Modern Continuous Optimization Algorithms for Tuning Real and Integer Algorithm Parameters

Zhi Yuan, Marco A. Montes de Oca, Thomas Stützle, and Mauro Birattari

IRIDIA – Technical Report Series

Technical Report No.
TR/IRIDIA/2010-010

April 2010

IRIDIA – Technical Report Series
ISSN 1781-3794

Published by:

IRIDIA, *Institut de Recherches Interdisciplinaires et de Développements en Intelligence Artificielle*
Université Libre de Bruxelles
Av F. D. Roosevelt 50, CP 194/6
1050 Bruxelles, Belgium

Technical report number TR/IRIDIA/2010-010

Modern Continuous Optimization Algorithms for Tuning Real and Integer Algorithm Parameters

Zhi Yuan, Marco A. Montes de Oca, Thomas Stützle, and Mauro Birattari

IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
{zyuan,mmontes,stuetzle,mbiro}@ulb.ac.be

Abstract. To design high performing algorithms in swarm intelligence requires frequently the choice of appropriate parameter settings. Frequently, the parameters to be set are either continuous ones or have a large integer domain. For such tasks it seems therefore interesting to apply state-of-the-art continuous optimization algorithms instead of using a tedious and error-prone hands-on approach. In this article we study the performance of various continuous optimization algorithms for this algorithm configuration task using various case studies of algorithms to be configured from the swarm intelligence literature.

1 Introduction

Assigning appropriate values to the parameters of optimization algorithms is a task that frequently arises as part of the solution of application problems. This task has traditionally been tackled by software solution architects, who have knowledge about the application problem, but who do not necessarily have knowledge about the specific optimization algorithms employed. When optimization algorithm designers are involved, they have to learn many details about the application problem before suggesting a set of parameter values that they believe will provide good results. In any case, a significant amount of human effort is devoted to the solution of the parameter tuning problem [1].

Tackling algorithmically the parameter tuning problem, or more in general, the algorithm configuration problem [2], is of practical relevance because it offers the possibility of freeing software architects and designers from this time-consuming task. This can be done by casting these problems as an optimization problem, in which the goal is to find the algorithmic components and their parameter settings that optimize some performance statistics (e.g., the average performance) on typical instances of the application problem. Common performance measures are the solution quality reached after a specific computation time limit, or the computation time necessary for reaching a target bound on the solution quality. A number of algorithms have been proposed for this task over the years. Early attempts focused on finding good values to numerical parameters (e.g., the mutation rate in a genetic algorithm) through meta-evolutionary algorithms [3, 4]. This effort has continued in recent years with methods such as CALIBRA [1], REVAC [5], *SPO* and SPO^+ [6, 7]. Methods for setting numerical as well as categorical parameters (e.g., the type of local search in a memetic algorithm) have also been proposed. Examples are F-Race and iterated F-Race [8–11], ParamILS [2], genetic programming [12, 13], and gender-based genetic algorithms [14].

While the above-mentioned methods have proved their potential, we believe that more effective configuration algorithms can be developed if the numerical part of the general algorithm configuration problem is treated and tackled as a stochastic continuous optimization problem. In this paper, we investigate the effectiveness of modern continuous optimization techniques for tackling the problem of setting numerical parameters of optimization algorithms. In particular, we tackle configuration tasks that involve continuous parameters, such as the temperature in simulated annealing or the pheromone evaporation rate in an ant colony optimization algorithm, and integer parameters, such as the population size in evolutionary algorithms or the perturbation strength in iterated local search. We treat integer parameters as "quasi-continuous", that is, we assume that the real-valued numbers given by continuous optimization techniques can be rounded to the nearest integer. This is a reasonable approach when the domain of the integer parameter is large.

The configuration tasks that we consider in this paper are rather small: we tackle problems with 2 to 6 numerical parameters. The rationale for limiting the number of parameters is the following. First, assuming that the configuration problem is part of a larger algorithm configuration problem with the values of categorical parameters fixed, we would like to explore the effectiveness of modern continuous optimization techniques as sub-solvers for exploring the search space of the remaining numerical parameters. The categorical parameters, which are typical for many configuration tasks, would then be handled by another solver at a higher level. The goal would be to explore the domains of the continuous or quasi-continuous parameters using as few evaluations as possible. Second, there are various algorithm configuration tasks, which involve only continuous or quasi-continuous parameters. An example is to adapt an already defined algorithm to tackle a new class of problem instances or to a new application situation. In these cases, before proceeding with the re-design of an algorithm, a first option may be to simply tune the algorithm's continuous or quasi-continuous parameters.

