parents $\mathcal{P}$ using a chosen recombination scheme (PCX in this article).
5 Choose a subset $\mathcal{R}$ of r candidates for replacement uniformly from the population $\mathcal{B}$.
6 Replace the subset $\mathcal{R}$ of population $\mathcal{B}$ with the best r members from the combined set $\mathcal{C} \cup \mathcal{R}$.

Many techniques that fight premature convergence were developed, usually by means of artificially enlarging the ML estimate of the variance of the learned distribution. In this article, we use the technique suggested by Pošík (2008), namely using the Cauchy distribution instead of the Gaussian. We call the resulting algorithm a Cauchy EDA (and, as mentioned previously, we shall use the abbreviation CEDA to refer to the algorithm). This modification allows us to use a constant multiplier k which works for both the slope-like local neighborhood and the valley-like neighborhood at the same time.

The model parameters are actually computed using the ML estimates for the Gaussian distribution (even though they are subsequently used for the Cauchy distribution, i.e., they cannot be considered ML estimates any more, but they can serve as a heuristic). The distribution center μ is computed as the average of the selected data points, and the rotation matrix R and the standard deviations σ are obtained by the eigendecomposition of the covariance matrix of the selected data points.

During the sampling process, for each ith offspring, we sample the radius r_i from the 1D Cauchy distribution and a random vector uniformly distributed on the unit sphere. This vector is subsequently multiplied by the radius r_i. This operation results in an isotropically distributed set of offspring vectors z_i. Based on z_i, the final offspring x_i are generated by

$$x_i = \mu + k \cdot R \times \mathrm{diag}(\sigma) \times z_i, \tag{1}$$

where k is the constant multiplier.

CEDA is invariant with respect to translation and rotation. It uses fitness values only in the rank-based (truncation) selection, and thus it is also invariant against order-preserving transformations of the fitness function.

2.3 Generalized Generation Gap with Parent Centric Crossover

The generalized generation gap (G3) is an elitist steady-state model of an evolutionary algorithm (see Algorithm 3). It was introduced by Deb (2005). In that article, the G3 model was used with the parent centric crossover (PCX) operator introduced in Deb et al. (2002). The combination of G3 and PCX was tested on three 20-dimensional functions and was shown to be more efficient than CMA-ES (Hansen and Ostermeier, 2001). The G3 model with the PCX operator was also an order of magnitude faster than G3 with other crossover operators (simulated binary crossover, unimodal normally distributed crossover, or simplex crossover).

The PCX operator was first introduced in Deb et al. (2002) as an extension of the simulated binary crossover for any number of parents. In PCX, one parent, say $\mathbf{x}_p$, is

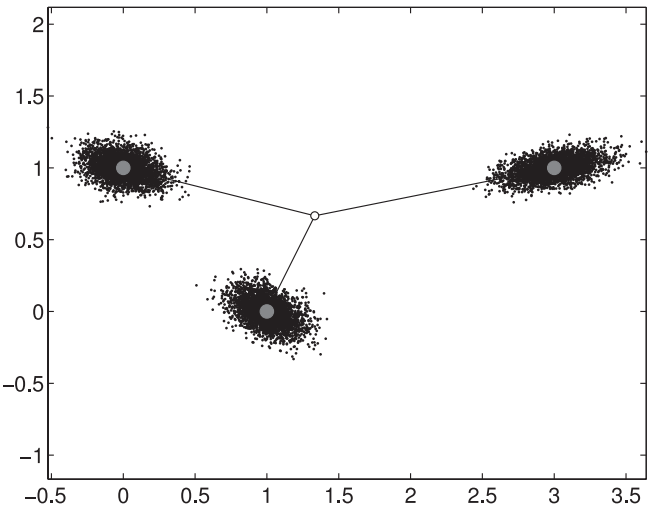

Figure 1: Demonstration of the offspring distribution induced by the PCX operator (10,000 points).

chosen from the population as the main one which will be used as the center for the distribution for sampling one offspring (for each offspring the main parent is chosen with equal probability). Let the other $\mu - 1$ parents be $\mathbf{x}_1, \ldots, \mathbf{x}_{\mu\text{-}1}$. First, the mean vector $\mathbf{g}$ of all μ parents is computed. Then, the direction vector $\mathbf{d}_p = \mathbf{x}_p - \mathbf{g}$ is computed. This vector has a significant role in PCX, since it represents the main direction, which is dealt with in a different way than with the other directions. For all other $\mu - 1$ parents, their perpendicular distances L_i, $i \in \langle 1, \mu - 1 \rangle$, to the $\mathbf{d}_p$ vector are computed. Based on them, the average distance $\bar{L}$ is found. The offspring is then created as

$$\mathbf{y} = \mathbf{x}_p + w_p \mathbf{d}_p + \sum_{i=1}^{D-1} w_i \bar{L} \mathbf{e}_i, \tag{2}$$

where $w_p \sim N(0, \sigma_p)$ is a normally distributed random value with the standard deviation σ_p and $w_i \sim N(0, \sigma_o)$ are normally distributed random values with the standard deviation σ_o (common to all w_i). The vectors $\mathrm{e}_1, \ldots, \mathrm{e}_{l\text{-}1}$ are the unit orthonormal base vectors spanning the linear subspace perpendicular to the vector $\mathbf{d}_p$.

In other words, the offspring is created from a D-dimensional normal distribution centered around the main parent $\mathbf{x}_p$ which is isotropic in all directions except the direction $\mathbf{d}_p$. The variance in all directions perpendicular to $\mathbf{d}_p$ is given by the standard deviation σ_o and by the average distance $\bar{L}$ of the other parents to the $\mathbf{d}_p$ vector, while variance in the direction of the $\mathbf{d}_p$ vector is given by the distance of the main parent $\mathbf{x}_p$ from the parent mean $\mathbf{g}$ and by the standard deviation σ_p. A demonstration of generating 10,000 points using PCX with three parents can be seen in Figure 1.

The G3PCX algorithm is invariant with respect to translation and rotation as can be seen in Section 5 on the graphs for unrotated–rotated versions of Ellipsoid (2 and 10), Rastrigin (3 and 15), or Rosenbrock (8 and 9) functions. The algorithm uses fitness values only in the rank-based selection, thus it is also invariant with respect to order-preserving transformations of the fitness function.

2.4 Reference Algorithms

Three other population-based algorithms were selected as competitors. They should emphasize where the main algorithms described above could do better.

The Nelder-Mead simplex search (NMSS) method was one of the first techniques that used several points (a population) as the description of the algorithm state. The algorithm is rather old (Nelder and Mead, 1965), but is still used very often—it survived the test of time. In D-dimensional space, it maintains the so-called simplex, a set of $D + 1$ points. Their relative positions and function values determine where to sample the next point(s) using the rules of

1. reflection (the worst vertex of the simplex is reflected around the center of the remaining D points),
2. extension (if reflection found a solution better than the best vertex, a new solution is sampled farther away in the same direction),
3. outside or inside contraction (if the reflection was not successful, a new point is sampled closer to the reflection center either outside or inside the simplex), and
4. shrinking (if none of the above operations found improvement of the simplex, shrink the simplex by moving all the simplex vertices closer to the best vertex).

Since the simplex changes its shape, can become elongated or stretched, the algorithm is sometimes called "amoeba." In this article, NMSS denotes the restarted version of the method as described in Hansen (2009b). The restarting procedure contains not only independent restarts, but also restarts initialized with a simplex in the neighborhood of the best solution found in the previous run, aimed at fine-tuning the solution.

The evolution strategy with covariance matrix adaptation (CMA-ES) was introduced by Hansen and Ostermeier (2001). For a decade it has been considered a state of the art method, with many successors (e.g. the BIPOP-CMA-ES in the next paragraph) and extensions. During the search, the method maintains a multivariate normal distribution (its center and its covariance matrix) which is used to sample new candidate solutions. They are in turn used to update the distribution parameters. The candidate solutions that are created in each generation can be thought of as mutated versions of the distribution center. A unique feature of the algorithm is that it estimates the distribution shape on the basis of successful mutation steps (rather than on the basis of locations of successful individuals in the search space as EDAs usually do). In this article, CMA-ES denotes a multistart version of the basic algorithm.

