A Continuous Variable Neighbourhood Search Based on *Specialised EAs*: Application to the Noisy BBO-Benchmark 2009 Testbed

Carlos García-Martínez
Dept. of Computing and Numerical Analysis
14071 University of Córdoba
Córdoba, Spain
cgarcia@uco.es

Manuel Lozano
Dept. of Computer Science and Artificial Intelligence
CITIC-UGR (Research Center on Information and Communications Technology)
18071 University of Granada
Granada, Spain
lozano@decsai.ugr.es

ABSTRACT

Variable Neighbourhood Search is a metaheuristic combining three components: generation, improvement, and shaking components. In this paper, we design a continuous Variable Neighbourhood Search algorithm based on three specialised Evolutionary Algorithms, which play the role of each aforementioned component: 1) an EA specialised in generating a good starting point as generation component, 2) an EA specialised in exploiting local information as improvement component, 3) and another EA specialised in providing local diversity as shaking component. Experiments are carried out on the noisy Black-Box Optimisation Benchmark 2009 testbed.

Categories and Subject Descriptors

G.1.6 [**Numerical Analysis**]: OptimizationGlobal Optimization, Unconstrained Optimization; F.2.1 [**Analysis of Algorithms and Problem Complexity**]: Numerical Algorithms and Problems

General Terms

Algorithms

Keywords

Benchmarking, Black-box optimization, Evolutionary computation, Hybrid metaheuristics, Variable neighbourhood search, Specialised evolutionary algorithms

1. INTRODUCTION

Metaheuristics (MHs) [11] are a family of search and optimisation algorithms based on extending basic heuristic methods by including them into an iterative framework augmenting their exploration capabilities. MHs coordinate *subordinate components* (such as probability distributions, tabu lists or genetic operators among others) with the aim of performing an effective and efficient process in searching for the global optimum of a problem.

Over the last years, a large number of search algorithms were reported that do not purely follow the concepts of one single classical MH, but they attempt to obtain the best from a set of MHs that perform together and complement each other to produce a profitable synergy from their combination. These approaches are commonly referred to as *hybrid MHs* [23].

Evolutionary Algorithms (EAs) [4] are stochastic search methods that mimic the metaphor of natural biological evolution. EAs rely on the concept of a population of individuals, which undergo probabilistic operators to evolve toward increasingly better fitness values of the individuals.

A novel method to build hybrid MHs, recently introduced in [20], concerns the incorporation of *specialised EAs* into classical MHs, replacing determinate components, but preserving the essence of the original MH as much as possible. The idea is to build customised EAs playing the same role as particular MH components, but more effectively, i.e., *evolutionary MH components*. In this way, a classical MH is transformed into an integrative hybrid MH (because one of its components is another MH). In the literature, we find some proposals that follow this idea: In [1] and [7], two EAs are applied to perform local search processes within a multi-start MH; and in [19, 20], an evolutionary ILS-perturbation technique is presented (ILS is for Iterated Local Search).

Variable Neighbourhood Search (VNS) [22] is a MH that exploits systematically the idea of neighbourhood change (from a given set of neighbourhood structures N_k, $k = 1, \ldots, k_{max}$), both in the descent to local minima and in the escape from the valleys which contain them. It mainly consists of the following three components, which work on a single candidate solution, the *current solution* ($\boldsymbol{s^c}$) (see Fig. 1):

1. *Generation component:* Firstly, a method is executed to generate $\boldsymbol{s^c}$ within the search space.
2. *Improvement component:* Secondly, $\boldsymbol{s^c}$ is refined, usually by a local search method.

GECCO'09, July 8–12, 2009, Montréal Québec, Canada.

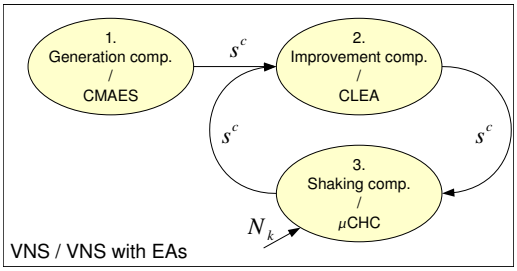

Figure 1: General VNS model / VNS based on specialised EAs

3. *Shaking component:* Then, shaking is performed to escape from the valley where $\boldsymbol{s^c}$ lies. It selects a random solution in the kth neighbourhood of $\boldsymbol{s^c}$, which becomes a new starting solution for improvement component. At the beginning of the run, and every time last improvement process improved the best found solution, k is set to one; otherwise, k is set to $k+1$.

A continuous VNS model based on three specialised EAs, which play the role of each VNS component, was benchmarked on the noiseless Black-Box Optimization Benchmark 2009 testbed [8]. CPU timing experiment was also included. In this study, we benchmark the same VNS model on the *noisy* Black-Box Optimization Benchmark 2009 testbed.

In Sect. 2, we briefly describe the proposed VNS model. A slight more extensive description can be found in [8]. In Sect. 3, we present the experimental results. Conclusions are presented in Sect. 4.

2. VNS BASED ON *SPECIALISED EAS*

In this study, we design a continuous VNS model based on three specialised EAs[1], which play the role of each VNS component (see Fig. 1):

1. *Covariance Matrix Adaptation Evolution Strategy* (CMA-ES) [14, 15] generates initial $\boldsymbol{s^c}$ (Sect. 2.1).

2. *Continuous Local EA* improves $\boldsymbol{s^c}$ (Sect. 2.2).

3. *Micro Cross-generational elitist sel., Heterogeneous recombination, and Cataclysmic mutation* (μCHC) performs shaking to scape from the valley where $\boldsymbol{s^c}$ lies, according to kth neighbourhood (Sect. 2.3).

   Adaptation of parameter k is performed the same way general VNS model does. However, to avoid small and frequent improvements when tackling continuous problems, k is set to one only if the improvement is superior to a threshold ($1e-8$). Otherwise, k is set to $k+1$.

Neighbourhoods structures are defined by using metric ρ:

$$N_k(\boldsymbol{s^c}) = \{\boldsymbol{s} \mid r_{k-1} \leq \rho(\boldsymbol{s^c}, \boldsymbol{s}) \leq r_k\} \quad (1)$$

where r_k is the radius of $N_k(\boldsymbol{s^c})$ monotonically nondecreasing with k. Hereafter, we will always consider the following distance metric because of computational reasons:

[1]Sourcecode can be found together with results within the corresponding data files archive.

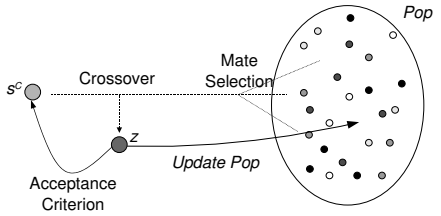

Figure 2: Continuous Local EA

$$\rho(\boldsymbol{x}, \boldsymbol{y}) = \sum_{i=1}^{n} |x_i - y_i| \quad (2)$$

Stop condition is reached when k surpasses k_{max} (in our case, k_{max} is set to 20). At this point, as it is suggested in [12], the algorithm performs a *restart*. In this way, exploiting larger number of function evaluations may increase the chance to achieve better function values or solve the function up to the final target. In our case, a restart means that VNS is run once again from the CMA-ES execution, generating a new $\boldsymbol{s^c}$.

2.1 CMA-ES as Generator Component

In this work, we apply CMA-ES [14, 15] to obtain a good point in the design space from which the algorithm carries out its search process. We have applied the octave code, available at `http://www.lri.fr/~hansen/cmaes_inmatlab.html` with the suggested parameter values from a random solution.

