Swarm Intell (2012) 6:49–75
DOI 10.1007/s11721-011-0065-9

Continuous optimization algorithms for tuning real and integer parameters of swarm intelligence algorithms

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

Received: 11 November 2010 / Accepted: 4 November 2011 / Published online: 15 December 2011

Abstract The performance of optimization algorithms, including those based on swarm intelligence, depends on the values assigned to their parameters. To obtain high performance, these parameters must be fine-tuned. Since many parameters can take real values or integer values from a large domain, it is often possible to treat the tuning problem as a continuous optimization problem. In this article, we study the performance of a number of prominent continuous optimization algorithms for parameter tuning using various case studies from the swarm intelligence literature. The continuous optimization algorithms that we study are enhanced to handle the stochastic nature of the tuning problem. In particular, we introduce a new post-selection mechanism that uses F-Race in the final phase of the tuning process to select the best among elite parameter configurations. We also examine the parameter space of the swarm intelligence algorithms that we consider in our study, and we show that by fine-tuning their parameters one can obtain substantial improvements over default configurations.

Keywords Automated algorithm configuration · Parameter tuning · Continuous optimization algorithm · Swarm intelligence · F-Race

Z. Yuan (✉) · M.A. Montes de Oca · M. Birattari · T. Stützle
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: zyuan@ulb.ac.be

M.A. Montes de Oca
e-mail: mmontes@ulb.ac.be

M. Birattari
e-mail: mbiro@ulb.ac.be

T. Stützle
e-mail: stuetzle@ulb.ac.be

Present address:
M.A. Montes de Oca
Dept. of Mathematical Sciences, University of Delaware, Newark, DE, USA
e-mail: mmontes@math.udel.edu

1 Introduction

Swarm intelligence algorithms such as ant colony optimization (ACO) (Dorigo et al. 1996, 2006; Dorigo and Stützle 2004; Dorigo 2007) and particle swarm optimization (PSO) (Kennedy and Eberhart 1995; Kennedy et al. 2001; Clerc 2006) are well-established optimization techniques. Their performance, either measured by the quality of the solution obtained in a fixed computation time, or by the computation time needed to find a solution of a certain desired quality, strongly depends on the values assigned to their parameters. Finding parameter values that optimize algorithm performance is itself a difficult optimization problem (Birattari et al. 2002; Birattari 2009) that has to be tackled with an appropriate automated tuning procedure.

Significant attention is currently being devoted to the offline tuning of algorithms (Birattari 2009). Offline tuning concerns the determination of optimal parameter values before the algorithm is deployed. Various offline tuning methods have recently been proposed. Some of them, such as CALIBRA (Adenso-Díaz and Laguna 2006), SPO (Bartz-Beielstein 2006) and SPO$^+$ (Hutter et al. 2009a), have been designed for tuning numerical parameters only. Others, such as F-Race (Birattari et al. 2002), iterated F-Race (Birattari et al. 2010), ParamILS (Hutter et al. 2009b), genetic programming (Oltean 2005; Fukunaga 2008), REVAC (Nannen and Eiben 2007), and gender-based genetic algorithms (Ansótegui et al. 2009), are able to also tune categorical parameters including, for example, the choice among different constructive approaches to be used in ACO, or the choice of crossover operators in evolutionary algorithms. The aforementioned methods have produced good results and, for many optimization algorithms, they have found parameter values that improve over the parameter values originally proposed by the designers of the algorithms.

For tuning numerical parameters, a promising alternative is to treat the offline tuning problem as a stochastic continuous optimization problem and to apply continuous optimization techniques to it (Yuan et al. 2010a). In this paper, we follow this direction. In particular, we use state-of-the-art continuous optimization algorithms and we enhance them with mechanisms for handling the stochasticity of the offline tuning problem. We consider tuning both, real parameters, such as the acceleration coefficients in PSO algorithms, and integer parameters, such as the colony size in ACO algorithms. When tuning integer parameters, the values returned by the continuous optimizers are rounded to the nearest integer, which is a reasonable approach if the domain of an integer parameter is large.

This paper is structured as follows. Section 2 gives a short overview of the offline tuning problem and Sect. 3 describes the continuous optimization algorithms we use as tuning algorithms. The tuning problems that we consider are introduced in Sect. 4. The experimental setup and the results are discussed in Sect. 5. Section 6 analyses the parameter space of the tuning problems considered, and Sect. 7 concludes the paper.