The bi-population CMA-ES algorithm (Hansen, 2009a) was chosen as another reference, since it was one of the best algorithms in the BBOB-2009 comparison regarding the final proportion of problems solved (**?**). In this article, it is denoted as BI-CMA. It uses the same base optimization method as the multi-start CMA-ES described above; however, it has a more sophisticated restarting scheme. The individual restarts differ in population size and in the initial size of the global step length. Two strategies of population size setting are interlaced. The first strategy multiplies its population size by a factor of two each time it is executed. The second strategy chooses the population size randomly, somewhere between the initial minimal population size and the half of the last population size used by the first strategy. Increasing the population size results in a more global and robust search.

The CMA-ES and BI-CMA techniques presented in this article share the same code (and thus also the same parameter settings). The only difference is that CMA-ES uses the same parameter settings for all restarts (and they are thus independent), while BI-CMA interlaces two strategies for setting the population size and the initial step size. The comparison of these two methods should show us where the more complicated restarting scheme of BI-CMA brings significant benefits over a plain multi-start strategy.

3 Experimental Framework Description

The experiments presented in this article were carried out using the COCO framework (Hansen, Auger, et al., 2009), which was also used as the basis for the BBOB workshop at the GECCO '09 and GECCO '10 conferences.

The numerical experiments are performed on a testbed consisting of 24 noiseless test functions (Finck et al., 2009a; Hansen et al., 2009a). These functions were constructed so that they reflect the real-world application difficulties and are categorized by function properties like multimodality, ill-conditioning, global structure, or separability. The role of these categories is to reveal the different aspects of the algorithms.

The COCO settings for BBOB 2009 were used in this article. All functions are scalable with the dimension D. The search domain is $[-5; 5]^D$, where $D = 2, 3, 5, 10, 20, 40$. Each of the functions has five instances with different rotations and offsets. The experiment was repeated three times for each instance, which means 15 trials for an algorithm on each function.

An *optimization problem* is defined as a particular (function, requested target value) pair. Each function is used to define several optimization problems differing in the requested target value $f_t = f_{opt} + \Delta f_t$, where f_{opt} is the optimal function value, and Δf_t is the precision (or tolerance) to reach. The success criterion of a trial (for each optimization problem) is to reach the requested target value f_t. Many precision levels $\Delta f_t \geq 10^{-8}$ are defined, for example, the ECDF figures of ERT (see the following paragraphs) use 50 target levels $\Delta f_t \in [10^{-8}, 10^2)$. If the optimizer solves a function to the ultimate precision value 10^{-8}, it actually solves many optimization problems along the way, and we shall say that it has found the optimum of the function. If the optimizer cannot reach the ultimate precision, it can gain some points for optimizing the function at least partially.

The main performance measure used in the COCO framework is the expected running time, ERT (Hansen, Auger, et al., 2009; Price, 1997). The ERT estimates the expected number of function evaluations needed to reach the particular target function value if the algorithm is restarted until a single success. The ERT thus depends on the given target function value, f_t, and is computed as "the number of function evaluations conducted in all trials, while the best function value was not smaller than f_t during the trial, divided by the number of trials that actually reached f_t" (Hansen, Auger, et al., 2009).

The results are conveniently presented using the Empirical Cumulative Distribution Function (ECDF). It shows the empirical cumulated probability of success on the problems considered depending on the allocated budget. The ECDF of the ERT is constructed as a bootstrap distribution of the ERT divided by the problem dimension D. In the bootstrapping process, 100 instances of ERT are generated by repeatedly drawing single trials with replacement until a successful trial is drawn for each optimization problem.

Since the ECDF graphs do not express reached function values, but rather the proportion of the solved problems, it is possible to meaningfully aggregate the ECDF graphs for several functions of the same class into a single graph. The downside of this aggregation is that we are not able to distinguish the individual functions. If a

graph shows aggregated ECDFs of five functions for a certain dimension D, reaching the 20% level of solved problems after n evaluations may mean many things. On the one hand, the algorithm could have found the minimum of one of the five functions, while the other functions may still remain completely unsolved. On the other hand, it may mean that only the problems related to the loose target levels were solved across all the aggregated functions. The latter case is the usual one. If the former explanation is correct, we will point it out explicitly.

4 Algorithm and Experiment Parameter Settings

The following sections describe the experimental setup and the parameter settings of POEMS, CEDA, and G3PCX. For the settings of the reference algorithms, we refer the reader to the original reports (Ros, 2009; Hansen, 2009a, b).

The experiments follow the BBOB 2009 methodology. The specification of the algorithm parameter values actually defines the particular algorithm instances benchmarked in this comparison. Whenever we talk, describe, or draw some conclusions about the algorithms on the basis of the results of this comparison, we actually mean the particular algorithm instances (with the parameter settings defined in the following sections).

All algorithms, including the reference ones, used the same parameter values (or the same parameter setting procedures) for all the benchmark functions in all dimensions.

4.1 POEMS

The majority of the POEMS control parameters used by Kubalík (2009a) were adopted. The configuration was only made dependent on the dimension of the current problem. The POEMS algorithm was configured as follows:

- $MaxGenes = D$, $NicheSize = 20$.
- $PopSize = MaxGenes \times NicheSize$.
- $P_{cross} = 0.75$, $P_{mutate} = 0.25$, $\alpha = 0.2$, $MaxChange = 0.5$.
- The tournament selection with $n = 2$ was used.
- $l_i = -5.0$ and $u_i = 5.0$ for all $i = 1 \ldots D$.
- The number of fitness evaluations calculated in each iteration was set to $10 \times PopSize$.
- The maximal number of fitness evaluations was $3 \times 10^5 D$.

4.2 CEDA

The algorithm has three parameters which determine its behavior:

1. The selection proportion ($\tau = 0.3$ is used throughout this article).
2. The population size N (depends on the search space dimensionality).
3. The variance enlarging factor k (also depends on the search space dimensionality).

Finding a good working combination of k and N is not an easy task. We resorted to the scenario in which (1) the parameters were tuned to one chosen function in all

dimensions, (2) the optimal values were then used to build expressions for k and N as functions of the problem dimension D, and (3) these equations were subsequently used for all tested problems.

The Rosenbrock function was chosen for the parameter tuning. The function is not easy to solve—the optimizers must follow the quadratic valley to the origin, then turn their search direction and continue to the optimum. Many optimizers actually converge to the origin, but then they are not able to change the search direction and/or enlarge the step size in the updated search direction. We believe that this feature of the Rosenbrock function makes it a good candidate for algorithm parameter tuning. The tuning itself was done by a global optimization algorithm called DIRECT (Jones et al., 1993). For each dimension,[2] the best found combination of the k and N values is presented in the following table:

D	2	3	5	10	20
k	0.7	1.2	1.5	1.6	1.7
N	30	50	80	250	700

Based on these measurements, the following models[3] for k and N were obtained:

$$N = 10^{1.05} \times D^{1.36} \tag{3}$$

$$k = \begin{cases} (0.3 + 0.25 \times D) \times \sqrt{D} & \text{if } D < 5, \\ (1.45 + 0.013 \times D) \times \sqrt{D} & \text{if } D \geq 5. \end{cases} \tag{4}$$

In each iteration, a check is performed if the model standard deviations did not fall below an acceptable limit. The limit was set to 10^{-10}, and if smaller values of σ_i are detected, they are artificially enlarged to the least acceptable value.

Several experiments with box constraints in the form $\langle -5, 5\rangle^D$ or $\langle -6, 6\rangle^D$ were carried out. Subjectively, the best (the most regular) results were produced when the algorithm was run completely unconstrained, that is, no constraints are used for the results presented in the next section.

The standard experimental procedure of BBOB 2009 was adopted: the algorithm was run on 24 test functions, with five instances each, and three runs on each instance. Each run was finished

- after finding a solution with the fitness difference $\Delta f \leq 10^{-8}$, or
- after performing more than $5 \times 10^4 D$ function evaluations.