2.2 Continuous Local EA as Improvement Component

Continuous Local EA is based on the principles of Binary Local Genetic Algorithm [10]. It is a steady-state *real-coded genetic algorithm* [16] that inserts one single new member into the population (*Pop*) in each iteration. It uses a replacement method in order to force a member of the current population to perish and to make room for the new offspring. It is important to know that the selected replacement method favours the acquisition of information about the search space. Then, the algorithm exploits local information around $\boldsymbol{s^c}$, provided by the individuals in *Pop* close to it, to orientate the improvement process.

Let's suppose that Continuous Local EA is to improve the given $\boldsymbol{s^c}$. Then, following steps are carried out (see Fig. 2):

- *Mate selection*: A solution from *Pop* is selected as $\boldsymbol{s^{mate}}$ by means of *nearest improving solution selection method* (Sect. 2.2.1).

- *Crossover*: $\boldsymbol{s^{mate}}$ and $\boldsymbol{s^c}$ are crossed over by means of *parent-centric BLX-*α, creating offspring $\boldsymbol{z}$ (Sec. 2.2.2).

- *Acceptance and replacement*: If $\boldsymbol{z}$ is accepted (Sect. 2.2.3), it replaces $\boldsymbol{s^c}$ and is inserted into the population forcing the individual with lowest information contribution to perish (Sect. 2.2.4).

All these steps are repeated until a maximun number of iterations without improving the best visited $\boldsymbol{s^c}$ is reached (we stop the run after 100 iterations without improving best $\boldsymbol{s^c}$). At last, the best found $\boldsymbol{s^c}$ is returned.

An important aspect when using Continuous Local EA in the proposed VNS model is that *Pop* undergoes initialisation only once, at the beginning of the run, and not at every invocation to improve $\boldsymbol{s^c}$. In this way, the algorithm may gather up valuable information about the search space and its optima, and employ accumulated search experience from previous refinements to enhance future ones.

2.2.1 Nearest Improving Solution Selection

In Continuous Local EA, first parent is always $\boldsymbol{s^c}$, then, *nearest improving solution selection* mechanism selects as second parent the individual in *Pop* most similar to $\boldsymbol{s^c}$ (at phenotypic level) fulfilling two conditions: 1) it has a better fitness value than $\boldsymbol{s^c}$ and 2) similarity between both solutions is above a given threshold. Mating threshold has been initially set to $1e-2$. Then, after every VNS restart, it is set to half its value to perform a more precise search. The interested reader is referred to [8] for the idea behind this selection mechanism.

An important remark is that this selection mechanism may return no mate for $\boldsymbol{s^c}$. It occurs when the mechanism has dismissed all solutions in *Pop* because they are either worse than $\boldsymbol{s^c}$, either their distances to $\boldsymbol{s^c}$ are under mating threshold. As mentioned before, crossover is not performed in this case. Offspring is generated by perturbing $\boldsymbol{s^c}$ instead (Sect. 2.2.2).

2.2.2 PBX-α Crossover Operator

PBX-α [21] is an instance of *parent-centric crossover operators*, which have arisen as a meaningful and efficient way of solving real-parameter optimisation problems [3, 9].

Given $\boldsymbol{s^c}=(s_1^c \cdots s_n^c)$ and $\boldsymbol{s^{mate}}=(s_1^{mate} \cdots s_n^{mate})$, PBX-$\alpha$ generates offspring $\boldsymbol{z}=(z_1 \cdots z_n)$, where z_i is a random (uniform) number from interval $[s_i^c - I \cdot \alpha, s_i^c + I \cdot \alpha]$, with $I = |s_i^c - s_i^{mate}|$. Parameter α is initially set to 0.5. Then, after every VNS restart, it is set to $0.5/log_e(number\ of\ restarts+1)$ for a more precise search.

As mentioned before, crossover operation occurs when mate selection returns a solution from *Pop*. Otherwise, offspring is generated by adding a normal random vector to $\boldsymbol{s^c}$, whose standard deviation is the product of the current values of aforementioned parameters α and mating threshold.

2.2.3 Acceptance Criterion

Once offspring $\boldsymbol{z}$ has been generated either by PBX-α or perturbation, Continuous Local EA decides which solution, between $\boldsymbol{z}$ and $\boldsymbol{s^c}$, becomes the new $\boldsymbol{s^c}$. It follows a similar idea to the ones behind *Simulated Annealing* [18] and *Noising Methods* [2] to overcome rugged landscapes and local optima. In particular, it always accepts $\boldsymbol{z}$ if it is better than a specific computed bound. At the beginning, that bound is set equal to the fitness value of $\boldsymbol{s^c}$, only allowing improving offspring. Then, every time a new best $\boldsymbol{s^c}$ is found, the bound is computed as the average of the fitness of the new best $\boldsymbol{s^c}$ and the old bound. This mechanism provides some selection pressure on the process that also lets the algorithm to accepts uphill moves, and overcome small local optima.

2.2.4 Lowest Information Contribution Replacement

In this work, we propose a method for updating *Pop* of Continuous Local EA, when solving mono-objective problems. Its aim is to maintain a set of good solutions properly spread over the search space. A more extensive description of this idea can be found in [8]. The mechanism firstly inserts the new offspring in *Pop*, then, it removes the most crowded solution, keeping population size constant (we use $|Pop| = 100$). To determine the solution to be removed, everyone in *Pop* holds a pointer to its nearest solution as well as the distance between them, at phenotypic level. Since, according to this definition, there are always, at least, two solutions holding the minimum distance, the worst of them is the one to be removed. Pointers and distances are updated every time *Pop* changes, i.e., when inserting the new solution and when removing the most crowded one. Notice that the best found solution is never removed.

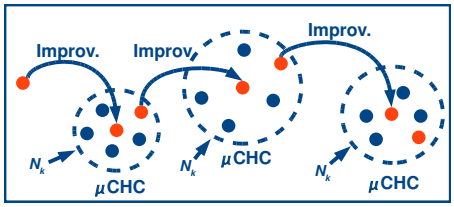

Figure 3: μCHC as shaking module

2.3 μCHC as Shaking Component

In this work, we propose μCHC as the shaking component of VNS. μCHC was firstly presented as an *evolutionary ILS-perturbation technique* for binary-coded problems [19, 20], obtaining promising results with regards to several perturbation techniques proposed in the literature. The role of μCHC is to receive $\boldsymbol{s^c}$, provide local diversity, and generate a solution that is then considered as starting point for the next improvement process (see Fig. 3).

2.3.1 μCHC Model

We have conceived μCHC to be an effective *explorer* in the neighbourhood of $\boldsymbol{s^c}$ to perform shaking process, because it provides local diversity. At the beginning, $\boldsymbol{s^c}$ is used to create its initial population (Sect. 2.3.2). Then, it is performed throughout a predetermined number of fitness function evaluations. The best reached individual is then considered as starting point for the next improvement process (see Fig. 3).

The main components of the algorithm are:

- *Population size:* μCHC manages a population with few individuals ($|Pop| = 5$), and thus, it may be seen as micro EA. In standard VNS models, the number of fitness function evaluations required by the shaking mechanism is very low as compared with the one for the improvement method. With the aim of preserving, as far as possible, the essence of VNS, we have considered an EA with a low sized population; for being able to work adequately under the requirement of spending reduced number of evaluations.