This article is structured as follows. In the next section, we give an overview of the algorithms we chose as candidate solvers. In Section 3, we introduce concisely the experimental setup and the benchmark domains on which we tested these algorithms. Results are described in Section 4. We conclude in Section 5.

2 Configuration Algorithms

For the configuration task, we selected state-of-the-art black-box continuous optimization algorithms developed in the mathematical programming as well as in the evolutionary computation communities. However, since these algorithms were not explicitly designed for tackling stochastic problems, we modified them in such a way as to make them able to deal with noise. For comparison reference, we also included simple and iterated random sampling in our experiments.

2.1 Basic Algorithms

Simple and Iterated Random Sampling (RS & IRS). The performance of simple and iterated random sampling is used as a baseline for evaluation. The simple random sampling technique explores the space of possible parameter settings uniformly at random. The obvious drawback of this approach is its inability to focus the search in promising regions of the search space. The method we refer to as iterated random sampling is the one described in [9], which consists in executing the steps of solution generation, selection and refinement iteratively. The solution generation step involves sampling from Gaussian distributions centered at promising solutions and with standard deviations that vary over time in order to focus the search around the best-so-far solutions.

Bound Optimization by Quadratic Approximation (BOBYQA). This is a derivative-free optimization algorithm based on the trust region paradigm [15]. It is an extension of the NEWUOA [16] algorithm that is able to deal with bound constraints. At each iteration, BOBYQA computes and minimizes a quadratic model that interpolates m automatically-generated points in the current trust region. Then, either the best-so-far solution, or the trust region radius is updated. The recommended number of points to compute the quadratic model is $m = 2d + 1$ [15], where d is the dimensionality of the search space. NEWUOA, and by extension BOBYQA, is considered to be a state-of-the-art continuous optimization technique [17, 18]. The initial and final trust region radii, as well as a maximum number of function evaluations before termination are parameters.

Mesh Adaptive Direct Search (MADS). In MADS [19], a number of trial points lying on a *mesh* is generated and evaluated around the best-so-far solution at every iteration. If a new better solution is found, the next iteration begins with, possibly, a coarser mesh. If a better solution is not found after this first step, trial points from a refined mesh are generated and evaluated. In this second step, MADS differs from the generalized pattern search class of algorithms [20] in that MADS allows a more flexible exploration of this refined mesh by allowing the evaluation of points at different distances from the best-so-far solution. When a new best solution is found the algorithm iterates.

Covariance Matrix Adaptation Evolution Strategy (CMA-ES). In CMA-ES [21], candidate solutions are sampled at each iteration from a multivariate Gaussian distribution. The main characteristic of CMA-ES is that the parameters of this distribution are adapted as the optimization process progresses. The mean of the sampling distribution is centered at a linear combination of the current "parent" population. The covariance matrix is updated using information from the trajectory the best solutions have followed so far. The aim of this transformation is to increase the chances of sampling improving solutions. CMA-ES is considered to be a state-of-the-art evolutionary algorithm [18].

2.2 Enhancing Noise Tolerance

The problem of configuring the parameters of a stochastic local search algorithm can be seen as a stochastic optimization problem. The sources of stochasticity are the randomized nature of the algorithm itself and the "sampling" of the problem instance tackled. We enhanced the algorithms described above with mechanisms for better estimating the real difference between two or more candidate solutions. These mechanisms are described below.

Repeated Evaluation. The simplest approach to deal with noise in the evaluation of an objective function is to evaluate it more than once and return the average evaluation as the closest estimate of the true value. We denote nr the number of times that each evaluation is repeated. The advantages of this approach are its simplicity and the confidence that can be associated to the estimate as a function of the number of repeated evaluations. We tried this approach with all methods using different numbers of repeated evaluations. The main disadvantage of this technique is that it is blind to the actual quality of the solutions being re-evaluated, and thus many function evaluations can be wasted.

F-Race. It is a technique aimed at making a more efficient use of computational power than repeated evaluation in the presence of noise. Given a set of candidate solutions and a noisy objective function, the goal of F-Race is to discard those solutions for which sufficient statistical evidence against them has been gathered. The elimination process continues until one single solution remains, or the maximum number of evaluations is reached. The composition of the initial set of candidate solutions is independent of the elimination mechanism of F-Race. It is thus possible to integrate it with any method that needs to select the best solutions from a given set. For this reason, F-Race is used with RS, IRS, MADS, and CMA-ES. F-Race is not used with BOBYQA because this algorithm generates one single trial point per iteration and does not need to select the best out of a set as the other algorithms do. More information about F-Race can be found in [8–11].