Independent restart of the basic CEDA was carried out after the model converged too much, that is, whenever $\max_i |\sigma_i| < 10^{-8}$.

[2]DIRECT did not found any working combination of k and N in 40D space. (As a result, CEDA does not work well in 40D.) This may be due to the fact that we did not try sufficiently hard, or there is really no working combination. This remains to be investigated as a future work.

[3]We have also tried to adapt the k using the one-fifth rule, and N using the IPOP procedure—using twice as big a population for each restart, in an attempt to make an adaptive, almost parameter-less algorithm. However, CEDA with such a parameter setting performed much worse than CEDA using the presented models for k and N.

4.3 G3PCX

The G3PCX algorithm has a large number of parameters. For most of them, the values suggested by Deb (2005) are used, that is, for the G3 model:

- The number of parents $\mu = 3$.
- The number of offspring $\lambda = 2$.
- The number of replacement candidates $r = 2$.

For the PCX operator, both standard deviations are $\sigma_\mathrm{p} = \sigma_\mathrm{o} = 0.1$.

An important fact is that a highly local search intensive variant of the PCX operator was used: the main parent $\mathbf{x}_\mathrm{p}$ is not selected randomly, but rather the best parent (and thus the best population member) is always chosen as $\mathbf{x}_\mathrm{p}$.

Special care should be given to choosing the population size. Deb (2005) used the population size of 100 for all three 20D functions. It was also shown that the performance of the G3PCX drops quickly if insufficient population size is used, while it drops slowly if a larger than optimal population size is used. Experiments with 2D functions revealed that a relatively large population size (about 90) is needed to find the solution reliably, while to solve the 40D Sphere function, a population size of about 300 is needed. Thus, the following equation was used as the population sizing model:

$$N = 90 + 5 \times D. \tag{5}$$

This is a rather conservative choice and the model is by no means optimal (for 20D space it gives $N = 190$, which is almost twice as much as the optimal value suggested by Deb, 2005). It is thus very well possible that even better results can be provided by the G3PCX algorithm.

Hansen, Auger, et al. (2009) state that the global optimum of many benchmark functions lies in $\langle -4, 4\rangle^D$, and that all the benchmark functions should have their optimum in $\langle -5, 5\rangle^D$. This hypercube could serve as the bounding box for the optimization, but it is not clear if it would not affect the search negatively. With this in mind, the following scenarios were tested.

1. The G3PCX algorithm was run with box constraints $\langle -5, 5\rangle^D$, but this setting resulted in poor performance on the simple linear slope function—it is very hard for the algorithm to approach the solution lying on the boundary from only one side, while it might be simpler if it was allowed to overshoot the boundary and then focus on it.
2. The algorithm was then run completely unconstrained and the results on the linear slope function were much better. However, the results for many other functions were worse than in the above bounded case, since the algorithm spent a lot of the available budget on evaluating the unnecessary distant points.
3. The final configuration used in this article was to use the $\langle -6, 6\rangle^D$ hypercube as the search space. It is a viable compromise between the two previous cases. When a solution outside these bounds is generated, it is thrown away and a new candidate is created.

The standard experimental procedure of BBOB 2009 was adopted: the algorithm was run on 24 test functions, for five instances each, and three runs on each instance. Each run was finished

- after finding a solution with the fitness difference $\Delta f \leq 10^{-8}$, or
- after performing more than $5 \times 10^4 D$ function evaluations.

A restart of G3PCX was carried out when the population converged too much, that is, when all the population members converged to a hypercube with all the edges shorter than 10^{-10}. With a probability of 0.6, the population was initialized uniformly in the whole search space $[-5, 5]^D$; otherwise, the population was initialized with points in the neighborhood of the best solution x found in the previous launch of G3PCX, that is, in the dth dimension, the interval $[x_d - 0.1, x_d + 0.1]$ was used.

5 Results

The complete set of results from experiments according to Hansen, Auger, Finck, et al. (2010) on the benchmark functions (Finck et al., 2009b; Hansen et al., 2009b) are presented in the companion report (Pošík and Kubalík, 2012). In this article, we present only a subset of the graphical results; for details, we refer the reader to the companion report.

The ERT used in the figures depends on the given target function value, $f_t = f_{opt} + \Delta f$. When computing ERT, all 15 trials are taken into account. For each trial, we determine the runtime $RT_i, i = 1, \ldots, 15$, that is, the number of evaluations executed before the target function value f_t was reached. The ERT is then computed as the sum of the RT_i over all trials divided by the number of trials that actually reached the particular f_t (Hansen, Auger, Finck, et al., 2010; Price, 1997).

Figures 2 and 3 give the ECDF curves of ERT for all algorithms and for 5D and 20D, respectively. The 5D and 20D spaces were chosen as the representatives of low and higher dimensionality, yet they reveal a great amount of differences among algorithms. The large cross on the ECDF curves indicates the evaluation budget used with the respective algorithm. The part of the ECDF curve on the left-hand side of the cross is thus computed using the values directly measured during the experiment, while the right-hand part is the result of the bootstrapping procedure described in Section 3.

Figure 4 describes the scaling of the algorithms on each of the 24 benchmark functions. It shows the expected runtime divided by the dimension versus the problem dimension. Thus, we should observe a constant function (a horizontal line) for algorithms which scale linearly with the problem dimension, and a linearly increasing line with the slope indicated by the dashed grid lines for algorithms which scale quadratically with the dimension.

The detailed tables with the numerical results in the companion report contain the ERT for targets $10^{1, -1, -3, -5, -7}$, for all algorithms applied to all functions in all dimensions. The results of all algorithms are statistically tested if they improve the results of the baseline algorithm. In this article, the multistart CMA-ES was chosen as the baseline. Statistical significance is tested with the Mann-Whitney rank-sum test. The Bonferroni correction of 24 was applied to compensate for the effects of multiple testing. A more complete description of the presented results and the statistical testing procedure can be found in the companion report (Pošík and Kubalík, 2012).

6 Discussion by Function Group

In this section, the discussion of the results is broken down by the function groups. The discussion mostly applies to the presented results for 5D and 20D. Detailed results in the form of graphs and tables for all tested dimensions can be found in the companion