- *Number of evaluations:* In particular, the number of evaluations assigned to μCHC for a particular invocation will be a fixed proportion (p_{evals}), of the number of evaluations consumed by the previously performed improvement method (Continuous Local EA). It is worth noting that p_{evals} should be set to a low value. We will use p_{evals} equals to 0.5.

- *Elitist selection:* Current population is merged with

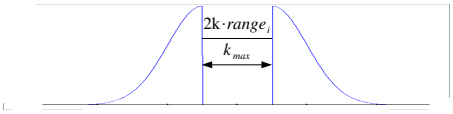

Figure 4: Shape of probability distribution for initialisation

offspring population obtained from it and the best $|Pop|$ individuals are selected for the new population.

- *Incest prevention mechanism*: Before mating, Hamming distance between the corresponding Gray-coding strings of paired individuals (with 20 bits per variable) is calculated. Only paired individuals whose distance exceed a difference threshold d are allowed to undergo crossover operation. Aforementioned threshold is initialised to $L/4$ (with L being the number of bits coding a potential solution). If no offspring is obtained in one generation, difference threshold is decremented by one.

- *BLX-α crossover operator*: Paired individuals allowed to produce offspring undergo BLX-α crossover operator [5], producing one offspring. Given $\boldsymbol{y^1}$ and $\boldsymbol{y^2}$, BLX-α generates offspring $\boldsymbol{z}$, where z_i is a random (uniform) number from $[y^{min} - I \cdot \alpha, y^{max} + I \cdot \alpha]$ interval, with $y^{min} = min(y_i^1, y_i^2)$, $y^{max} = max(y_i^1, y_i^2)$, and $I = |y_i^1 - y_i^2|$. This parameter α is set to 0.5.

- *Mating with $\boldsymbol{s^c}$:* μCHC incorporates the strategy of recombining $\boldsymbol{s^c}$ with another solution [17]. In addition to the typical recombination phase of CHC, our algorithm mates $\boldsymbol{s^c}$ with a member of Pop (selected at random) and, if they are finally crossed over (attending on the incest prevention mechanism), the resulting offspring will be introduced into the offspring population.

- *Cataclysmic mutation:* μCHC, as CHC, uses no mutation in the classical sense of the concept, but instead, it goes through a process of cataclysmic mutation when the population has converged. The difference threshold is considered to measure the stagnation of the search, which happens when it has dropped to one. Then, the population is reinitialised with individuals generated by perturbing $\boldsymbol{s^c}$ attending to the kth neighbourhood structure (Sect. 2.3.2).

2.3.2 Initialisation and Cataclysmic Mutation

Every individual in the initial μCHC population is generated by perturbing $\boldsymbol{s^c}$ according to a relaxed idea of the given neighbourhood structure N_k (see equation 1). In particular, each individual is generated by adding a vector of random values from the following equation (see Fig. 4):

$$N(0,1) \cdot range_i/k_{max} \pm k \cdot range_i/k_{max} \qquad (3)$$

where $N(0,1)$ is a normal random value with mean 0 and variance 1, $range_i$ is the range of ith variable domain, k_{max} is the maximum number of neighbourhood structures VNS manages, and k is the current neighbourhood index; $\pm$ is chosen according to the sign of the random value.

Cataclysmic mutation fills the population with individuals created by the same way as initial population is built, but preserving the best performing individual found in the previous generation. After applying cataclysmic mutation, the difference threshold is set to: $\sigma \cdot (1-\sigma) \cdot L$, with L being the number of bits coding a potential solution of the problem (20 bits per variable) and σ is the average of the minimal radius of neighbourhoods N_k and N_{k+1} $((2k+1)/(2k_{max}))$.

3. RESULTS

Results from experiments according to [12] on the benchmarks functions given in [6, 13] are presented in Figures 5 and 6 and in Tables 1 and 2.

Individual experiments were run with a maximum and adaptive time budget equal to the available time (from 2009-03-15 to 2009-03-23) divided by the number of remaining experiments (initially, $5dims \cdot 30funcs \cdot 15runs$). However, experiments on Cauchy-Noisy functions (version 3.3) were rerun with a reduced time budget. Results of longer runs would be presented at the camera-ready paper submission.

4. CONCLUSIONS

We have presented a continuous VNS model based on three *specialised EAs*: 1) CMA-ES as an EA specialised in generating a good starting point, as generation component, 2) Continuous Local EA, specialised in exploiting local information, as improvement component, 3) and μCHC, which provides local diversity, as shaking component. Experiments have been carried out on the noisy Black-box Optimization Benchmark 2009 testbed.

5. ACKNOWLEDGMENTS

This work was supported by Research Projects TIN2008-05854 and P08-TIC-4173.

6. REFERENCES

[1] A. Auger and N. Hansen. Performance evaluation of an advanced local search evolutionary algorithm. In D. Corne and et al., editors, *Proc. of the IEEE Int. Conf. Evolutionary Computation*, volume 2, pages 1777–1784. IEEE Press, 2005.

[2] I. Charon and O. Hudry. The noising methods: A generalization of some metaheuristics. *Eur. J. Oper. Res.*, 135(1):86–101, 2001.

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

[4] A. Eiben and J. Smith. *Introduction to Evolutionary Computing*. Springer-Verlag, 2003.

[5] L. Eshelman and J. Schaffer. Real-coded genetic algorithms and interval-schemata. In L. Whitley, editor, *Foundations of Genetic Algorithms 2*, pages 187–202. Morgan Kaufmann, 1993.

[6] S. Finck, N. Hansen, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Presentation of the noisy functions. Technical Report 2009/21, Research Center PPE, 2009.

[7] C. García-Martínez and M. Lozano. Local search based on genetic algorithms. In P. Siarry and Z. Michalewicz, editors, *Advances in Metaheuristics for Hard Optimization*, Natural Computing, pages 199–221. Springer, 2008.

[8] C. García-Martínez and M. Lozano. A continuous variable neighbourhood search based on specialised eas: Application to the noiseless bbo-benchmark 2009.