3 Benchmark configuration problems

We have compared the performance of the continuous optimization algorithms described in Section 2 on six benchmark *configuration problems*. Each *configuration problem* consists of a parameterized algorithm to be configured, and an optimization problem to which this algorithm is applied. The six configuration problems are originated from three classes of case studies with three underlying algorithms to be configured. Two of them belong to the Swarm Intelligence algorithms, Ant Colony Optimization (ACO) and Particle Swarm Optimization (PSO), and another algorithm Differential Evolution belongs to the more general class called Evolutionary Algorithm. Also the underlying problems tackled by these to-be-configured algorithms include combinatorial optimization as well as the global optimization problem. We also describe the conditions under which experiments were carried out.

3.1 Case studies

$\mathcal{MAX}$–$\mathcal{MIN}$ Ant System - Traveling Salesman Problem (MMASTSP). In $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [22], as in other ant colony optimization [23] algorithm, the relative importance of pheromone information α vs. heuristic information β, the pheromone evaporation rate ρ, and the number of ants m, are parameters that influence algorithm performance. Moreover,

Table 1. Range and default value of each parameter considered for configuring MMASTSP with 2, 4, and 6 parameters.

MMASTSP-2			MMASTSP-4			MMASTSP-6		
param.	range	def.	param.	range	def.	param.	range	def.
α	$[0.0, 5.0]$	1.0	ρ	$[0.0, 1.00]$	0.5	γ	$[0.01, 5.00]$	2.0
β	$[0.0, 10.0]$	2.0	m	$[1, 1200]$	25	nn	$[5, 100]$	25

Table 2. Range and default value of each parameter considered for configuring DE with 3 parameters (left) and PSO with 2 and 5 parameters (right).

DE-3		
param.	range	def.
N	$[4, 1000]$	1000
xo	$[0.0, 1.0]$	0.9
s	$[0.0, 1.0]$	0.8

PSO-2			PSO-5		
param.	range	def.	param.	range	def.
χ	$[0.0, 1.0]$	0.729	N	$[4, 1000]$	30
ϕ_1	$[0.0, 4.0]$	2.05	p	$[0.0, 1.0]$	1
			ϕ_2	$[0.0, 4.0]$	2.05

in $\mathcal{MM}$AS, the ratio γ between the maximum and minimum pheromone trail values is also of importance. Here we tackle the well-known traveling salesman problem (TSP). For MMASTSP, an extra parameter nn, which gives the number of nearest neighbors considered in the solution construction, is also to be set. No local search is applied. We extract three configuration problems from MMASTSP with 2, 4, and 6 parameters, namely MMASTSP-2 (with α and β), MMASTSP-4 (plus m and ρ), and MMASTSP-6 (plus γ and nn). This is done by fixing the unused parameters to their default values (see Table 1). For the default values we follow the instruction of the ACOTSP software [24].

We used the DIMACS instance generator [25] to create Euclidean TSP instances with 750 nodes, where the nodes are uniformly distributed in a square of side length 10 000. 1000 such instances are generated for the configuration process, and 300 for the testing process.

Differential Evolution - Rastrigin Function. Differential Evolution (DE) [26] is a population-based, stochastic continuous optimization method that generates new candidate solutions by mutating and recombining existing ones. In the so-called *DE/rand/S/bin* variant (the most common version is the one with $S = 1$) [27], there are $S + 2$ parameters to set: the population size N, the crossover probability xo, and the value of the scaling factor s. In this variant, the population size must be at least equal to $2S + 2$. We refer the reader to the left columns of Table 2 for the parameter ranges and the default values used in our experiments, and to [26, 27] for more information about DE and its parameters. As the default parameter value of DE, we adopted from [28].

The experiments are carried out on problems derived from the Rastrigin function, each of which has different fitness distance correlation (FDC) [29]. The Rastrigin function, whose n-dimensional formulation is $nA + \sum_{i=1}^{n} (x_i^2 - A\cos(\omega x_i))$, can be thought of as a parabola with a superimposed sinusoidal wave with an amplitude and frequency controlled by parameters A and ω respectively. By changing the values of A and ω one can obtain a family of problems. In our experiments, we set $\omega = 2\pi$, and set the amplitude A to obtain functions with FDC normally distributed in the range $(0.4, 1.0)$. The computation of the FDC was estimated using 10^4 uniformly distributed random samples over the search range. The mean FDC of the family of functions used in our experiments is 0.7 with a standard deviation equal to 0.1. In the space of amplitudes, these values correspond to a mean amplitude equal to 10.60171 and a standard deviation equal to 2.75. Other settings are the search range and the dimensionality of the problem, which we set to $[-5.12, 5.12]^n$ and $n = 100$, respectively.

