Modern Continuous Optimization Algorithms for Tuning Real and Integer Algorithm Parameters

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

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

Abstract. To obtain peak performance from optimization algorithms, it is required to set appropriately their parameters. Frequently, algorithm parameters can take values from the set of real numbers, or from a large integer set. To tune this kind of parameters, it is interesting to apply state-of-the-art continuous optimization algorithms instead of using a tedious, and error-prone, hands-on approach. In this paper, we study the performance of several continuous optimization algorithms for the algorithm parameter tuning task. As case studies, we use a number of optimization algorithms 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 [6] 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 the parameter tuning problem as an optimization problem, where the goal is to find 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 finding a solution of a specific quality level. A number of algorithms have been proposed for this task over the years. Several of these efforts focused on finding good values to numerical parameters, which resulted in methods such as CALIBRA [1], REVAC [16], SPO [5] and SPO$^+$ [12]. 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 [7,8], ParamILS [13], genetic programming [17,10], and gender-based genetic algorithms [2].

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 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 parameter tuning tasks that involve continuous parameters, such as the pheromone evaporation rate in an ant colony optimization (ACO) algorithm, and integer parameters, such as the population size in evolutionary algorithms. 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 scale of the tuning tasks considered in this paper is small. We tackle problems with two to six numerical parameters. The reasons for limiting the number of parameters are the following. First, assuming that the parameter tuning 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 several parameter tuning tasks that involve continuous or quasi-continuous parameters only. For example, if an algorithm is going to be used for tackling a new class of problem instances, or on a new application situation, it would be better to first tune the algorithm's continuous or quasi-continuous parameters before proceeding with the re-design of the algorithm.

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

2 Tuning Algorithms

For the parameter tuning task, we selected state-of-the-art black-box continuous optimization algorithms developed in the mathematical programming and in the evolutionary computation communities. Since these algorithms were not explicitly designed for tackling stochastic problems, we enhanced them in order to let them handle noise. As a baseline for comparison, we included uniform random and iterated random sampling in our experiments.

2.1 Basic Algorithms

Uniform Random and Iterated Random Sampling (URS & IRS). The results obtained with URS and IRS are used as a baseline for evaluation. URS explores the space of possible parameter settings uniformly at random. The obvious drawback of this approach is its inability to focus the search on promising regions of the search space. The method we refer to as IRS is the sampling part of Iterated F-Race as described in [8]. In IRS, the steps of solution generation, selection and refinement are executed 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 search around the best-so-far solutions.

Bound Optimization by Quadratic Approximation (BOBYQA). BOBYQA [20] is a derivative-free optimization algorithm based on the trust region paradigm. It is an extension of the NEWUOA [19] 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 $m = 2d + 1$ [20], 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 [4]. The initial and final trust region radii, as well as a maximum number of function evaluations before termination are its parameters.

Mesh Adaptive Direct Search (MADS). In MADS [3], a number of trial points lying on a *mesh* are generated and evaluated around the best-so-far solution at every iteration. If a new better solution is found, the next iteration begins with a possibly 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 [26] in that MADS allows a more flexible exploration of this refined mesh by allowing the sampling 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 [11], 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 [4].

2.2 Enhancing Noise Tolerance

The problem of tuning the parameters of an optimization 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 instances. 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 by nr the number of times the objective function evaluation is repeated. The advantages of this approach are its simplicity and the confidence that can be associated with the estimate as a function of nr. We tried this approach with all methods using different values for nr. 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 solution remains, or the maximum number of evaluations is reached. The elimination mechanism of F-Race is independent of the composition of the initial set of candidate solutions. 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 URS, 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. More information about F-Race can be found in [7,8].

3 Benchmark Tuning Problems

We compare the performance of the continuous optimization algorithms described in Section 2 on six benchmark (parameter) *tuning problems*. Each *tuning problem* consists of a parameterized algorithm to be tuned, and an optimization problem to which this algorithm is applied. The six tuning problems are originated from three classes of case studies with three underlying algorithms to be tuned. Two of them are swarm intelligence algorithms, ACO and particle swarm optimization (PSO), while the other differential evolution (DE) is an evolutionary algorithm. ACO is used to tackle a combinatorial optimization problem. PSO and DE are used to tackle a continuous optimization problem. The three case studies are the following.