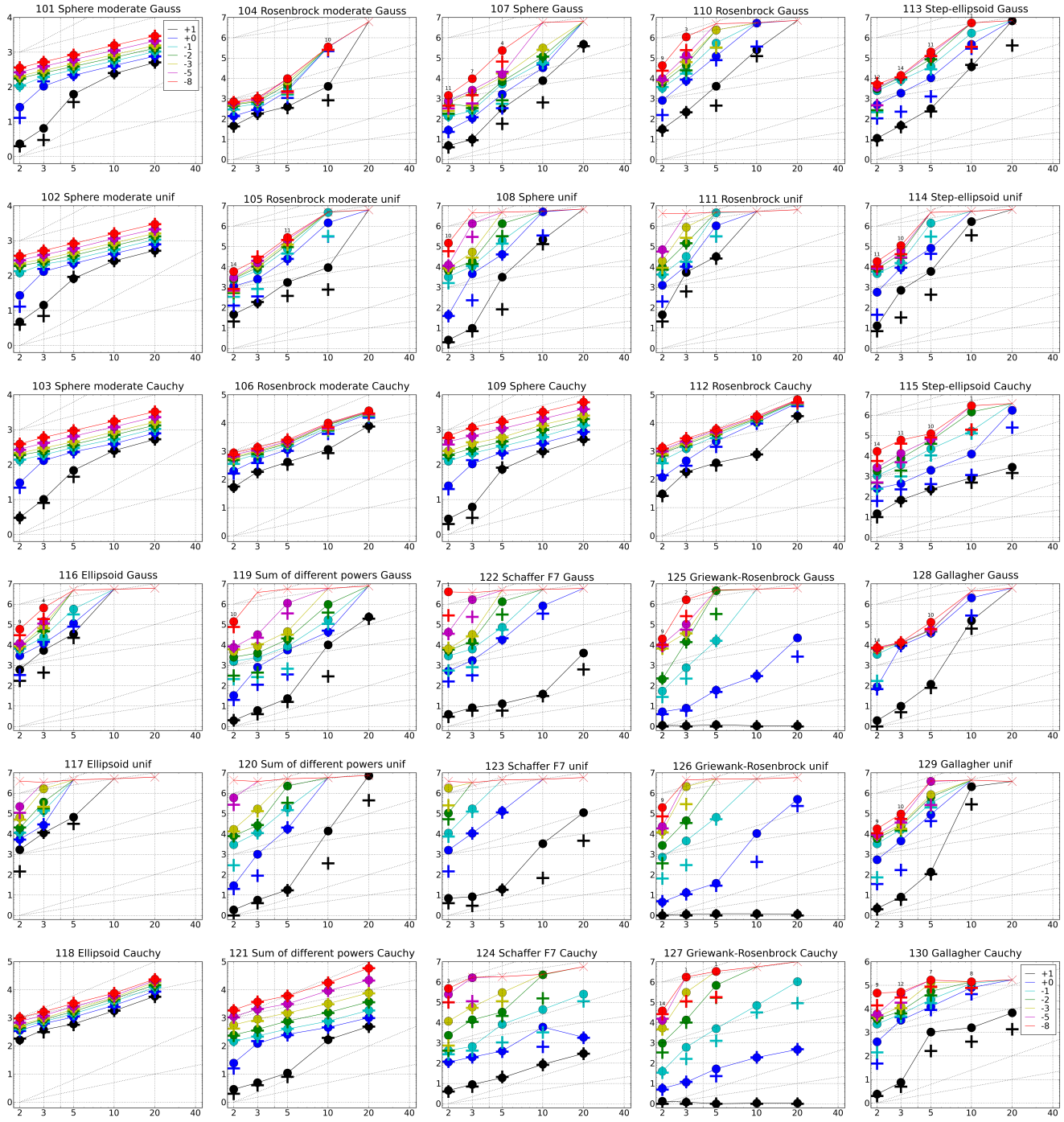

Figure 5: Expected Running Time (ERT, ●) **to reach** $f_{\mathrm{opt}} + \Delta f$ **and median number of function evaluations of successful trials** (+), **shown for** $\Delta f = 10, 1, 10^{-1}, 10^{-2}, 10^{-3}, 10^{-5}, 10^{-8}$ **(the exponent is given in the legend of** f_{101} **and** f_{130}**) versus dimension in log-log presentation. The** $\mathrm{ERT}(\Delta f)$ **equals to** $\#\mathrm{FEs}(\Delta f)$ **divided by the number of successful trials, where a trial is successful if** $f_{\mathrm{opt}} + \Delta f$ **was surpassed during the trial. The** $\#\mathrm{FEs}(\Delta f)$ **are the total number of function evaluations while** $f_{\mathrm{opt}} + \Delta f$ **was not surpassed during the trial from all respective trials (successful and unsuccessful), and** f_{opt} **denotes the optimal function value. Crosses (×) indicate the total number of function evaluations** $\#\mathrm{FEs}(-\infty)$**. Numbers above ERT-symbols indicate the number of successful trials. Annotated numbers on the ordinate are decimal logarithms. Additional grid lines show linear and quadratic scaling.**

Δf	f_{101} in 5-D, N=15, mFE=958 #	ERT	10%	90%	RT_{succ}	f_{101} in 20-D, N=15, mFE=3054 #	ERT	10%	90%	RT_{succ}
10	15	6.2e1	4.4e1	8.0e1	6.2e1	15	5.0e2	4.9e2	5.2e2	5.0e2
1	15	2.2e2	2.0e2	2.3e2	2.2e2	15	7.7e2	7.5e2	7.9e2	7.7e2
1e−1	15	3.1e2	3.0e2	3.2e2	3.1e2	15	1.0e3	1.0e3	1.1e3	1.0e3
1e−3	15	4.7e2	4.5e2	4.8e2	4.7e2	15	1.6e3	1.5e3	1.6e3	1.6e3
1e−5	15	6.1e2	5.9e2	6.3e2	6.1e2	15	2.1e3	2.1e3	2.1e3	2.1e3
1e−8	15	8.4e2	8.3e2	8.5e2	8.4e2	15	2.9e3	2.9e3	2.9e3	2.9e3

Δf	f_{102} in 5-D, N=15, mFE=926 #	ERT	10%	90%	RT_{succ}	f_{102} in 20-D, N=15, mFE=3234 #	ERT	10%	90%	RT_{succ}
10	15	8.3e1	6.4e1	1.0e2	8.3e1	15	5.2e2	5.1e2	5.4e2	5.2e2
1	15	2.2e2	2.1e2	2.3e2	2.2e2	15	7.9e2	7.7e2	8.1e2	7.9e2
1e−1	15	2.9e2	2.8e2	3.1e2	2.9e2	15	1.1e3	1.0e3	1.1e3	1.1e3
1e−3	15	4.6e2	4.5e2	4.7e2	4.6e2	15	1.6e3	1.6e3	1.7e3	1.6e3
1e−5	15	6.0e2	5.8e2	6.1e2	6.0e2	15	2.1e3	2.1e3	2.2e3	2.1e3
1e−8	15	8.3e2	8.1e2	8.5e2	8.3e2	15	3.0e3	2.9e3	3.0e3	3.0e3

Δf	f_{103} in 5-D, N=15, mFE=1158 #	ERT	10%	90%	RT_{succ}	f_{103} in 20-D, N=15, mFE=3666 #	ERT	10%	90%	RT_{succ}
10	15	6.8e1	5.2e1	8.8e1	6.8e1	15	5.3e2	5.1e2	5.4e2	5.3e2
1	15	2.3e2	2.2e2	2.4e2	2.3e2	15	8.0e2	7.8e2	8.1e2	8.0e2
1e−1	15	3.1e2	3.0e2	3.2e2	3.1e2	15	1.1e3	1.0e3	1.1e3	1.1e3
1e−3	15	4.5e2	4.4e2	4.6e2	4.5e2	15	1.6e3	1.6e3	1.7e3	1.6e3
1e−5	15	6.5e2	6.3e2	6.7e2	6.5e2	15	2.3e3	2.3e3	2.4e3	2.3e3
1e−8	15	9.4e2	9.2e2	9.7e2	9.4e2	15	3.3e3	3.2e3	3.3e3	3.3e3

Δf	f_{104} in 5-D, N=15, mFE=112551 #	ERT	10%	90%	RT_{succ}	f_{104} in 20-D, N=15, mFE=531925 #	ERT	10%	90%	RT_{succ}
10	15	4.0e2	3.4e2	4.7e2	4.0e2	0	*18e+0*	*15e+0*	*19e+0*	1.8e5
1	15	2.4e3	1.1e3	3.7e3	2.4e3	.	.	.	.	.
1e−1	15	3.1e3	1.6e3	4.6e3	3.1e3	.	.	.	.	.
1e−3	15	6.9e3	1.9e3	1.2e4	6.9e3	.	.	.	.	.
1e−5	15	9.5e3	2.1e3	1.7e4	9.5e3	.	.	.	.	.
1e−8	15	9.7e3	2.3e3	1.7e4	9.7e3	.	.	.	.	.