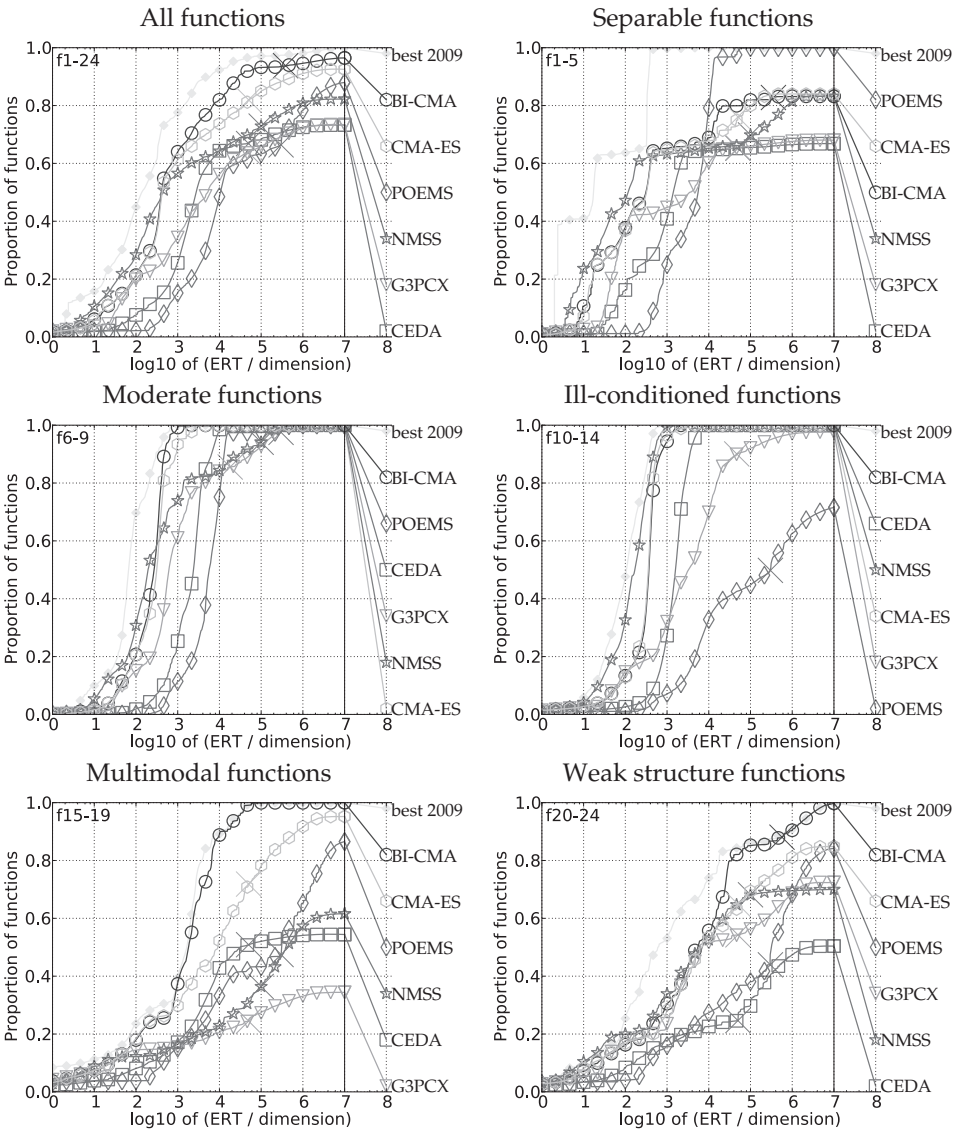

Figure 2: Empirical cumulative distribution function of the bootstrapped distribution of ERT over dimension for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 5D. The best 2009 line corresponds to the algorithms from BBOB-2009 with the best ERT for each of the targets considered.

report (Pošík and Kubalík, 2012). For a discussion on the individual algorithms, see Section 7.

6.1 All Functions Aggregated

The results for all functions are aggregated in the ECDF graphs of ERT for 5D and 20D functions in Figures 2 and 3, respectively, in the upper left part.

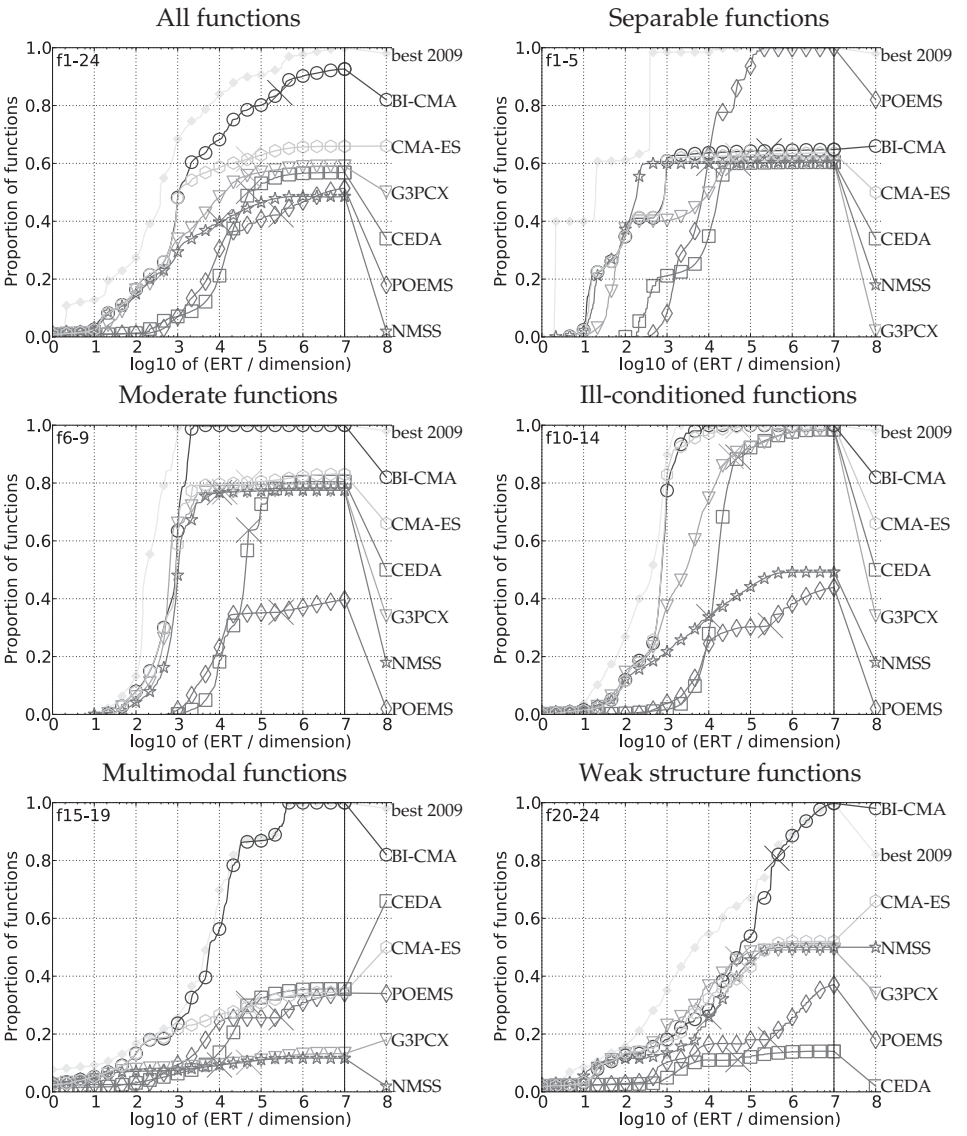

Figure 3: Empirical cumulative distribution function of the bootstrapped distribution of ERT over dimension for 50 targets in $10^{[-8..2]}$ for all functions and subgroups in 20D. The best 2009 line corresponds to the algorithms from BBOB-2009 with the best ERT for each of the targets considered.

In the 5D space, for the low function budgets (#FEs $< 400D$), NMSS is the most successful algorithm, solving about 50% of the problems. The success rate of BI-CMA and CMA-ES closely follows the NMSS method. CEDA and POEMS are slow and start showing some progress only after $100D$ evaluations. From $400D$ evaluations on, BI-CMA is clearly the best competitor, solving more than 95% of the problems eventually, followed by CMA-ES. After $400D$ evaluations, NMSS holds the third place; only for budgets larger than $10^6 D$ does POEMS outperform NMSS (but otherwise POEMS