$\mathcal{MAX}$–$\mathcal{MIN}$. Ant System - Traveling Salesman Problem (MMASTSP). $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) [24] is one of the most successful ACO [9] algorithms and we consider here the following parameters. The weight of the pheromone information α and the heuristic information β, the pheromone evaporation rate ρ, and the number of ants m. Moreover, 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 is nn, which gives the

Table 1. Range and default value of each parameter considered for tuning 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 tuning 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

number of nearest neighbors considered in the solution construction. No local search is applied. We extract three tuning 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 ACOTSP software [23].

We used the DIMACS instance generator [14] 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 tuning process, and 300 for the testing process.

Differential Evolution - Rastrigin Function. DE [22] is a population-based, stochastic continuous optimization method that generates new candidate solutions by exploiting the spatial distribution of existing ones. In the so-called *DE/rand/S/bin* variant (the most common version is the one with $S = 1$) [25], 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 [22,25] for more information about DE and its parameters. The default parameter settings for DE were those suggested in [21].

The experiments are carried out on problems derived from the Rastrigin function, each of which has different fitness distance correlation (FDC) [15]. 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 whole family of problems. In our experiments, we set $\omega = 2\pi$, and we vary the amplitude A to obtain functions with different FDCs. This was done by sampling the values of A from a normal distribution with mean equal to 10.60171 and standard deviation equal to 2.75. These values approximately map to a normally distributed FDC with mean equal to 0.7 and standard deviation equal to 0.1. The FDC was estimated using 10^4 uniformly distributed random samples over the search range. Other settings are the search range and the dimensionality, n, of the problem, which we set to $[-5.12, 5.12]^n$ and $n = 100$, respectively.

Particle Swarm Optimization - Rastrigin Function. We use a PSO [18] algorithm with two and five 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 p of connecting any two particles in the population topology. For more information about PSO and its parameters, we refer the reader to [18]. More details about the parameters used in our experiments are listed in the right two columns of Table 2. For the default value of the parameters in PSO we follow [18]. We tackle the same family of Rastrigin functions as in the DE experiments, and generate 1000 instances for the tuning process and 1000 for the testing process.

4 Experiments

4.1 Experimental Setup

For each of the six benchmark tuning problems, 10 `trials` were run. Each `trial` is the execution of the `tuning process` together with a subsequent `testing process`. In the testing process, the final parameter setting returned by the tuning process is evaluated on a set of test instances. For the purpose of reducing experimental variance, in each trial of the tuning process, we use a fixed random order of the tuning instances, and each instance is evaluated with a common random seed. The comparison of sampling algorithms is done across six benchmark tuning problems using the pairwise Wilcoxon signed rank test with blocking on each instance and Holm's adjustment for multiple test correction.

For each sampling algorithm, four levels of nr, that is, the number of times the optimization algorithm being tuned is executed with the same parameter settings, are considered: $5, 10, 20, 40$. We also consider for each tuning problem four different *tuning budgets*, that is, the maximum number of times that the target algorithm can be run during the tuning process. Let d be the dimensionality of the tuning problem, that is, the number of parameters to be tuned. The first and minimum level of the tuning 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 this way since BOBYQA needs at least $2n + 1$ points to make the first quadratic interpolation, and this setting guarantees that BOBYQA with $nr = 40$ can make at least one quadratic interpolation guess. The rest of the three levels of tuning budgets double the previous level respectively, that is, $B_i = 2^{i-1} \cdot B_1, i = 2, 3, 4$.

In our experiments, all parameters are tuned with a 2 significant digits precision. This was done by rounding each sampled value. This choice was made because we observed during experimentation 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.

4.2 Settings of the Sampling Algorithms

In our experiments, the sampling algorithms, which generate the trial configurations, that is, algorithms such as CMA-ES, MADS and BOBYQA, are extended by a restart mechanism that is triggered whenever stagnation behavior is detected. Stagnation can be detected when the search coarseness of the mesh or the trust region radius 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 tuning 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 $max\{10, nr\}$-th instance, instead of five as in the normal `F-Race` setting to make the selection more conservative. Based on our experiments, the hybrid with `F-Race` is indeed better performing than its counterpart, and the advantage becomes statistically significant in the case of low nr value ($nr = 5$).