Particle Swarm Optimization - Rastrigin Function. We use a particle swarm optimization (PSO) [30] algorithm with 2 and 5 parameters. In the first case, the two parameters are the so-called constriction factor χ and a value assigned to both acceleration coefficients ϕ_1. In the second case, the free parameters are the population size N, the constriction factor χ, the value of each acceleration coefficient, ϕ_1 and ϕ_2, and a probability of connection between any pair of particles in the population topology p. For more information about PSO and its parameters, we refer the reader to [31–33]. More details about the parameters used in our experiments are listed in the right columns of Table 2. For the default value of the parameters in PSO we follow [32]. We use the same family of Rastrigin functions that we used with DE. The same as in the DE experiments, 1000 instances are generated for the configuration process and 1000 for the testing process.

4 Experimental Results

4.1 Experimental setup

For each sampling algorithm, four levels of nr, i.e. the number of times that each evaluation is repeated, are considered: $5, 10, 20, 40$. We also consider for each configuration problem four different *configuration budgets*. The term *configuration budget* is determined as the maximum number of times that the algorithm to be configured can be applied during the configuration process. Let d be the dimensionality of the configuration problem, i.e. the number of parameters to be determined, the first and minimum level of the configuration budget is chosen to be $B_1 = 40 \cdot (2d + 2)$, e.g. $B_1 = 240$ when $d = 2$. The setting of B_1 is chosen in such a way since BOBYQA needs at least $2n + 1$ points to make the first quadratic interpolation, and this setting guarantees BOBYQA with $nr = 40$ can make at least one quadratic interpolation guess. The rest of the three levels of configuration budget doubles the previous level respectively, that is $B_i = 2^{i-1} \cdot B_1, i = 2, 3, 4$.

4.2 Detailed settings

In each of the six benchmark configuration problems, 10 `trials` were run. Each `trial` is the execution of the `configuration process` together with a subsequent `testing process`. In the testing process, the final parameter setting returned by the configuration process is evaluated on the set of test instances. For the purpose of reducing experimental variance, in each trial of the configuration process, we used a fixed random order of the tuning instances, and each instance will be evaluated with a common random seed.

In our experiments, all parameters are tuned with 2 significant digits. This was done by rounding each sampled values. The choice of this significant digit is made due to our observation during the experiments that the lower the number of significant digits, the higher the performance of the tuned parameters. In each trial, the historical evaluation results are stored in an archive list, so that if the same algorithm configuration is sampled twice, the results on the evaluated instances will be read from the archive list without re-evaluating.

For sampling algorithms CMAES, MADS and BOBYQA, a restart mechanism is triggered whenever stagnation behavior is detected. Stagnation can be observed when the search coarseness of the mesh or the radius of the trust region drop to a very low level, for example, less than the degree of the significant digit. Each restart best solution is stored, and in the post-execution phase the best across all restart best solutions is selected by `F-Race`. The configuration budget reserved for the post-execution `F-Race` is determined by a factor μ_{post} times the number of restart best solutions. The factor μ_{post} in the repeated evaluation experiments is determined by $\mu_{post} = max\{5, (20 - nr)\}$, where nr is the number of repeated evaluations. Also in the post-execution `F-Race`, we start the Friedman test for discarding candidates from the $min\{10, nr\}$-th instance, instead of five as in the normal `F-Race` setting to make the selection more conservative.

For the setting of `MADS/F-Race` we follow [34] by allowing the budget for each `F-Race` 10 times the number of candidate configurations. For the setting of `CMAES/F-Race`, since CMAES uses a (μ, λ)-ES, i.e. μ elite configurations are used to generate λ new candidate configurations of the next iteration, the iteration best configurations are stored, and the best configuration will be selected in a post-execution from the set of iteration best configurations by `F-Race`. The post-execution `F-Race` is performed in the same way described above as for restart best configurations, except that we set $\mu_{post} = 10$. For the experiments, we further modified CMAES by considering an initial uniform random sampling of the configuration space: we sample first λ random solutions and then CMAES started using its default parameter settings from the best of these solutions. This modification result in significant improvements for several case studies, especially when the number of sampled points is small.