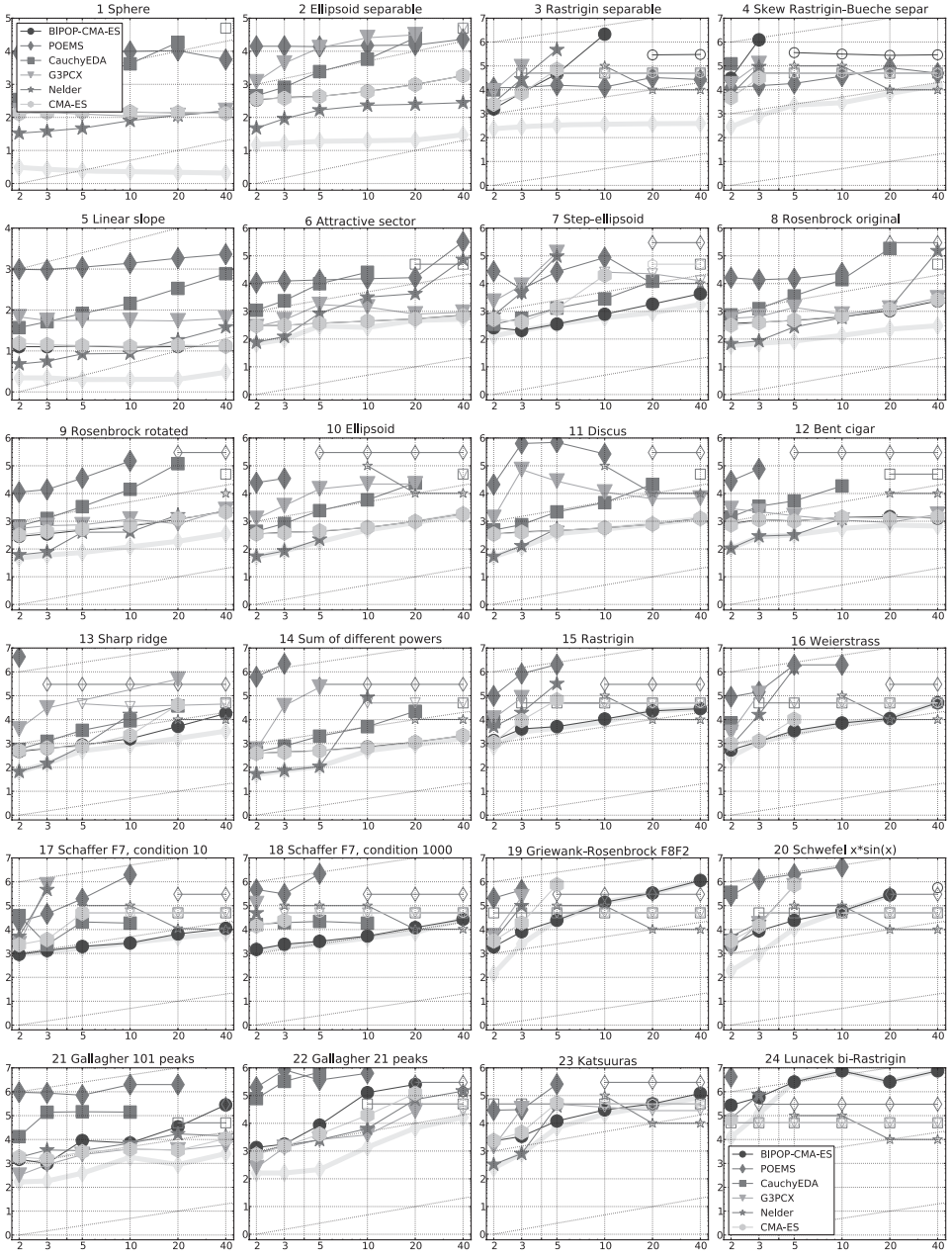

Figure 4: Expected running time (ERT) divided by dimension versus dimension in log-log presentation for the target function value 10^{-8} (filled symbols). Different symbols correspond to different algorithms given in the legend of f_1 and f_{24}. Hollow symbols give the average number of function evaluations divided by the dimension when there is no success. Horizontal lines give linear scaling, dashed lines give quadratic scaling. Legend: ○: BIPOP-CMA-ES, ◇: POEMS, □: CauchyEDA, ▽: G3PCX, ⋆: Nelder, ⬡: CMA-ES.

is rather slow—at least 100 times slower than the best BBOB 2009 competitors). Eventually, BI-CMA and CMA-ES place first, solving about 95% of the problems, POEMS is third, almost reaching the level of 90%, followed by NMSS, solving more than 80% of the problems. G3PCX and CEDA solved almost 75%.

In the 20D space, the situation is a bit different. Up to about $10D$ evaluations, virtually no problem was solved by any of the algorithms. For $10D < \#\text{FEs} < 500D$, BI-CMA, CMA-ES, G3PCX, and NMSS improved their success rate from zero to about 25%, while CEDA and POEMS solved only 5%. For budgets larger than $500D$, BI-CMA clearly wins, reaching the level of more than 90% of the solved problems. With a huge gap, CMA-ES eventually solved about 65% of the problems. G3PCX continued with a steady increase of success rate until it reached the level of 60%. NMSS holds its fourth position in success rate from 10^3D to 10^4D evaluations, but CEDA is better for budgets larger than 2×10^4D evaluations, and POEMS is better for the budgets larger than 2×10^6D evaluations. Eventually, CEDA reaches a success rate over 55%, POEMS about 50%, and NMSS solved less than 50% of the problems.

6.2 Separable Functions f_1–f_5

The results for the separable functions f_1–f_5 are aggregated in the ECDF graphs of ERT for the 5D and 20D functions in Figures 2 and 3, respectively, in the upper right part.

In the 5D space, for the budgets under $400D$ evaluations, NMSS is the fastest method and solves 60% of the problems. It is closely followed by BI-CMA and CMA-ES, which are, however, about two times slower. G3PCX reaches the level of 40% at $100D$ evaluations (similarly to BI-CMA), but then it needs a long time to get to 60% (to solve the ellipsoid function f_2). CEDA and POEMS are the slowest methods when solving the sphere (f_1), ellipsoidal (f_2), and linear slope (f_5) functions; but they reach the level of 60% sooner than G3PCX. The other two functions, f_3 and f_4, separable but multimodal, are much harder for the compared algorithms. At around 10^4D evaluations, POEMS shows a huge increase in success rate and solves both of these functions. BI-CMA, CMA-ES, and NMSS were eventually able to solve f_3, but not f_4. G3PCX and CEDA solved these functions only to the loose target levels. For the 5D separable functions, POEMS is the winner, solving all the problems (but it was about 12 times slower than the fastest algorithm of BBOB 2009). BI-CMA placed second, with more than 80% of the solved problems. The same level was also reached by NMSS, which needed a larger budget of evaluations. G3PCX and CEDA solved less than 70% of the problems.

In the 20D space, POEMS was still able to solve all the functions. BI-CMA, CMA-ES, and NMSS were not able to solve the Rastrigin function, f_3, and together with CEDA and G3PCX they solved about 60% of the problems. In the initial stages of the search, NMSS is the fastest algorithm, reaching a success rate of 60% with only about $200D$ evaluations. The initial speed of BI-CMA, CMA-ES, and G3PCX is acceptable; POEMS and CEDA are 50 to 100 times slower.

6.3 Unimodal Functions with Moderate Conditioning f_6–f_9

The results for the unimodal functions with moderate conditioning f_6–f_9 are aggregated in the ECDF graphs of ERT for the 5D and 20D functions in Figures 2 and 3, respectively, in the middle left part.

In the 5D space, the unimodal functions with moderate conditioning are relatively easy for the compared algorithms, since all of them eventually solved all functions. They differ only in their efficiency. For budgets lower than $300D$ evaluations, NMSS is the fastest, closely followed by BI-CMA, CMA-ES (less than two times slower), and G3PCX

(about three times slower). CEDA and POEMS have a larger gap; they are about 10 and 50 times slower, respectively. After $300D$ evaluations, BI-CMA (closely followed by CMA-ES) takes the lead and solves all the problems in about $10^3 D$ evaluations. NMSS and G3PCX made further progress only slowly (step-ellipsoid function f_7 is hard, but solvable for them), CEDA and POEMS eventually became better and reached the 100% level in about $10^4 D$ evaluations. G3PCX and NMSS needed more than $10^5 D$ evaluations to solve all the problems.

For the 20D problems, BI-CMA, CMA-ES, NMSS, and G3PCX are the most successful methods for the budgets under $1000D$ evaluations. The differences between them are small, the worst of them being less then two times slower then the best one. However, they are about eight times slower than the best competitors in BBOB 2009. For budgets larger than $1000D$, BI-CMA was the most successful algorithm, solving all the problems in less then $2000D$ evaluations. This time, CMA-ES did not follow its sibling and similarly to G3PCX and NMSS quickly reached the level of 80% of the solved problems; none of them solved the problems associated with the step-ellipsoid function, f_7. CEDA eventually reached and surpassed the level of 80% as well; but contrary to G3PCX and NMSS, CEDA did not solve the attractive sector function f_6. Eventually, BI-CMA solved all the problems, CMA-ES and CEDA slightly more than G3PCX and NMSS, but all about 80%. POEMS solved only 40% of the problems, mostly associated with f_6 and f_7, but it did not succeed in solving the Rosenbrock functions to the tighter target levels.