Δf	f_{105} in 5-D, N=15, mFE=399225 #	ERT	10%	90%	RT_{succ}	f_{105} in 20-D, N=15, mFE=501086 #	ERT	10%	90%	RT_{succ}
10	15	1.7e3	3.6e2	3.1e3	1.7e3	0	*18e+0*	*17e+0*	*19e+0*	2.0e5
1	15	2.4e4	1.8e4	3.1e4	2.4e4	.	.	.	.	.
1e−1	15	7.6e4	5.3e4	1.0e5	7.6e4	.	.	.	.	.
1e−3	14	1.6e5	1.2e5	1.9e5	1.4e5	.	.	.	.	.
1e−5	13	2.2e5	1.7e5	2.6e5	2.0e5	.	.	.	.	.
1e−8	11	2.8e5	2.1e5	3.3e5	2.3e5	.	.	.	.	.

Δf	f_{106} in 5-D, N=15, mFE=4342 #	ERT	10%	90%	RT_{succ}	f_{106} in 20-D, N=15, mFE=68186 #	ERT	10%	90%	RT_{succ}
10	15	4.1e2	3.4e2	4.9e2	4.1e2	15	7.9e3	7.5e3	8.4e3	7.9e3
1	15	1.1e3	9.6e2	1.3e3	1.1e3	15	2.1e4	1.6e4	2.6e4	2.1e4
1e−1	15	1.6e3	1.4e3	1.8e3	1.6e3	15	2.3e4	1.8e4	2.7e4	2.3e4
1e−3	15	2.0e3	1.8e3	2.2e3	2.0e3	15	2.5e4	2.0e4	3.0e4	2.5e4
1e−5	15	2.2e3	2.0e3	2.4e3	2.2e3	15	2.6e4	2.1e4	3.1e4	2.6e4
1e−8	15	2.5e3	2.3e3	2.7e3	2.5e3	15	2.7e4	2.2e4	3.1e4	2.7e4

Δf	f_{107} in 5-D, N=15, mFE=131770 #	ERT	10%	90%	RT_{succ}	f_{107} in 20-D, N=15, mFE=579983 #	ERT	10%	90%	RT_{succ}
10	15	3.2e2	5.5e1	5.8e2	3.2e2	8	4.8e5	3.7e5	5.8e5	2.7e5
1	15	1.6e3	3.9e2	2.9e3	1.6e3	0	*96e−1*	*46e−1*	*28e+0*	1.8e5
1e−1	15	5.3e3	2.8e3	7.9e3	5.3e3	.	.	.	.	.
1e−3	15	1.2e4	7.8e3	1.5e4	1.2e4	.	.	.	.	.
1e−5	15	1.6e4	1.1e4	2.0e4	1.6e4	.	.	.	.	.
1e−8	4	2.4e5	1.8e5	2.8e5	5.7e4	.	.	.	.	.

Δf	f_{108} in 5-D, N=15, mFE=341788 #	ERT	10%	90%	RT_{succ}	f_{108} in 20-D, N=15, mFE=587280 #	ERT	10%	90%	RT_{succ}
10	15	3.1e3	1.1e2	5.8e3	3.1e3	0	*61e+0*	*30e+0*	*88e+0*	1.8e5
1	15	4.1e4	3.5e4	4.8e4	4.1e4	.	.	.	.	.
1e−1	13	1.9e5	1.5e5	2.3e5	1.7e5	.	.	.	.	.
1e−3	0	*48e−3*	*71e−4*	*24e−2*	8.9e4	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	f_{109} in 5-D, N=15, mFE=2734 #	ERT	10%	90%	RT_{succ}	f_{109} in 20-D, N=15, mFE=7014 #	ERT	10%	90%	RT_{succ}
10	15	7.4e1	5.8e1	9.2e1	7.4e1	15	5.4e2	5.2e2	5.5e2	5.4e2
1	15	2.2e2	2.1e2	2.3e2	2.2e2	15	8.6e2	8.4e2	8.8e2	8.6e2
1e−1	15	3.3e2	3.2e2	3.6e2	3.3e2	15	1.4e3	1.3e3	1.4e3	1.4e3
1e−3	15	6.0e2	5.6e2	6.4e2	6.0e2	15	2.6e3	2.5e3	2.7e3	2.6e3
1e−5	15	1.1e3	1.0e3	1.1e3	1.1e3	15	3.9e3	3.8e3	4.0e3	3.9e3
1e−8	15	1.7e3	1.6e3	1.9e3	1.7e3	15	6.2e3	6.0e3	6.3e3	6.2e3

Δf	f_{110} in 5-D, N=15, mFE=340707 #	ERT	10%	90%	RT_{succ}	f_{110} in 20-D, N=15, mFE=598812 #	ERT	10%	90%	RT_{succ}
10	15	4.1e3	1.6e3	6.7e3	4.1e3	0	*10e+1*	*55e+0*	*25e+1*	2.5e5
1	14	1.2e5	8.5e4	1.6e5	1.1e5	.	.	.	.	.
1e−1	7	5.5e5	4.7e5	6.2e5	2.4e5	.	.	.	.	.
1e−3	2	2.4e6	2.3e6	2.5e6	3.0e5	.	.	.	.	.
1e−5	0	*10e−2*	*34e−5*	*80e−2*	1.8e5	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	f_{111} in 5-D, N=15, mFE=331678 #	ERT	10%	90%	RT_{succ}	f_{111} in 20-D, N=15, mFE=562864 #	ERT	10%	90%	RT_{succ}
10	15	3.2e4	2.5e4	3.9e4	3.2e4	0	*64e+2*	*13e+2*	*13e+3*	1.8e5
1	4	1.0e6	9.3e5	1.1e6	2.7e5	.	.	.	.	.
1e−1	1	4.5e6	4.2e6	4.8e6	3.2e5	.	.	.	.	.
1e−3	0	*16e−1*	*72e−2*	*40e−1*	8.9e4	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	f_{112} in 5-D, N=15, mFE=15335 #	ERT	10%	90%	RT_{succ}	f_{112} in 20-D, N=15, mFE=123946 #	ERT	10%	90%	RT_{succ}
10	15	3.9e2	3.3e2	4.5e2	3.9e2	15	1.8e4	1.7e4	1.9e4	1.8e4
1	15	2.2e3	1.4e3	3.2e3	2.2e3	15	5.2e4	4.3e4	6.1e4	5.2e4
1e−1	15	3.5e3	2.6e3	4.5e3	3.5e3	15	5.7e4	4.8e4	6.6e4	5.7e4
1e−3	15	4.7e3	3.8e3	5.7e3	4.7e3	15	6.2e4	5.3e4	7.1e4	6.2e4
1e−5	15	5.2e3	4.3e3	6.2e3	5.2e3	15	6.4e4	5.5e4	7.3e4	6.4e4
1e−8	15	5.9e3	5.0e3	6.9e3	5.9e3	15	6.7e4	5.8e4	7.6e4	6.7e4

Δf	f_{113} in 5-D, N=15, mFE=343674 #	ERT	10%	90%	RT_{succ}	f_{113} in 20-D, N=15, mFE=623984 #	ERT	10%	90%	RT_{succ}
10	15	3.2e2	2.4e2	4.0e2	3.2e2	1	6.8e6	6.4e6	7.2e6	5.6e5
1	15	1.1e4	6.9e3	1.4e4	1.1e4	0	*31e+0*	*13e+0*	*80e+0*	2.0e5
1e−1	15	3.9e4	2.5e4	5.5e4	3.9e4	.	.	.	.	.
1e−3	14	1.6e5	1.3e5	1.9e5	1.6e5	.	.	.	.	.
1e−5	14	1.6e5	1.2e5	1.9e5	1.6e5	.	.	.	.	.
1e−8	11	2.0e5	1.6e5	2.5e5	1.5e5	.	.	.	.	.