4.3 Comparisons of algorithms with repeated evaluations

First, we study the nr value for all sampling algorithms. For each algorithm, we compared the four levels $nr = 5, 10, 20, 40$ across six benchmark configuration problems using the pairwise Wilcoxon signed rank test with blocking on each instance and Holm's adjustment for multiple test correction. The statistical results unanimously show that for each sampling algorithm, the smaller the value of nr, the better performance is obtained. In fact, the statistical test show that a setting $nr = 5$ is significantly better than all others. Furthermore, all pairwise comparisons show statistically significant differences. than 20, and Based on this study, in the following we will compare the performance of all sampling algorithms with $nr = 5$.

Given that a setting of $nr = 5$ resulted in best performance, we directly compared the algorithms using this setting to the variants that were using F-race for the selection of the best candidates. Pursuing the same analysis, we found that for two algorithms, iterated sampling and random sampling, the versions with F-race perform substantially better than the fixed sample size of $nr = 5$, while on CMAES and MADS the setting $nr = 5$ perform slightly, but statistically significantly better than the F-race variants. Note that for MADS/F-race the observation here contradicts the conclusions in [34]; a reason may be that here a restart version of MADS is used while in [34] restarts are not considered.

Given this overall result for the comparison between the variants with F-race and $nr = 5$ and the fact that BOBYQA is applied only with a fixed sample size, in the following analysis we focus on the case of using $nr = 5$ only.

4.4 Comparisons of continuous optimization algorithms

Here, we compare the performance of the five continuous optimization algorithms for the setting of $nr = 5$. The Figure 4.4 shows the average costs of each of the five continuous optimization algorithms across four different configuration budget. Each of the six plots shows the results for one configuration problem.

The conclusions we obtain from this analysis is that random sampling is clearly the worst algorithm in almost all configuration problems. (The same still holds if random sampling is combined with F-race.) For very small dimensional configuration problems with two and three parameters to be tuned, BOBYQA is the best performing algorithm. This is the case for the case studies MMASTSP2, PSO2, and DE3. However, BOBYQA's performance degrades very strongly for increasing dimensionality and for PSO5 it is even worse performing than the random sampling approach. The performance of CMAES was relatively robust across the various dimensionalities of the configuration problems and across the various budget levels. The average ranking of iterated sampling and MADS is the same (2.83), which is substantially worse than CMAES (2.29).

Most of the differences that can be observed in Figure 4.4 are actually statistically significant. In Table 3 we indicate for each configuration problem and for each level of the configuration budget the ranking of the algorithms (from best to worst). All differences between consecutive pairs of algorithms are statistically significant except those where two algorithms are connected by a line (above or below the identifiers). Finally, Figure 2 indicates the average ranking of the five algorithms, averages across all budget levels, for varying dimensionality of the configuration problems. This figure confirms the observation that BOBYQA is the best for low dimensional tasks while CMAES shows rather robust performance.

4.5 Comparisons of the tuned the default parameters

We finally compared the results that are obtained by the tuned parameter configurations with those by the default parameter configurations. Please refer to Table 1 and 2 for the default parameter settings of the three case studies. The tuned parameter configurations strongly outperform the default configurations in all cases. Throughout the experiments, the default parameter configuration is only comparable with the tuned configuration in PSO2 with the lowest level budget (240). In the case study of MMASTSP, the tuned configurations improve on average over the default configuration by more than 10%. In the PSO case study, the tuned parameters on average improve over the default configuration by more than 30% in the PSO-2, and more than 50% in PSO-5.