2 Offline tuning of optimization algorithms

The problem of tuning algorithms offline (*tuning problem* for short) comprises an optimization algorithm with parameters that are to be set to appropriate values, and an application problem that is to be solved by the given optimization algorithm. By *parameter configuration* (*configuration* for short) we refer to a setting of all the parameters of the optimization algorithm to be tuned. The objective of the tuning problem is to find a configuration of the optimization algorithm that results in the best possible performance on the considered application problem for unseen problem instances. For finding such a performance-optimizing

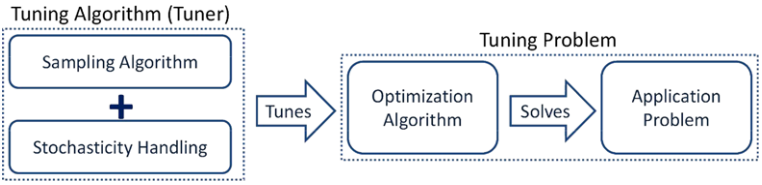

Fig. 1 A tuning problem consists of an optimization algorithm to be tuned, and an application problem that is to be solved by this optimization algorithm. The goal is to optimize the performance of the optimization algorithm on (unseen instances of) the application problem. A tuning algorithm for addressing the tuning problem usually consists of a sampling algorithm that generates candidate parameter configurations, and a stochasticity handling technique that deals with the stochasticity during the evaluation of the parameter configurations

configuration, a *tuning algorithm* (*tuner* for short) can be applied. Such a tuning algorithm consists of a sampling algorithm that generates candidate parameter configurations and a way to handle the stochasticity of the tuning problem. The overall tuning process is illustrated in Fig. 1.

An important aspect of the tuning problem is that it is stochastic. The stochasticity of the tuning problem is due to the randomness of the optimization algorithm to be tuned and the fact that it is not known at tuning time which specific instances will be solved in the future. The objective of the tuning problem is to optimize the expected performance of the underlying optimization algorithm with respect to the possible instances it will tackle. The performance expectation cannot be computed analytically, and has therefore to be estimated in a Monte Carlo fashion. For a more formal and precise definition of the tuning problem, the interested reader is referred to Birattari (2009).

In this article, the goal is to evaluate the quality of tuners. The evaluation of a tuner is divided into two phases, namely the *tuning phase* and the *testing phase*. In the tuning phase, a configuration $\bar{\theta}$ of the optimization algorithm to be tuned is selected based on a set of training instances of the application problem. The quality of $\bar{\theta}$ is then assessed during the testing phase based on the set of test instances of the application problem. The evaluation on an independent test set is crucial, since the algorithm configurations should perform well on the unseen future instances; in other words, the algorithm configuration should generalize (Birattari et al. 2006).

3 Tuning algorithms

As stated in Sect. 2, a tuning algorithm is composed of a sampling algorithm and a stochasticity handling method. In what follows, we describe the possible choices we have considered in this article for these two components.

3.1 Sampling algorithms

In this study, only real-valued parameters or integer parameters with a large domain are considered. In this case, the tuning problem can be cast in terms of a continuous optimization problem if integer parameters with a large domain are handled by rounding. We therefore consider state-of-the-art black-box continuous optimization algorithms as sampling algorithms. The five continuous optimization algorithms we consider are taken from the literature on mathematical optimization and evolutionary computation.

3.1.1 Bound optimization by quadratic approximation (BOBYQA)

BOBYQA (Powell 2009) is a model-based trust-region algorithm for derivative-free optimization. It extends the NEWUOA algorithm (Powell 2006) by enabling it to handle bound constraints. At each iteration, on the basis of m sample points, BOBYQA identifies a quadratic model that covers the current trust region. BOBYQA first determines the point that minimizes the quadratic model; then, the actual value of this point is determined by direct evaluation; finally, the model is refined by considering the actual value of the generated point. If in these steps a new best point is found, the center of the trust region is moved to this point and the radius of the region is enlarged; otherwise, if the new point is not the best, the radius of the trust region is reduced. For identifying the quadratic model, it is recommended that $m = 2d + 1$ points are used (Powell 2009), where d denotes the dimensionality of the search space. NEWUOA (BOBYQA without bound constraints) is considered to be a state-of-the-art algorithm for derivative-free continuous optimization (Auger et al. 2009).