Δf	f_{114} in 5-D, N=15, mFE=333009 #	ERT	10%	90%	RT_{succ}	f_{114} in 20-D, N=15, mFE=575611 #	ERT	10%	90%	RT_{succ}
10	15	6.0e3	3.6e3	8.7e3	6.0e3	0	*18e+1*	*98e+0*	*29e+1*	2.0e5
1	15	8.2e4	5.9e4	1.1e5	8.2e4	.	.	.	.	.
1e−1	3	1.4e6	1.3e6	1.5e6	3.1e5	.	.	.	.	.
1e−3	0	*28e−2*	*22e−3*	*60e−2*	1.4e5	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	f_{115} in 5-D, N=15, mFE=163969 #	ERT	10%	90%	RT_{succ}	f_{115} in 20-D, N=15, mFE=258564 #	ERT	10%	90%	RT_{succ}
10	15	2.3e2	2.1e2	2.4e2	2.3e2	15	2.8e3	1.5e3	4.1e3	2.8e3
1	15	2.0e3	8.5e2	3.2e3	2.0e3	2	1.7e6	1.6e6	1.8e6	2.4e5
1e−1	15	2.2e4	1.3e4	3.2e4	2.2e4	0	*21e−1*	*98e−2*	*34e−1*	1.1e5
1e−3	14	6.5e4	4.9e4	8.3e4	6.5e4	.	.	.	.	.
1e−5	14	6.5e4	4.8e4	8.2e4	6.5e4	.	.	.	.	.
1e−8	10	1.2e5	9.4e4	1.6e5	9.7e4	.	.	.	.	.

Δf	f_{116} in 5-D, N=15, mFE=343038 #	ERT	10%	90%	RT_{succ}	f_{116} in 20-D, N=15, mFE=455841 #	ERT	10%	90%	RT_{succ}
10	15	3.5e4	2.4e4	4.7e4	3.5e4	0	*27e+2*	*81e+1*	*56e+2*	1.8e5
1	14	1.1e5	8.5e4	1.4e5	9.8e4	.	.	.	.	.
1e−1	7	5.7e5	5.2e5	6.3e5	3.0e5	.	.	.	.	.
1e−3	0	*11e−2*	*26e−3*	*87e−2*	1.8e5	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	f_{117} in 5-D, N=15, mFE=316452 #	ERT	10%	90%	RT_{succ}	f_{117} in 20-D, N=15, mFE=528164 #	ERT	10%	90%	RT_{succ}
10	14	6.6e4	3.5e4	1.0e5	4.5e4	0	*67e+2*	*36e+2*	*19e+3*	1.8e5
1	0	*28e−1*	*13e−1*	*76e−1*	7.1e4	.	.	.	.	.
1e−1	.	.	.	.	.	.	.	.	.	.
1e−3	.	.	.	.	.	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Δf	f_{118} in 5-D, N=15, mFE=4078 #	ERT	10%	90%	RT_{succ}	f_{118} in 20-D, N=15, mFE=27594 #	ERT	10%	90%	RT_{succ}
10	15	6.1e2	5.6e2	6.5e2	6.1e2	15	5.9e3	5.6e3	6.2e3	5.9e3
1	15	1.1e3	9.6e2	1.2e3	1.1e3	15	8.9e3	8.3e3	9.5e3	8.9e3
1e−1	15	1.4e3	1.3e3	1.5e3	1.4e3	15	1.2e4	1.1e4	1.3e4	1.2e4
1e−3	15	2.0e3	1.9e3	2.1e3	2.0e3	15	1.9e4	1.8e4	2.0e4	1.9e4
1e−5	15	2.5e3	2.4e3	2.7e3	2.5e3	15	2.2e4	2.1e4	2.2e4	2.2e4
1e−8	15	3.3e3	3.1e3	3.4e3	3.3e3	15	2.4e4	2.3e4	2.5e4	2.4e4

Δf	f_{119} in 5-D, N=15, mFE=388060 #	ERT	10%	90%	RT_{succ}	f_{119} in 20-D, N=15, mFE=613187 #	ERT	10%	90%	RT_{succ}
10	15	2.3e1	1.5e1	3.3e1	2.3e1	13	2.3e5	1.7e5	2.9e5	2.2e5
1	15	5.6e3	2.9e3	8.7e3	5.6e3	0	*57e−1*	*35e−1*	*11e+0*	2.2e5
1e−1	15	8.3e3	5.0e3	1.2e4	8.3e3	.	.	.	.	.
1e−3	15	4.5e4	3.4e4	5.7e4	4.5e4	.	.	.	.	.
1e−5	4	1.1e6	9.7e5	1.2e6	3.0e5	.	.	.	.	.
1e−8	0	*34e−6*	*37e−7*	*56e−6*	2.2e5	.	.	.	.	.

Δf	f_{120} in 5-D, N=15, mFE=351395 #	ERT	10%	90%	RT_{succ}	f_{120} in 20-D, N=15, mFE=616785 #	ERT	10%	90%	RT_{succ}
10	15	1.8e1	1.3e1	2.3e1	1.8e1	1	7.2e6	6.6e6	7.8e6	5.9e5
1	15	1.8e4	1.4e4	2.2e4	1.8e4	0	*15e+0*	*10e+0*	*20e+0*	2.0e5
1e−1	12	1.8e5	1.3e5	2.4e5	1.7e5	.	.	.	.	.
1e−3	0	*62e−3*	*98e−4*	*15e−2*	1.4e5	.	.	.	.	.
1e−5	.	.	.	.	.	.	.	.	.	.
1e−8	.	.	.	.	.	.	.	.	.	.

Table 1: Shown are, for functions f_{101}-f_{120} and for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{\text{opt}} + \Delta f$ (ERT, see Figure 5); the 10%-tile and 90%-tile of the bootstrap distribution of ERT; the average number of function evaluations in successful trials or, if none was successful, as last entry the median number of function evaluations to reach the best function value (RT_{succ}). If $f_{\text{opt}} + \Delta f$ was never reached, figures in *italics* denote the best achieved Δf-value of the median trial and the 10% and 90%-tile trial. Furthermore, N denotes the number of trials, and mFE denotes the maximum of number of function evaluations executed in one trial. See Figure 5 for the names of functions.