6.4 Unimodal Ill-Conditioned Functions f_{10}–f_{14}

The results for the unimodal ill-conditioned functions f_{10}–f_{14} are aggregated in the ECDF graphs of ERT for the 5D and 20D functions in Figures 2 and 3, respectively, in the middle right part.

In the 5D space, regarding the final proportion of the solved problems, the ill-conditioning does not affect the compared algorithms much, with the exception of POEMS; it was able to solve all the problems with moderate conditioning, but only about 70% of the ill-conditioned ones. It is also the slowest algorithm. NMSS, BI-CMA, and CMA-ES need only $1000D$ evaluations to solve all the problems; NMSS is two to four times faster than BI-CMA and CMA-ES. After $1000D$ evaluations, G3PCX and CEDA only reached the level of 30% of the solved problems; G3PCX had slower progress, but was better than CEDA in the initial phases (up to about $200D$ evaluations, it was actually comparable to BI-CMA). For budgets larger than $1000D$, the success rate of CEDA increases quickly and it solves all the problems after about $5000D$ evaluations (being only five times slower than NMSS or BI-CMA). G3PCX needs a large budget of about $10^6 D$ evaluations to reach a level close to 100%.

The functions where POEMS did not find the tightest target levels are f_{10} (ellipsoid), f_{12} (bent cigar), and f_{13} (sharp ridge). The f_{13} function was also hard for the G3PCX algorithm, which needs large budgets to solve it at least partially.

In the 20D space, POEMS is the worst algorithm among our competitors; it is slow and eventually solves less than 50% of the problems. BI-CMA, CMA-ES, G3PCX, and NMSS behave almost equally for the low budgets (#FEs $< 500D$). All four solved by this time about 20% of the problems, while CEDA and POEMS solved virtually no problem at this point. After $500D$ evaluations, the success rate of NMSS declines and improves only slowly. NMSS eventually solves about 50% of the problems. The improvements of G3PCX are slower than those of BI-CMA and CMA-ES, so that the gap between them increases. CEDA aligns with G3PCX after about 4×10^4 evaluations and together they almost reach the level of 100% of the solved problems.

For all functions, POEMS and NMSS fail to find the tightest target levels. On the ellipsoid function, f_{10}, POEMS is not able to make any progress at all. Functions f_{10} and f_{11} are also the hardest for the NMSS method.

6.5 Multimodal Functions f_{15}–f_{19}

The results for the multimodal functions f_{15}–f_{19} are aggregated in the ECDF graphs of ERT for the 5D and 20D functions in Figures 2 and 3, respectively, in the bottom left part.

For the 5D functions, G3PCX, NMSS, CMA-ES, and BI-CMA are the most successful algorithms for the low budgets (#FEs $< 50D$). Their initial procedures allow them to solve more than 10% of the problems by this limit. After $50D$ evaluations, both CMA-ES versions start to outperform the other algorithms. From $1000D$ evaluations, BI-CMA clearly dominates all the other algorithms and solves all the problems with less than $5 \times 10^4 D$ evaluations. CMA-ES makes a slower progress than BI-CMA, but eventually solves 95% of the problems. After about 5×10^4 evaluations, CEDA solved about 50% of the problems, POEMS slightly above 40, NMSS around 30, and G3PCX about 25%. After $10^5 D$ evaluations, the performance of CEDA and G3PCX increases only slowly, while POEMS and NMSS are able to make a significant progress. Eventually, POEMS solved about 85%, NMSS above 60%, CEDA about 55%, and G3PCX only 35% of the problems.

G3PCX was not able to solve any of the functions to the tight target levels. CEDA worked for the Schaffer functions, f_{17} and f_{18}, but found only the loose target levels for the other functions. NMSS solved the Rastrigin function, f_{15}, and the Weierstrass function, f_{16}, which was the opposite of CEDA. POEMS did not solve the Griewank-Rosenbrock function, f_{19}.

In the 20D space, the performance of all solvers dropped, with the exception of BI-CMA. It solved all the problems in less than $5 \times 10^5 D$ evaluations. NMSS and G3PCX do not show any substantial progress and eventually solve only about 15% of the problems. POEMS and CEDA ended up solving about 35% of the problems. CMA-ES followed the performance of BI-CMA in the initial stages, but at about $1000D$ evaluations, its success rate starts to decline and eventually it reached a similar success level to CEDA and POEMS, that is, 35%. The initial procedures of NMSS and G3PCX seem to be similarly effective as those of both CMA-ES variants.

6.6 Multimodal Functions with Weak Structure f_{20}–f_{24}

The results for the multimodal functions f_{20}–f_{24} are aggregated in the ECDF graphs of ERT for the 5D and 20D functions in Figures 2 and 3, respectively, in the bottom right part.

In the 5D space, NMSS, both variants of CMA-ES, and G3PCX exhibit similar performance until $10^4 D$ evaluations; by this limit, they were able to solve about 55% of the problems. POEMS and CEDA are much less successful: right from the beginning, they solved only 25% and 20% of the problems, respectively. After $10^4 D$ evaluations, BI-CMA is still rather effective in the search for the tight target levels and eventually solves all the problems, but needs about $10^7 D$ evaluations. Thanks to its more global behavior, POEMS was effective even in the later stages of the search and eventually reached the level of almost 85%, similar to CMA-ES, which was, however, faster than POEMS. G3PCX and NMSS follow, with about 73% and 70% of the solved problems, respectively.

CEDA can be described as the least effective and the slowest algorithm for this function group; it only solved 50% of the problems. It solved most of these 50% by solving the Gallagher functions, f_{21} and f_{22}; for the other functions, it found only the loose targets. The Gallagher functions were actually solved by all the algorithms. G3PCX did not solve the Katsuura function, f_{23}, and solved the Schwefel function, f_{20}; for NMSS, it was quite the opposite. The hardest function was f_{24}, Lunacek bi-Rastrigin, which was solved by BI-CMA only.

In the 20D space, the situation is similar, except that the performance of POEMS dropped significantly. The worst algorithm was CEDA again; it was slow and ineffective—it solved only about 15% of the problems. POEMS was only slightly better for most of the search, but eventually it reached almost 40% of the solved problems. CMA-ES, G3PCX, and NMSS solved virtually an equal proportion of the problems, 50%. Note that for budgets between $500D$ and $2 \times 10^4 D$, G3PCX is the most successful algorithm out of those six compared in this article. But the winner regarding the final proportion of the solved problems is BI-CMA again; it solved all the problems with about $10^7 D$ evaluations.

CEDA was not able to solve any problem to the tightest target levels. POEMS solved the Gallagher function, f_{21}, but failed on f_{22}. CMA-ES, G3PCX, and NMSS solved the f_{21} and f_{22} functions which gave them a 40% success rate. BI-CMA was the only algorithm to solve all the problems, including f_{20} and f_{24}, that were too hard for the other competitors.

7 Discussion by Algorithm

In this section, we look at the results from the point of view of the individual algorithms. A global view of the algorithms results is presented in Table 1. The table items describe the number of functions for which we are able to compute a finite ERT(10^{-8}), that is, the number of functions with at least one trial in which the ultimate precision 10^{-8} was successfully reached by the respective algorithm.

The table shows only one particular and very rough view of the results. It describes the effectiveness of the algorithms, not their efficiency. It can be seen that all algorithms are rather effective for $D \leq 5$. The differences start to be more pronounced for $D > 5$. The BI-CMA algorithm leaves the others consistently behind in this comparison, and it is in a class of its own. The other CMA-ES variant can be designated as the best of the rest.

The following sections present a detailed discussion of the results broken down by the main algorithms under the study: POEMS, CEDA, and G3PCX. Finally, a general discussion of all algorithms is included.

Table 1: The number of functions (out of 24) for which the algorithm found the ultimate precision of 10^{-8} for at least one run (out of 15) on the function.