3.1.2 Mesh adaptive direct search (MADS)

MADS (Audet and Dennis 2006) is a mesh-based direct search algorithm that systematically adapts the mesh coarseness, the search radius, and the search direction. At each iteration, a number of points lying on a mesh are sampled. If a new best point is found, the mesh center is moved to this point, the mesh is made coarser, and the search radius is increased; otherwise, the mesh is made finer and the search radius is reduced. MADS extends the generalized pattern search methods (Torczon 1997) by a more extensive exploration of the variable space. For example, it allows sampling of mesh points that are at different distances from the incumbent point, and sampling from possibly infinitely many search directions.

3.1.3 Covariance matrix adaptation evolution strategy (CMAES)

CMAES (Hansen 2006) is a (μ, λ) evolutionary strategy algorithm: at each iteration, λ offspring points are generated by the μ elite parent points. In CMAES, the offspring are sampled from a multivariate Gaussian distribution. The center of this distribution is a linear combination of the elite parent points. The covariance matrix of the distribution is automatically adapted by taking into account the search trajectory in order to better predict the influence of the variables and their interactions. CMAES is considered to be a state-of-the-art evolutionary algorithm (Auger et al. 2009).

The default CMAES is further improved here by using a uniform random sampling in the first iteration, instead of a biased Gaussian distribution sampling. The best uniformly sampled point will serve as a starting point for CMAES. This modification results in statistically significant improvements in our experiments, especially when the total number of points sampled in a run of CMAES is small, which is usually the case in tuning problems.

3.1.4 Uniform random and iterated random sampling (URS & IRS)

URS and IRS are used as a baseline for the assessment of other tuning algorithms. They are the sampling methods that are used in two tuners, known as F-Race(RSD) and I/F-Race (Birattari et al. 2010), respectively. URS samples a given space uniformly at random, while IRS has a mechanism to focus the search on promising regions. IRS is a model-based optimization algorithm (Zlochin et al. 2004). As in I/F-Race, in IRS we keep a list of $2 + \mathit{round}(\log_2 d)$ best points, where d is the dimension of the search space. At each iteration, a set of new points are generated around these best points. Each newly generated

point is sampled from a Gaussian distribution centered at one of the best points, and the standard deviation decreases over time in order to force convergence of the search around the best points. The list of the best points is updated after each iteration by considering the new best points.

3.2 Handling stochasticity

Handling stochasticity is an important aspect in the tuning problem. There are two sources of stochasticity: the randomized nature of the optimization algorithm being tuned, and the sampling of the instances of the application problem. We considered the following two mechanisms to deal with the stochasticity of the tuning problem.

3.2.1 Repeated evaluation

The most straightforward approach is to evaluate the objective function more than once and return the average value as an estimate of the expected value. Here, *nr* refers to the number of repetitions of the objective function evaluation. The advantages of this approach are that it is simple and that the confidence in the estimate can be expressed as a function of *nr*. We tested repeated evaluation with all sampling methods using different values of *nr*. The main disadvantage of this technique is that it is blind to the actual quality of the parameter configurations being re-evaluated.

3.2.2 F-Race

F-Race is a technique aimed at making a more efficient use of computational power than done by repeated evaluation. Given a set of parameter configurations, the goal of F-Race is to select the best one. F-Race does so by evaluating the configurations instance-by-instance and by eliminating inferior configurations from consideration as soon as statistical evidence is gathered against them. The early elimination of statistically inferior configurations focuses computational resources on the more promising ones. F-Race terminates when either only one configuration remains, or a predefined computational budget is reached. The elimination mechanism of F-Race is independent of the composition of the initial set of parameter configurations. It is thus possible to integrate F-Race with any sampling method that requires selecting the best configurations from a given set.

3.3 Tuners

In the context of the tuning problem, a *point* sampled by a sampling algorithm corresponds to a parameter configuration of the underlying optimization algorithm being tuned. Since the evaluation of each point is stochastic (see Sect. 2), a sampling algorithm should be combined with a stochasticity handling technique to form a *tuner*. From the possible combinations of the five sampling algorithms and the two ways of handling stochasticity, we can obtain a number of tuners. Repeated evaluation can be combined with any of the five sampling algorithms. URS, IRS, MADS, and CMAES can also be combined in a rather straightforward way with F-Race. For BOBYQA, however, the combination with F-Race is not feasible because (i) BOBYQA generates one single configuration per iteration and therefore it does not need to select the best out of a set; and (ii) the numerical evaluations of each configuration are used in the identification of the quadratic model. Thus, BOBYQA requires that each configuration is evaluated using the same instances. As a result, we have nine possible tuners.

4 Benchmark tuning problems

In order to compare the performance of the tuners described in Sect. 3, we devised two sets of case studies. One set of case studies is obtained by considering the application of an ACO algorithm to the traveling salesman problem (TSP); another set of case studies is obtained by considering a PSO algorithm applied to continuous optimization.