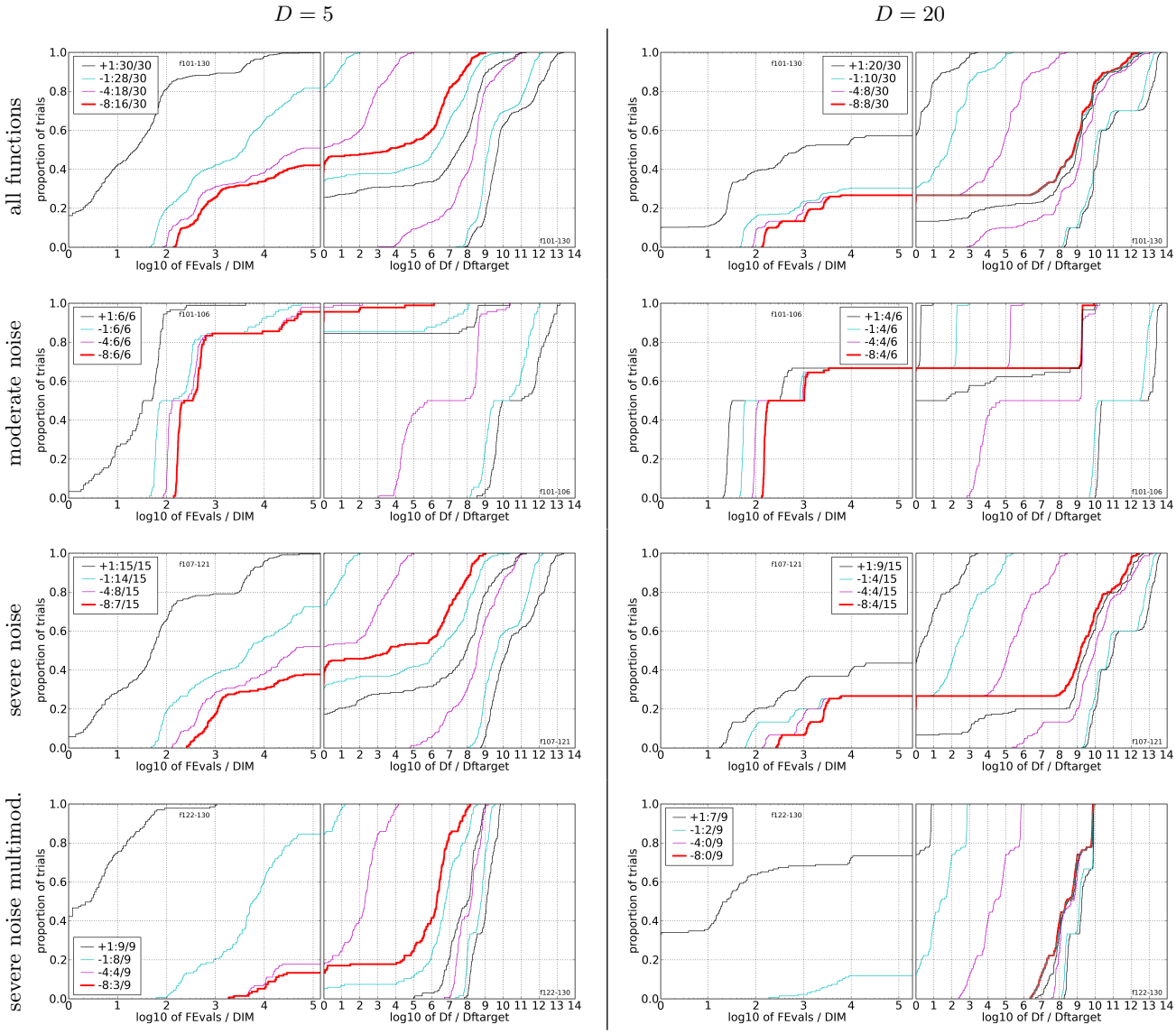

Figure 6: Empirical cumulative distribution functions (ECDFs), plotting the fraction of trials versus running time (left) or Δf. Left subplots: ECDF of the running time (number of function evaluations), divided by search space dimension D, to fall below $f_{\mathrm{opt}} + \Delta f$ with $\Delta f = 10^k$, where k is the first value in the legend. Right subplots: ECDF of the best achieved Δf divided by 10^k (upper left lines in continuation of the left subplot), and best achieved Δf divided by 10^{-8} for running times of $D, 10\,D, 100\,D \ldots$ function evaluations (from right to left cycling black-cyan-magenta). Top row: all results from all functions; second row: moderate noise functions; third row: severe noise functions; fourth row: severe noise and highly-multimodal functions. The legends indicate the number of functions that were solved in at least one trial. FEvals denotes number of function evaluations, D and DIM denote search space dimension, and Δf and Df denote the difference to the optimal function value.

Δf	#	ERT	10%	90%	RT_{succ}
f_{121} in 5-D, N=15, mFE=8902					
10	15	1.1e1	7.9e0	1.4e1	1.1e1
1	15	2.6e2	2.4e2	2.8e2	2.6e2
1e−1	15	4.0e2	3.8e2	4.2e2	4.0e2
1e−3	15	1.5e3	1.4e3	1.6e3	1.5e3
1e−5	15	3.1e3	2.9e3	3.4e3	3.1e3
1e−8	15	6.1e3	5.7e3	6.4e3	6.1e3
f_{121} in 20-D, N=15, mFE=69510					
10	15	4.9e2	4.6e2	5.3e2	4.9e2
1	15	1.0e3	9.7e2	1.1e3	1.0e3
1e−1	15	1.9e3	1.8e3	1.9e3	1.9e3
1e−3	15	7.7e3	7.5e3	8.0e3	7.7e3
1e−5	15	2.2e4	2.1e4	2.3e4	2.2e4
1e−8	15	5.9e4	5.6e4	6.1e4	5.9e4
f_{123} in 5-D, N=15, mFE=310600					
10	15	1.9e1	1.6e1	2.3e1	1.9e1
1	13	1.3e5	9.9e4	1.7e5	1.0e5
1e−1	0	*61e−2*	*31e−2*	*11e−1*	1.3e5
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{123} in 20-D, N=15, mFE=416895					
10	14	1.1e5	7.0e4	1.5e5	1.1e5
1	0	*79e−1*	*54e−1*	*98e−1*	1.8e5
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{125} in 5-D, N=15, mFE=340655					
10	15	1.2e0	1.0e0	1.4e0	1.2e0
1	15	6.0e1	4.4e1	7.5e1	6.0e1
1e−1	15	1.6e4	1.2e4	2.1e4	1.6e4
1e−3	0	*27e−3*	*13e−3*	*45e−3*	1.4e5
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{125} in 20-D, N=15, mFE=548740					
10	15	1.1e0	1.0e0	1.1e0	1.1e0
1	15	2.2e4	4.5e3	4.1e4	2.2e4
1e−1	0	*67e−2*	*49e−2*	*81e−2*	1.3e5
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{127} in 5-D,N=15, mFE=397096					
10	15	1.0e0	1.0e0	1.0e0	1.0e0
1	15	5.2e1	3.7e1	6.8e1	5.2e1
1e−1	15	5.0e3	3.0e3	7.3e3	5.0e3
1e−3	1	3.3e6	2.8e6	3.8e6	2.5e5
1e−5	1	3.3e6	2.8e6	3.9e6	2.5e5
1e−8	1	3.3e6	2.8e6	3.9e6	2.5e5
f_{127} in 20-D,N=15, mFE=2165166					
10	15	1.1e0	1.0e0	1.1e0	1.1e0
1	15	4.7e2	4.4e2	5.0e2	4.7e2
1e−1	5	1.0e6	5.1e5	1.6e6	2.0e5
1e−3	0	*12e−2*	*54e−3*	*20e−2*	2.5e5
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{129} in 5-D, N=15, mFE=283533					
10	15	1.3e2	1.0e2	1.7e2	1.3e2
1	14	8.9e4	6.2e4	1.2e5	8.4e4
1e−1	9	3.2e5	2.7e5	3.6e5	1.8e5
1e−3	4	8.4e5	7.4e5	9.4e5	2.1e5
1e−5	1	3.8e6	3.6e6	4.0e6	2.6e5
1e−8	0	*87e−3*	*11e−5*	*53e−2*	1.6e5
f_{129} in 20-D, N=15, mFE=429291					
10	0	*69e+0*	*64e+0*	*73e+0*	7.1e2
1	.	.	.	.	.
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.