D	2	3	5	10	20	40
POEMS	24	22	19	15	7	6
CEDA	21	16	16	15	10	1
G3PCX	23	22	14	11	11	8
NMSS	24	23	18	11	8	7
BI-CMA	24	24	23	23	22	20
CMA-ES	24	23	21	14	13	11

7.1 Discussion on POEMS

In this study, POEMS exhibits rather poor performance compared to its performance demonstrated on various discrete optimization problems (Kubalík, 2009a, 2011). This situation can be attributed to several aspects related to real-valued optimization. First, recall that when applied to combinatorial optimization problems, the so-called trivial action sequences are usually quite easily recognized; thus, they can be excluded from the evolutionary process in a straightforward way. In real-valued domains, this is no longer so easy. In fact, any action sequence that truly modifies the current prototype is acceptable, even if the magnitude of the modification is almost negligible. Moreover, there is a strong bias toward action sequences that change the current prototype as little as possible, since they are much more likely to generate better solutions than the action sequences that change the current prototype to a much greater extent. Consequently, POEMS starts to evolve these near-trivial action sequences at some point when the prototype is already fit. Naturally, this effect is even amplified for hard problems and/or for problems in high dimensions.

Another POEMS issue is the fact that it only adapts the size of the prototype neighborhood (σ_i values) from which the initial population of the action sequences is sampled at the beginning of each EA run, while the coordinate system defining the orientation of the prototype neighborhood stays invariant for the whole POEMS run. It is known that in the real-valued black box optimization, the ability to rotate the coordinate system plays an important role. If the coordinate system of the sampling operators is not aligned with the coordinate system of the objective function, undesirable effects such as premature convergence may occur even on the slope of the objective function.

Moreover, the σ_i values are updated based on a very limited information—the difference between the current and the evolved modified prototype. Obviously, such an adaptation scheme can be very imprecise.

The crossover operator based on swapping the actions between two parental action sequences is very ineffective for the real value optimization domain. In fact, it realizes the simple crossover (Wright, 1991) which is not appropriate for real-valued optimization as it induces only a very limited set of possible offspring solutions that can be scattered in the decision space, poorly referring to the original parental solutions.

The component-wise representation and crossover, on the other hand, allow the algorithm to solve the separable multimodal problems f_3 and f_4, which the other algorithms in this comparison did not solve.

The above listed causes of the POEMS inefficacy could have been eliminated to some extent. For example, to remedy the deficiency with the coordinate system adaptation, the adaptive encoding of Hansen (2008) can be used. Similarly, the action-swapping crossover could be replaced with some other recombination operator designed for the real-valued EAs, such as the arithmetic crossover of parent centric crossover (Deb et al., 2002). Some mechanisms to avoid generating near-trivial action sequences could be devised as well.

7.2 Discussion on CEDA

A first impression from looking at Figures 2 and 3 is that CEDA is a rather slow algorithm. This can be attributed to the generational replacement scheme and rather large population sizes it uses. The adaptation to the local neighborhood is thus slow, and before the right rotation is found, CEDA hardly makes any progress. But after the right coordinate system is found, the progress is usually fast, which can be seen on the ECDF graphs for unimodal functions.

Thanks to the fact that the modeling process does not use the function values of the individual points, the algorithm is able to solve the step-ellipsoid function, f_7, with a relatively small gap behind BI-CMA. On the other hand, it fails on the attractive sector function, f_6, in higher dimensions, probably because it samples new points around the mean of the better points; but for this function, the optimum always lies beyond the boundary of the selected points.

Although the performance of CEDA on the multimodal functions can hardly be described as satisfactory, CEDA was better (regarding the final success rate) than other algorithms (excluding BI-CMA) on the subgroup of multimodal functions with adequate structure in 20D. It seems that the large population size and the fact that it estimates a global model help the algorithm to find the global structure, at least in some cases (i.e., f_{17} and f_{18}).

Careful inspection of the results and logs revealed another deficiency of the algorithm: despite the fact that it was designed as a multi-start algorithm, the restart was virtually never executed! The majority of the results for all function instances thus come from one long run of the algorithm. The reason may be an incorrect setting of the variance enlargement factor k, which prevents the model from converging and firing up the restart condition.

As the scaling graphs in Figure 4 suggest, for functions where the algorithm works, the scaling is usually at least quadratic.

7.3 Discussion on G3PCX

In this comparison, G3PCX is the algorithm closest to an ordinary real-valued EA. It is a greedy algorithm (i.e., always searching in the neighborhood of the current best solution). Thanks to its steady-state model, it makes small incremental updates and shows some progress right from the beginning. For many problems, it belongs to the fastest algorithms in the initial stages.

Interestingly, for the unimodal problems, its final success rate is almost the same as that of CEDA. (Of course, this does not mean that they both solved the same problems or that both failed on the same problems.) The success rates of CEDA and G3PCX differ, however, in the case of the multimodal problems: CEDA is better for functions with adequate structure, while G3PCX is better on the weak-structure functions. This can be explained by the above-mentioned deficiency of CEDA: while G3PCX effectively uses restarts, CEDA relies on a single long run.

On functions which can be solved by G3PCX, its scaling is often subquadratic.

7.4 General Discussion

An interesting aspect of all the algorithms is the restart strategy they use. G3PCX and CMA-ES use the ordinary multi-start strategy. BI-CMA and NMSS use more sophisticated strategies where the restarts are not independent, and the algorithm parameters differ between the individual restarts. POEMS is a special case—it is designed as an iterated local search method where the individual runs of the inner EA try to search for an improvement in the neighborhood of the solution found in the previous runs. Complete independent restarts are possible with POEMS, but the algorithm seldom carries them out. CEDA was almost never restarted.

By comparing the results of BI-CMA and CMA-ES, we can evaluate the potential benefits that a more sophisticated restart strategy can have compared to the plain multi-start strategy. For the majority of the unimodal functions, there is no difference between BI-CMA and CMA-ES, since usually the first launch of CMA-ES finds the optimum.

The exceptions are the f_7 (step-ellipsoid) and f_{13} (sharp ridge), where some restarts are needed (especially in higher dimensions) and the more complex strategy used by BI-CMA is preferable to a simple multi-start strategy. We hypothesize that for both functions, the restarts with increasing population size are profitable. The difference between BI-CMA and CMA-ES becomes clear on the multimodal functions. The restarts of BI-CMA (with much smaller initial step size than the default) allow it to converge faster and perform more restarts. The increasing population size in BI-CMA is suitable for the cases where the function has some global structure but otherwise is highly rugged; with a larger population size, BI-CMA can spot and use that structure during model building.

There is also another interesting research question: Is it better to have a good local optimizer quickly converging to a local optimum and restart it often, or to have a global optimizer able to preserve the diversity in the population for a long time and hope that it will find an optimum in a single long run? The results seem to support the first option. Algorithms with a small population size (BI-CMA, CMA-ES, NMSS) are allowed to quickly converge in the initial phases and do not waste function evaluations on evaluating the large population sizes (CEDA, POEMS). However, they must be able to use the information gained by the small population size effectively (both CMA-ES variants), not to lose their performance for the harder problems or the tighter target levels (NMSS). The most promising variant seems to be to use the best of both worlds via a sophisticated restart scheme with an increasing population size (BI-CMA): start with small population sizes with many restarts and subsequently allow for larger population sizes and longer runs. This, however, requires an algorithm able to effectively use the information hidden in a population, no matter whether it is small or large.

There is another algorithmic feature that can be used to classify the algorithms: POEMS, G3PCX, and NMSS directly use the population members to create the offspring individuals by combining some features of the parents, while BI-CMA, CMA-ES, and CEDA use a middle step—they extract the information hidden in all the parents to build/update a probabilistic model and create the offspring by sampling from the model. This feature seems to be very useful for some functions in the current comparison (and may also be beneficial for highly rugged landscapes or functions with noise). For example, the f_7, step-ellipsoid, was efficiently solved only by BI-CMA and CEDA, and for lower dimensions also by CMA-ES. In higher dimensions, a larger population is needed. BI-CMA gets a sufficiently large population after several restarts, CEDA has a large population right from the beginning, but CMA-ES cannot acquire a sufficiently large population and is not able to solve this function in 20D (but is still better than the other algorithms).