For the execution of each sampling algorithm, the default settings are adopted in our experiments, except CMA-ES. Due to the small number of sampled points, we modify CMA-ES by applying a uniform random sampling in the first iteration, where the best point will serve as a starting point for CMA-ES. This modification results in significant improvements for most of the case studies, especially when the number of sampled points is small.

We first study the setting of nr for all sampling algorithms. The statistical results unanimously show that for each sampling algorithm, the smaller the value of nr, the better performance is obtained, that is, $nr = 5$ is the best setting. Furthermore, all pairwise comparisons show statistically significant differences.

We consider also use of `F-Race` during the execution of the sampling algorithms instead of the fixed number of repeated evaluations. For IRS and URS, we adopted the iterated `F-Race` and random sampling `F-Race` from [8]. For `MADS/F-Race` we have adopted the algorithm of [27] by allowing the budget for each `F-Race` to be 10 times the number of candidate configurations. CMA-ES uses a (μ, λ)-ES, that is, μ elite configurations are used to generate λ new candidate configurations of the next iteration. Therefore, for `CMAES/F-Race` 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 $\mu_{post} = 10$.

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, IRS and URS, the versions with F-Race perform substantially better than the fixed sample size of $nr = 5$, while on CMA-ES 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 [27]; a reason may be that here a restart version of MADS is used, while in [27] 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 $nr = 5$ only.

4.3 Comparisons of Continuous Optimization Algorithms

Here, we compare the performance of the five continuous optimization algorithms for the setting of $nr = 5$. Figure 1 shows the average costs of each of the five continuous optimization algorithms across four different tuning budgets. Each of the six plots shows the results for one tuning problem.

The main conclusions we obtain from this analysis is the following. For very small dimensional tuning 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 rapidly as the problem dimensionality increases. In PSO5, for example, it even performs worse than URS. The performance of CMA-ES is relatively robust across the various dimensionalities of the tuning problems and across the budget levels tested. The average ranking of CMA-ES among five algorithms is 2.29, substantially better than IRS and MADS (tied as 2.83). Finally, all the continuous optimization algorithms tested clearly outperformed URS in almost all tuning problems. (The same also holds if URS is combined with F-Race.)

Most of the differences that can be observed in Figure 1 are actually statistically significant. In Table 3 we indicate for each tuning problem and for each level of the tuning 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 shows the average ranking of the five algorithms grouped by the dimensionality of the tuning problem averaged across all budget levels. This figure confirms the observation that BOBYQA is the best for low dimensional tasks while CMA-ES shows rather robust performance.

4.4 Comparison between the Tuned and Default Configurations

We finally compared the results obtained with the tuned parameter configurations with those obtained with the default parameter configurations. Please refer