Δf	#	ERT	10%	90%	RT_{succ}
f_{122} in 5-D, N=15, mFE=340028					
10	15	1.3e1	9.0e0	1.7e1	1.3e1
1	15	1.8e4	1.4e4	2.3e4	1.8e4
1e−1	15	7.3e4	5.5e4	9.2e4	7.3e4
1e−3	0	*20e−3*	*40e−4*	*75e−3*	1.0e5
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{122} in 20-D, N=15, mFE=437181					
10	15	4.0e3	1.2e3	6.8e3	4.0e3
1	0	*58e−1*	*22e−1*	*72e−1*	1.8e5
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{124} in 5-D, N=15, mFE=154876					
10	15	1.9e1	1.5e1	2.3e1	1.9e1
1	15	4.0e2	3.6e2	4.3e2	4.0e2
1e−1	15	8.2e3	5.1e3	1.2e4	8.2e3
1e−3	5	3.0e5	2.6e5	3.3e5	1.2e5
1e−5	0	*12e−4*	*30e−5*	*63e−4*	3.5e4
1e−8	.	.	.	.	.
f_{124} in 20-D, N=15, mFE=769405					
10	15	3.0e2	2.7e2	3.2e2	3.0e2
1	15	1.9e3	1.7e3	2.1e3	1.9e3
1e−1	11	2.5e5	1.7e5	3.5e5	2.1e5
1e−3	0	*72e−3*	*28e−3*	*24e−2*	1.4e5
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{126} in 5-D, N=15, mFE=344618					
10	15	1.2e0	1.1e0	1.3e0	1.2e0
1	15	3.8e1	2.5e1	5.0e1	3.8e1
1e−1	15	6.5e4	4.5e4	8.8e4	6.5e4
1e−3	0	*52e−3*	*27e−3*	*85e−3*	1.3e5
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{126} in 20-D, N=15, mFE=425667					
10	15	1.1e0	1.0e0	1.3e0	1.1e0
1	8	5.0e5	4.2e5	5.8e5	2.5e5
1e−1	0	*97e−2*	*75e−2*	*12e−1*	1.8e5
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{128} in 5-D, N=15, mFE=232211					
10	15	1.2e2	8.0e1	1.5e2	1.2e2
1	15	3.8e4	3.4e4	4.3e4	3.8e4
1e−1	15	5.2e4	4.2e4	6.2e4	5.2e4
1e−3	15	5.3e4	4.4e4	6.4e4	5.3e4
1e−5	15	5.6e4	4.6e4	6.6e4	5.6e4
1e−8	10	1.3e5	1.0e5	1.6e5	8.4e4
f_{128} in 20-D, N=15, mFE=463842					
10	0	*68e+0*	*61e+0*	*71e+0*	2.0e5
1	.	.	.	.	.
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.
f_{130} in 5-D, N=15, mFE=111163					
10	15	1.0e3	4.4e2	1.6e3	1.0e3
1	15	1.2e4	9.1e3	1.6e4	1.2e4
1e−1	15	2.5e4	1.8e4	3.3e4	2.5e4
1e−3	8	1.3e5	1.1e5	1.5e5	8.9e4
1e−5	7	1.7e5	1.4e5	1.9e5	9.2e4
1e−8	7	1.7e5	1.4e5	1.9e5	9.2e4
f_{130} in 20-D, N=15, mFE=12870					
10	10	6.8e3	4.6e3	9.0e3	4.3e3
1	0	*81e−1*	*19e−1*	*30e+0*	7.9e3
1e−1	.	.	.	.	.
1e−3	.	.	.	.	.
1e−5	.	.	.	.	.
1e−8	.	.	.	.	.

Table 2: Shown are, for functions f_{121}-f_{130} and for a given target difference to the optimal function value Δf: the number of successful trials (#); the expected running time to surpass $f_{\text{opt}} + \Delta f$ (ERT, see Figure 5); the 10%-tile and 90%-tile of the bootstrap distribution of ERT; the average number of function evaluations in successful trials or, if none was successful, as last entry the median number of function evaluations to reach the best function value (RT_{succ}). If $f_{\text{opt}} + \Delta f$ was never reached, figures in *italics* denote the best achieved Δf-value of the median trial and the 10% and 90%-tile trial. Furthermore, N denotes the number of trials, and mFE denotes the maximum of number of function evaluations executed in one trial. See Figure 5 for the names of functions.

In *Genetic and Evolutionary Computation Conf.*, page Submitted, 2009.

[9] C. García-Martínez, M. Lozano, F. Herrera, D. Molina, and A. Sánchez. Global and local real-coded genetic algorithms based on parent-centric crossover operators. *Eur. J. Oper. Res.*, 185(3):1088–1113, 2008.

[10] C. García-Martínez, M. Lozano, and D. Molina. A local genetic algorithm for binary-coded problems. In T. Runarsson and et al., editors, *Proc. of the Int. Conf. on Parallel Problem Solving from Nature*, volume 4193 of *LNCS*, pages 192–201. Springer, 2006.

[11] F. Glover and G. Kochenberger, editors. *Handbook of Metaheuristics*. Kluwer Academic Publishers, 2003.

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

[13] N. Hansen, S. Finck, R. Ros, and A. Auger. Real-parameter black-box optimization benchmarking 2009: Noisy functions definitions. Technical Report RR-6869, INRIA, 2009.

[14] N. Hansen, S. Müller, and P. Koumoutsakos. Reducing the time complexity of the derandomized evolution strategy with covariance matrix adaptation (CMA-ES). *Evol. Comput.*, 11(1):1–18, 2003.

[15] N. Hansen and A. Ostermeier. Completely derandomized self-adaptation in evolution strategies. *Evol. Comput.*, 9(2):159–195, 2001.

[16] F. Herrera, M. Lozano, and J. Verdegay. Tackling real-coded genetic algorithms: operators and tools for behavioral analysis. *Artif. Intell. Rev.*, 12(4):265–319, 1998.

[17] K. Katayama and H. Narihisa. A new iterated local search algorithm using genetic crossover for the travelling salesman problem. In *ACM Symposium on Applied Computing*, pages 302–306, 1999.

[18] S. Kirkpatrick, C. Gelatt Jr, and M. Vecchi. Optimization by simulated annealing. *Sci.*, 220(4598):671–680, 1983.

[19] M. Lozano and C. García-Martínez. An evolutionary ILS-perturbation technique. In *Proc. of the Int. Workshop on Hybrid Metaheuristics, LNCS*, volume 5296, pages 1–15, 2008.

[20] M. Lozano and C. García-Martínez. Hybrid metaheuristics with evolutionary algorithms specializing in intensification and diversification: Overview and progress report. *Comput. Oper. Res.*, page In press, 2009.

[21] M. Lozano, F. Herrera, N. Krasnogor, and D. Molina. Real-coded memetic algorithms with crossover hill-climbing. *Evol. Comput.*, 12(3):273–302, 2004.

[22] N. Mladenovic and P. Hansen. Variable neighborhood search. *Comput. Oper. Res.*, 24:1097–1100, 1997.

[23] E. Talbi. A taxonomy of hybrid metaheuristics. *J. Heuristics*, 8(5):541–564, 2002.