8 Summary

Six population-based optimization methods were compared in this article. Three of them, POEMS, CEDA, and G3PCX, were reintroduced and discussed in detail, while the remaining three, NMSS, BI-CMA, and CMA-ES, played the role of reference optimizers.

Overall, the best performing algorithm in this comparison was clearly BI-CMA. It wins thanks to its robustness, that is, thanks to its very good results on a wide range of functions. For particular functions, function groups, target values, and/or dimensions, there may be faster algorithms, but BI-CMA is usually slower only by a factor from two to five. The other methods produced results of mixed quality.

Instead of conclusions, several hypotheses can be formulated on the basis of the observed results; some of them may be quite trivial. First, restarting an algorithm from a random point in the search space makes from an incomplete algorithm an

asymptotically complete one, but generally it does not increase the success rate of the algorithm significantly. (But there are problems where independent restarts work nicely, e.g., f_{21} and f_{22}.) Especially in the case of slowly converging methods, the resulting frequency of the restarts is low. Restarting with increasing population size, as used by BI-CMA, can be a better option.

Second, the algorithm's ability to adapt its state to the local neighborhood (in particular, to rotate its own coordinate system) is crucial for good algorithm performance for most of the problems. All the compared algorithms, with the exception of POEMS, are able to adapt themselves in this sense (using different means). This ability is, however, misleading in the case of separable, highly multimodal functions: among the tested algorithms, only POEMS was able to solve the f_3 and f_4 functions. Nonetheless, such functions are very rare in the real world; it is definitely profitable to be able to learn the rotation.

Third, building a global model (as done by BI-CMA, CMA-ES, and CEDA) can be helpful in the case of multimodal functions with an adequate global structure. We hypothesize that such functions are actually similar to noisy functions and that such algorithms are also more resistant to noise.

Fourth, the steady-state model (or the incremental adaptation) is useful for the algorithm's ability to improve the results right from the beginning of the optimization. The generational replacement scheme in combination with large population sizes actually do not allow the algorithm to make quick progress in the initial stages. Large population sizes on their own also do not guarantee good performance on multimodal functions.

Acknowledgments

The authors would like to thank Nikolaus Hansen, Raymond Ros, and Anne Auger for the tremendous amount of work during the creation and development of the framework for systematic algorithm comparisons used in this article. The first author was supported by the Grant Agency of the Czech Republic with grant No. 102/08/P094, entitled "Machine learning methods for solution construction in evolutionary algorithms." The second author was supported by the Ministry of Education, Youth and Sport of the Czech Republic with grant No. MSM6840770012, entitled "Transdisciplinary Research in Biomedical Engineering II."

References

Bosman, P. A. N., and Thierens, D. (2000). Expanding from discrete to continuous estimation of distribution algorithms: The IDEA. In *PPSN VI: Proceedings of the 6th International Conference on Parallel Problem Solving from Nature*, pp. 767–776.

Deb, K. (2005). A population-based algorithm-generator for real-parameter optimization. *Soft Computing*, 9(4):236–253.

Deb, K., Anand, A., and Joshi, D. (2002). A computationally efficient evolutionary algorithm for real-parameter optimization. *Evolutionary Computation*, 10(4):371–395.

Finck, S., Hansen, N., Ros, R., and Auger, A. (2009a). Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE.

Finck, S., Hansen, N., Ros, R., and Auger, A. (2009b). Real-parameter black-box optimization benchmarking 2009: Presentation of the noiseless functions. Technical Report 2009/20, Research Center PPE. Updated February 2010.

Hansen, N. (2008). Adaptive encoding: How to render search coordinate system invariant. In G. Rudolph (Ed.), *Parallel Problem Solving from Nature—PPSN X. Lecture notes in computer science*, Vol. 5199 (pp. 205–214). Berlin: Springer-Verlag.

Hansen, N. (2009a). Benchmarking a BI-population CMA-ES on the BBOB-2009 function testbed. In F. Rothlauf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09*, pp. 2389–2396.

Hansen, N. (2009b). Benchmarking the Nelder-Mead downhill simplex algorithm with many local restarts. In F. Rothlauf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09*, pp. 2403–2408.

Hansen, N., Auger, A., Finck, S., and Ros, R. (2009). Real-parameter black-box optimization benchmarking 2009: Experimental setup. Technical Report RR-6828, INRIA.

Hansen, N., Auger, A., Finck, S., and Ros, R. (2010). Real-parameter black-box optimization benchmarking 2010: Experimental setup. Technical Report RR-7215, INRIA.

Hansen, N., Auger, A., Ros, R., Finck, S., and Pošík, P. (2010a). Comparing results of 31 algorithms from the black-box optimization benchmarking BBOB-2009. In M. Pelikan and J. Branke (Eds.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '10*, pp. 1689–1696.

Hansen, N., Finck, S., Ros, R., and Auger, A. (2009a). Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA.

Hansen, N., Finck, S., Ros, R., and Auger, A. (2009b). Real-parameter black-box optimization benchmarking 2009: Noiseless functions definitions. Technical Report RR-6829, INRIA. Updated February 2010.

Hansen, N., and Ostermeier, A. (2001). Completely derandomized self-adaptation in evolution strategies. *Evolutionary Computation*, 9(2):159–195.

Jones, D. R., Perttunen, C. D., and Stuckman, B. E. (1993). Lipschitzian optimization without the Lipschitz constant. *Journal of Optimization Theory and Applications*, 79(1):157–181.

Kubalik, J. (2009a). Black-box optimization benchmarking of prototype optimization with evolved improvement steps for noiseless function testbed. In F. Rothlauf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09*, pp. 2303–2308.

Kubalík, J. (2009b). Solving the sorting network problem using iterative optimization with evolved hypermutations. In *Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO '09*, pp. 301–308.

Kubalík, J. (2011). Evolutionary-based iterative local search algorithm for the shortest common supersequence problem. In *Proceedings of the 13th Annual Conference on Genetic and Evolutionary Computation, GECCO '11*, pp. 315–322.

Larrañaga, P., and Lozano, J. A. (Eds.). (2002). *Estimation of distribution algorithms*. Norwell, MA: Kluwer Academic Publishers.

Nelder, J., and Mead, R. (1965). A simplex method for function minimization. *The Computer Journal*, 7(4):308–313.

Očenášek, J., Kern, S., Hansen, N., and Koumoutsakos, P. (2004). A mixed Bayesian optimization algorithm with variance adaptation. In X. Yao (Ed.), *Parallel Problem Solving from Nature—PPSN VIII* (pp. 352–361). Berlin: Springer-Verlag.

Pošík, P. (2008). Preventing premature convergence in a simple EDA via global step size setting. In G. Rudolph (Ed.), *Parallel Problem Solving from Nature—PPSN X. Lecture notes in computer science*, Vol. 5199 (pp. 549–558). Berlin: Springer-Verlag.

Pošík, P. (2009a). BBOB-benchmarking a simple estimation of distribution algorithm with Cauchy distribution. In F. Rothlauf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09*, pp. 2309–2314.

Pošík, P. (2009b). BBOB-benchmarking the generalized generation gap model with parent centric crossover. In F. Rothlauf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09*, pp. 2321–2328.

Pošík, P., and Kubalík, J. (2012). Experimental comparison of five population-based algorithms for continuous black-box optimization: A companion. Retrieved from http://labe.felk.cvut.cz/~posik/papers/PopAlgs/PopAlgsComp.pdf

Price, K. (1997). Differential evolution vs. the functions of the second ICEO. In *Proceedings of the IEEE International Congress on Evolutionary Computation*, pp. 153–157.

Ros, R. (2009). Benchmarking the NEWUOA on the BBOB-2009 function testbed. In F. Rothlauf (Ed.), *Proceedings of the Genetic and Evolutionary Computation Conference, GECCO '09*, pp. 2421–2428.

Wright, A. H. (1991). Genetic algorithms for real parameter optimization. In G. J. Rawlins (Ed.), *Foundations of genetic algorithms* (pp. 205–218). San Mateo, CA: Morgan Kaufmann.