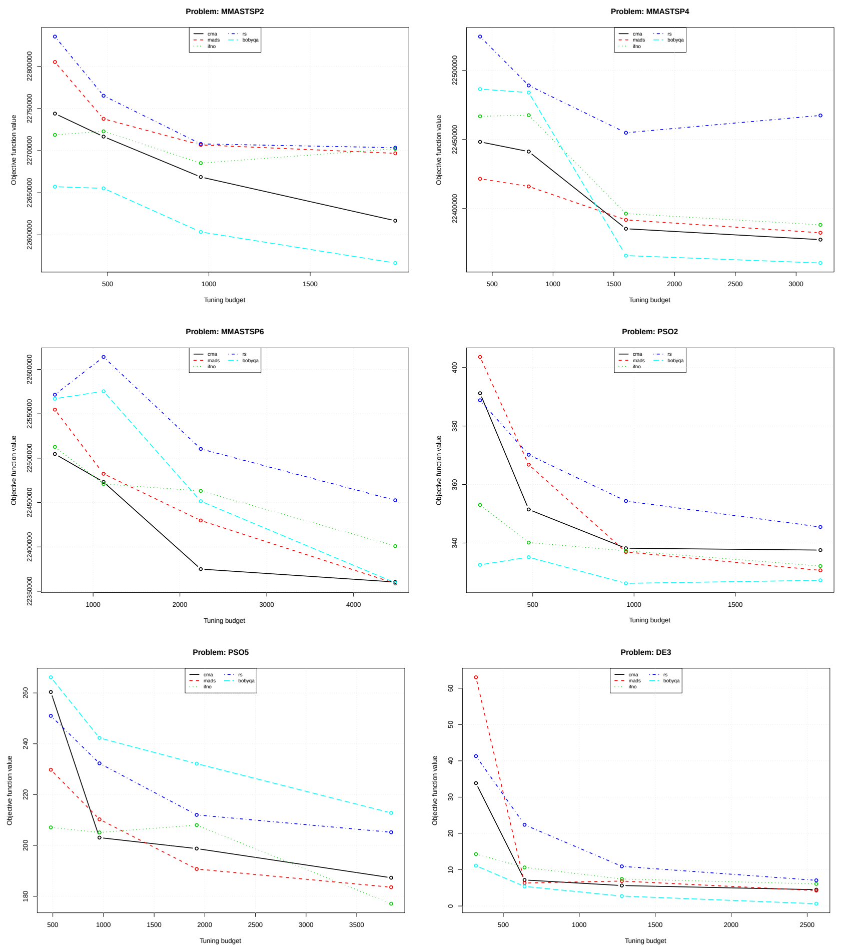


Fig. 1. The comparisons of different algorithms with $nr = 5$ on the six configuration problems

The strongest improvement by tuning is observed in the case study of differential evolution. The average cost obtained by the default configuration is 2168, which is one order of magnitude worse than the costs obtained by the worst tuned configurations using the fewest configuration budget. In fact, almost all tuning algorithms can find configurations that give results close to the optimal value 0. Furthermore, by the default parameter configuration, DE gives worse results than PSO (2168 vs. 589). However, the tuned DE performs much better than the tuned PSO (0.64 vs. 177). On the one side this shows DE is very parameter sensitive, on the other side this also proves that the algorithm configuration procedure can exploit the full potential of the algorithm, and should be applied before algorithm comparisons. These results confirm the practical importance of automated algorithm configuration in deriving high-performing algorithms.

5 Conclusions

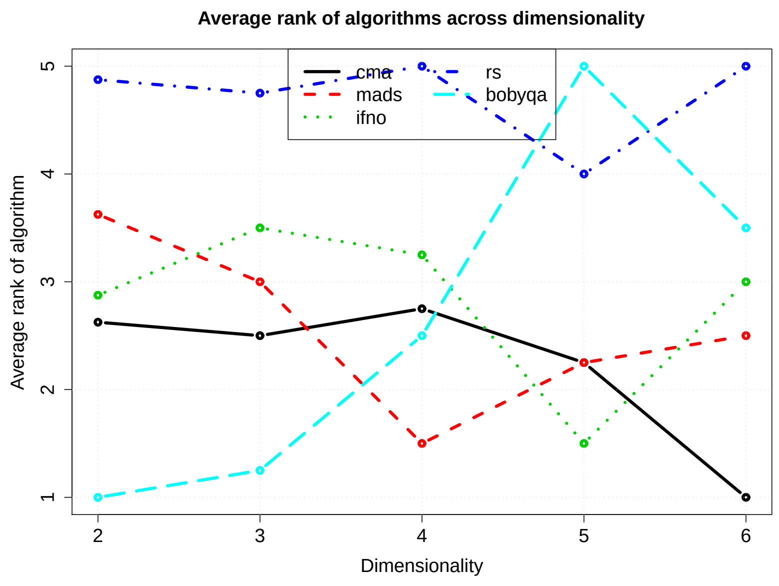
In the article we compared the performance of three modern state-of-the-art continuous optimization algorithms, CMA-ES, BOBYQA and MADS, together with the population-based iterated sampling and random sampling, for the automatic algorithm configuration of the numerical parameters. Two swarm intelligence algorithms, an ant colony optimization algorithm applied to the TSP and a PSO algorithm are considered as case studies, together with a differential evolution algorithm. The sampling algorithms are improved by a restart mechanism, and CMA-ES is hybridized with a uniform random sampling in the first iteration.