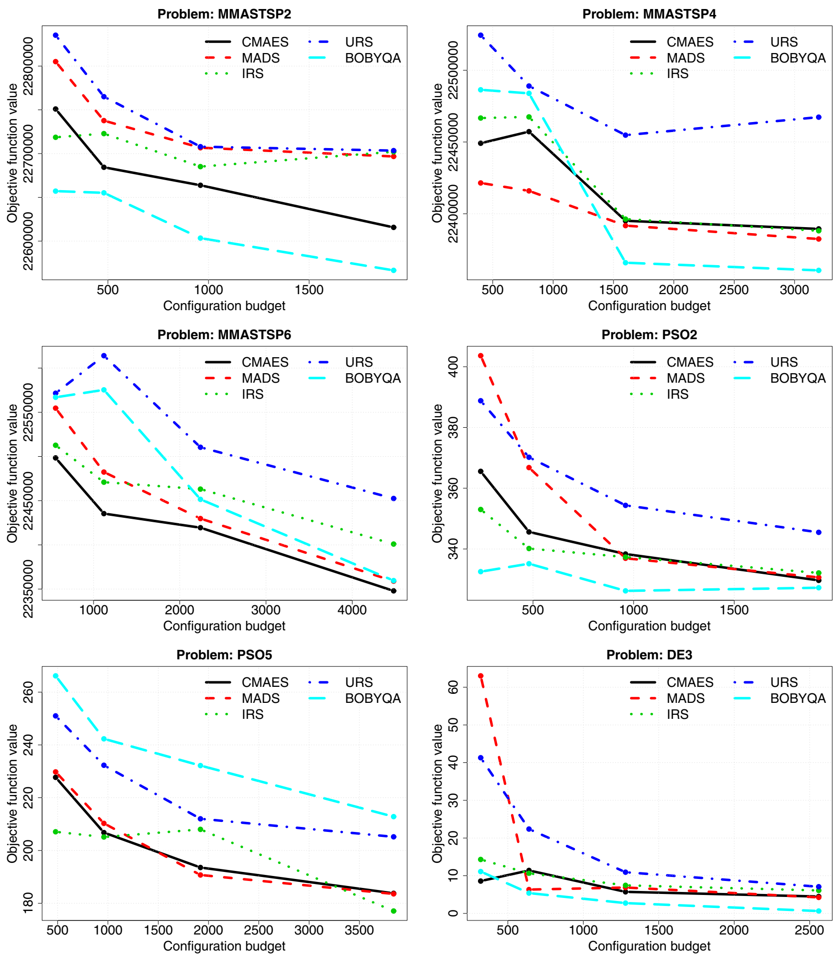

Fig. 1. The comparisons of different algorithms with $nr = 5$ on the six tuning problems

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 improve, on average, over the default configuration by more than 30% in the PSO-2, and by more than 50% in PSO-5. The strongest improvement by tuning is observed in the case study of DE. 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 smallest tuning budget. In fact, almost all tuning algorithms can find configurations

Table 3. Algorithms (using $nr = 5$) are ordered according to the ranking (form best to worst) for each tuning problem and for each budget level. The abbreviations used are B for BOBYQA, C for CMA-ES, I for IRS, M for MADS, and R for URS. 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	$\text{B }\overline{\text{C }\underline{\text{M}}}\underline{\text{ I}}\text{ R}$	$\text{I }\overline{\text{M C}}\text{ R B}$

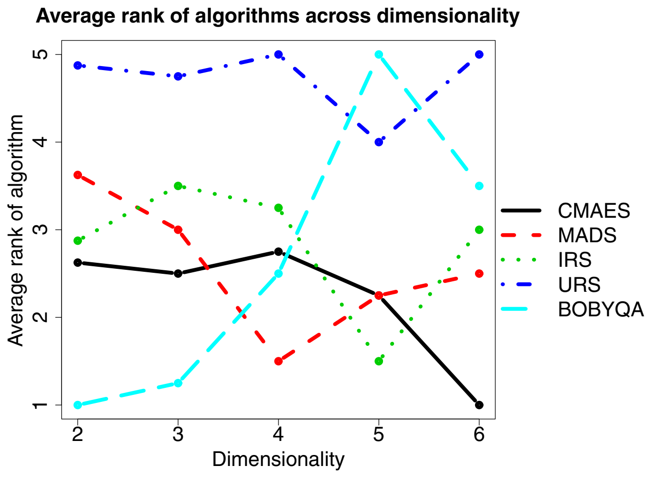

Fig. 2. The average rank of the sampling algorithms across dimensionalities of the tuning problems

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 that DE is very sensitive to its parameter settings. On the other side, this also proves that the algorithm tuning 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 tuning in deriving high-performing algorithms.