The experiments show that, among the five continuous optimization algorithms, BOBYQA performs the best in low dimensional problems (with two or three parameter to be set), but that

Table 3. Algorithms (using $nr = 5$) are ordered according to the ranking (form best to worst) for each configuration problem and for each budget level. The abbreviations used are B for BOBYQA, C for CMAES, I for iterated sampling, M for MADS, and R for random sampling. The budget is from B_1 (low) to B_4 (high). In the ordering, an overline or an underline indicates that between these algorithms no statistically significant differences were observed.

budget	MMAS-2	MMAS-4	MMAS-6	DE-3	PSO-2	PSO-5
B_1	B I C M R	M C I B R	$\overline{\text{C I}}\ \overline{\text{M B R}}$	C B I R M	B I C R M	$\text{I }\overline{\text{C M}}\text{ R B}$
B_2	$\text{B C }\overline{\text{I M}}\text{ R}$	$\text{M }\overline{\text{C }\underline{\text{I}}}\,\underline{\text{B R}}$	$\text{C }\overline{\text{I M}}\text{ B R}$	B M I C R	$\text{B I C }\overline{\text{M R}}$	I C M R B
B_3	$\text{B C I }\overline{\text{M R}}$	$\text{B }\overline{\text{M C I}}\text{ R}$	$\overline{\text{C M}}\ \overline{\text{B I}}\text{ R}$	B C M I R	$\text{B }\overline{\text{M }\underline{\text{I}}}\,\underline{\text{C}}\text{ R}$	M C I R B
B_4	$\text{B C }\overline{\text{M I R}}$	$\text{B }\overline{\text{M I C}}\text{ R}$	$\overline{\text{C M B}}\text{ I R}$	B M C I R	$\text{B }\overline{\text{C }\underline{\text{M}}}\,\underline{\text{I}}\text{ R}$	$\text{I }\overline{\text{M C}}\text{ R B}$

Fig. 2. The average rank of the sampling algorithms across dimensionalities of the configuration problems.



it performs poorly on the case studies with more parameters to be set. CMA-ES appears to be a rather robust algorithm across all dimensionalities. In future work, we want to integrate continuous optimization algorithms with other configuration methods that work on categorical parameters. In fact, considering a hybrid solver, where iteratively continuous configuration tasks arise that are then tackled by effective algorithms specialized to such problems, may be an interesting way to go to improve the performance of automated configuration algorithms also on more complex configuration tasks.

Acknowledgments. We thank Michael J. D. Powell for providing the Fortran code of BOBYQA. This work has been supported by META-X, an ARC project funded by the French Community of Belgium. The authors acknowledge support from the fund for scientific research F.R.S.-FNRS of the French Community of Belgium, of which they are research associates (MB and TS), or an aspirant (ZY), respectively. This work was supported by the META-X project, an *Action de Recherche Concertée* funded by the Scientific Research Directorate of the French Community of Belgium. Zhi Yuan, Mauro Birattari, and Thomas Stützle acknowledge support from the Belgian F.R.S.-FNRS.

References

1. Adenso-Díaz, B., Laguna, M.: Fine-Tuning of Algorithms using Fractional Experimental Designs and Local Search. Operations Research **54**(1) (2006) 99–114

2. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An Automatic Algorithm Configuration Framework. Journal of Artificial Intelligence Research **36** (2009) 267–306
3. Mercer, R.E., Sampson, J.R.: Adaptive search using a reproductive meta-plan. Kybernetes **7**(3) (1978) 215–228
4. Greffenstette, J.F.: Optimization of control parameters for genetic algorithms. IEEE Transactions on Systems, Man, and Cybernetics **16**(1) (1986) 122–128
5. Nannen, V., Eiben, A.E.: Relevance Estimation and Value Calibration of Evolutionary Algorithm Parameters. In: Proceedings of IJCAI 2007. AAAI Press, Menlo Park, CA (2007) 975–980
6. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation–The New Experimentalism. Springer, Berlin, Germany (2006)
7. Hutter, F., Hoos, H.H., Leyton-Brown, K., Murphy, K.P.: An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Proceedings of GECCO 2009. ACM press, New York, NY, USA (2009) 271–278
8. Birattari, M.: The Problem of Tuning Metaheuristics as seen from a Machine Learning Perspective. PhD thesis, Université Libre de Bruxelles (2004)
9. Balaprakash, P., Birattari, M., Stützle, T.: Improvement Strategies for the F-Race algorithm: Sampling Design and Iterative Refinement. In Bartz-Beielstein, T., et al., eds.: Proceedings of HM 2007. Volume 4771 of LNCS. Springer, Heidelberg, Germany (2007) 108–122
10. Birattari, M.: Tuning Metaheuristics: A machine learning perspective. Springer, Berlin, Germany (2009)
11. Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T.: F-Race and iterated F-Race: An overview. In Bartz-Beielstein, T., et al., eds.: Experimental Methods for the Analysis of Optimization Algorithms. Springer, Berlin, Germany (2009) 311–336
12. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. Evolutionary Computation **13**(3) (2005) 387–410
13. Fukunaga, A.S.: Automated Discovery of Local Search Heuristics for Satisfiability Testing. Evolutionary Computation **16**(1) (2008) 31–61
14. Ansotegui Gil, C., Sellmann, M., Tierney, K.: A Gender-Based Genetic Algorithm for the Automatic Configuration of Solvers. In: Proceedings of CP 2009. Volume 5732 of LNCS. Springer, Heidelberg, Germany (2009) 142–157
15. Powell, M.J.D.: The BOBYQA algorithm for bound constrained optimization without derivatives. Technical Report NA2009/06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge (2009)
16. Powell, M.J.D.: The NEWUOA software for unconstrained optimization. In: Large-Scale Nonlinear Optimization. Volume 83 of Nonconvex Optimization and Its Applications. Springer-Verlag, Berlin, Germany (2006) 255–297
17. More, J., Wild, S.: Benchmarking derivative-free optimization algorithms. SIAM Journal on Optimization **20**(1) (2009) 172–191
18. Auger, A., Hansen, N., Zerpa, J.M.P., Ros, R., Schoenauer, M.: Experimental Comparisons of Derivative Free Optimization Algorithms. In: Proceedings of SEA 2009. Volume 5526 of LNCS. Springer, Heidelberg, Germany (2009) 3–15
19. Audet, C., J. E. Dennis, J.: Mesh Adaptive Direct Search Algorithms for Constrained Optimization. SIAM Journal on Optimization **17**(1) (2006) 188–217
20. Torczon, V.: On the convergence of pattern search algorithms. SIAM Journal on Optimization **7**(1) (1997) 1–25
21. Hansen, N., Ostermeier, A.: Completely derandomized self-adaptation in evolution strategies. Evolutionary Computation **9**(2) (2001) 159–195
22. Stützle, T., Hoos, H.: $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Future Generation Computer Systems **16**(8) (2000) 889–914
23. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge, MA, USA (2004)
24. Stützle, T.: Software ACOTSP. `http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html` (webpage last visited in March 2010)
25. Johnson, D.S., McGeoch, L.A., Rego, C., Glover, F.: 8th DIMACS implementation challenge. `http://www.research.att.com/~dsj/chtsp/` (webpage last visited in April 2009)
26. Storn, R., Price, K.: Differential evolution a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization **11**(4) (1997) 341–359
27. Ting, C.K., Huang, C.H.: Varying Number of Difference Vectors in Differential Evolution. In: Proceedings of CEC 2009. IEEE Press, Piscataway, NJ, USA (2009) 1351–1358
28. Storn, R.: Differential evolution homepage. `http://www.icsi.berkeley.edu/~storn/code.html#prac` (webpage last visited in March 2010)
29. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proceedings of the 6th International Conference on Genetic Algorithms, San Francisco, CA, Morgan Kaufmann (1995) 184–192

30. Kennedy, J., Eberhart, R.: Particle Swarm Optimization. In: Proceedings of IEEE International Conference on Neural Networks, IEEE Press, Piscataway, NJ, USA (1995) 1942–1948
31. Clerc, M.: Particle Swarm Optimization. ISTE, London, UK (2006)
32. Poli, R., Kennedy, J., Blackwell, T.: Particle Swarm Optimization. An Overview. Swarm Intelligence **1**(1) (2007) 33–57
33. Dorigo, M., Montes de Oca, M.A., Engelbrecht, A.P.: Particle Swarm Optimization. Scholarpedia **3**(11) (2008) 1486
34. Yuan, Z., Stützle, T., Birattari, M.: MADS/F-Race: mesh adaptive direct search meets F-Race. Technical Report TR/IRIDIA/2010-001, IRIDIA (January 2010)