5 Conclusions

In this paper, we compare the performance of three modern state-of-the-art continuous optimization algorithms, CMA-ES, BOBYQA and MADS, together with the population-based URS and IRS, for the automatic tuning of numerical parameters. Two swarm intelligence algorithms, an ACO algorithm applied to the TSP and a PSO algorithm are considered as case studies, together with a DE 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 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 configuration methods that work on categorical parameters. In fact, considering a hybrid solver, where iteratively continuous tuning 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 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), 99–114 (2006)
2. Ansotegui Gil, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of solvers. In: Gent, I.P. (ed.) CP 2009. LNCS, vol. 5732, pp. 142–157. Springer, Heidelberg (2009)
3. Audet, C., Dennis, J.E., Mesh, J.: adaptive direct search algorithms for constrained optimization. SIAM Journal on Optimization 17(1), 188–217 (2006)
4. Auger, A., Hansen, N., Zerpa, J.M.P., Ros, R., Schoenauer, M.: Experimental comparisons of derivative free optimization algorithms. In: SEA 2009. LNCS, vol. 5526, pp. 3–15. Springer, Heidelberg (2009)
5. Bartz-Beielstein, T.: Experimental Research in Evolutionary Computation–The New Experimentalism. Springer, Berlin (2006)
6. Birattari, M.: The Problem of Tuning Metaheuristics as seen from a Machine Learning Perspective. Ph.D. thesis, Université Libre de Bruxelles (2004)
7. Birattari, M.: Tuning Metaheuristics: A machine learning perspective. Springer, Berlin (2009)
8. 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, pp. 311–336. Springer, Berlin (2009)
9. Dorigo, M., Stützle, T.: Ant Colony Optimization. MIT Press, Cambridge (2004)
10. Fukunaga, A.S.: Automated discovery of local search heuristics for satisfiability testing. Evolutionary Computation 16(1), 31–61 (2008)

11. Hansen, N.: The CMA evolution strategy: a comparing review. In: Lozano, J., et al. (eds.) Towards a new evolutionary computation, pp. 75–102. Springer, Berlin (2006)
12. Hutter, F., Hoos, H.H., Leyton-Brown, K., Murphy, K.P.: An experimental investigation of model-based parameter optimisation: SPO and beyond. In: Proc. of GECCO 2009, pp. 271–278. ACM press, New York (2009)
13. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
14. Johnson, D.S., McGeoch, L.A., Rego, C., Glover, F.: 8th DIMACS implementation challenge, `http://www.research.att.com/~dsj/chtsp/`
15. Jones, T., Forrest, S.: Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In: Proc. of 6th Int. Conf. on Genetic Algorithms, pp. 184–192. Morgan Kaufmann, San Francisco (1995)
16. Nannen, V., Eiben, A.E.: Relevance estimation and value calibration of evolutionary algorithm parameters. In: Proc. of IJCAI 2007, pp. 975–980 (2007)
17. Oltean, M.: Evolving evolutionary algorithms using linear genetic programming. Evolutionary Computation 13(3), 387–410 (2005)
18. Poli, R., Kennedy, J., Blackwell, T.: Particle swarm optimization. An overview. Swarm Intelligence 1(1), 33–57 (2007)
19. Powell, M.J.D.: The NEWUOA software for unconstrained optimization. In: Large-Scale Nonlinear Optimization, Nonconvex Optimization and Its Applications, vol. 83, pp. 255–297. Springer, Berlin (2006)
20. Powell, M.J.D.: The BOBYQA algorithm for bound constrained optimization without derivatives. Tech. Rep. NA2009/06, Department of Applied Mathematics and Theoretical Physics, University of Cambridge (2009)
21. Storn, R.: Differential evolution homepage, `http://www.icsi.berkeley.edu/~storn/code.html#prac`
22. Storn, R., Price, K.: Differential evolution – a simple and efficient heuristic for global optimization over continuous spaces. Journal of Global Optimization 11(4), 341–359 (1997)
23. Stützle, T.: Software ACOTSP, `http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html`
24. Stützle, T., Hoos, H.: $\mathcal{MAX}$–$\mathcal{MIN}$. Ant System. Future Generation Computer Systems 16(8), 889–914 (2000)
25. Ting, C.K., Huang, C.H.: Varying number of difference vectors in differential evolution. In: Proc. of CEC 2009, pp. 1351–1358. IEEE Press, Piscataway (2009)
26. Torczon, V.: On the convergence of pattern search algorithms. SIAM Journal on Optimization 7(1), 1–25 (1997)
27. Yuan, Z., Stützle, T., Birattari, M.: MADS/F-race: mesh adaptive direct search meets F-race. In: Ali, M., et al. (eds.) Trends in Applied Intelligent Systems. LNCS, vol. 6096, pp. 41–50. Springer, Heidelberg (2010)