4.1 Ant colony optimization—traveling salesman problem

As first set of case studies, we have chosen $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System ($\mathcal{MM}$AS) (Stützle and Hoos 2000), one of the most successful ACO algorithms, as the optimization algorithm to be tuned. The application problem being addressed is the TSP. In the TSP, one is given a graph $G = (V, A)$, where V is the set of nodes and A is the set of arcs that fully connect the graph. Each arc (i, j) has associated a cost d_{ij}. The goal of the TSP is to find a minimum cost Hamiltonian cycle on the given graph G.

$\mathcal{MM}$AS is an iterative algorithm in which a colony of m ants is deployed to construct solutions during each iteration. Initially, each ant is placed on a randomly chosen node. At each step of the solution construction, an ant k at node i stochastically selects the next node j to go to. The stochastic choice is biased by two factors: the pheromone trail value $\tau_{ij}(t)$, where t is the iteration counter, and the locally available heuristic information η_{ij}, which in the TSP case is set to $\eta_{ij} = 1/d_{ij}$. An ant k located at node i selects to go to node j with probability

$$p_{ij}^k(t) = \begin{cases} \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}, & \text{if } j \in \mathcal{N}_i^k, \\ 0, & \text{otherwise,} \end{cases} \tag{1}$$

where $\mathcal{N}_i^k$ denotes the set of nodes ant k can move to from node i, and α and β are parameters that control the relative influence of pheromone trails and heuristic information, respectively. At the end of each iteration, on each arc (i, j) the pheromone trails are updated by

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{\text{best}}(t), \tag{2}$$

where ρ, $0 < \rho \leq 1$, denotes the fraction of pheromone that is evaporated. In $\mathcal{MM}$AS, only one ant is allowed to deposit pheromone at each iteration, being either the ant with the best solution generated so far, the ant with the best solution since a re-initialization of the pheromone trails, or the ant with the best solution that was generated in the current iteration. The amount of pheromone deposited is set to

$$\Delta\tau_{ij}^{\text{best}}(t) = \begin{cases} 1/f(s^{\text{best}}(t)), & \text{if } (i, j) \text{ is used by the best ant in iteration } t, \\ 0, & \text{otherwise,} \end{cases} \tag{3}$$

where $f(s^{\text{best}}(t))$ denotes the solution cost of solution $s^{\text{best}}(t)$. One of the most important features that distinguishes $\mathcal{MM}$AS from other ACO algorithms is the use of a maximum and minimum bound on the pheromone trail values, $\tau_{\min} \leqslant \tau_{ij} \leqslant \tau_{\max}$, to avoid search stagnation. The maximum pheromone trail is set to

$$\tau_{\max} = \frac{1}{\rho \cdot f(s^{\text{gb}}(t))}, \tag{4}$$

Table 1 Range and default value of each parameter of $\mathcal{MM}$AS that we tuned

Parameter	α	β	ρ	m	γ	nn	q_0
Range	[0.0, 5.0]	[0.0, 10.0]	[0.0, 1.00]	[1, 1200]	[0.01, 5.00]	[5, 100]	[0.0, 1.0]
Default	1.0	2.0	0.5	25	2.0	20	0

Table 2 Parameters that we tuned in each case study of $\mathcal{MM}$AS. We have studied the tuning problem with two to six parameters to be tuned. There are three case studies for each number of parameters from two to six. Hence, there are 15 case studies in total

#parameters			
2	$\alpha\ \beta$	$\rho\ m$	$\gamma\ nn$
3	$\alpha\ \beta\ m$	$\beta\ \rho\ nn$	$\rho\ \gamma\ nn$
4	$\alpha\ \beta\ \rho\ m$	$\alpha\ \beta\ \gamma\ nn$	$\rho\ m\ \gamma\ nn$
5	$\alpha\ \beta\ \rho\ m\ nn$	$\alpha\ \beta\ \rho\ m\ \gamma$	$\alpha\ \beta\ m\ \gamma\ nn$
6	$\alpha\ \beta\ \rho\ m\ \gamma\ nn$	$\alpha\ \beta\ \rho\ m\ \gamma\ q_0$	$\alpha\ \beta\ \rho\ m\ nn\ q_0$

where $f(s^{\mathrm{gb}}(t))$ denotes the solution cost of the global best solution, which will be updated at every iteration. The initial pheromone trail is set to the upper bound $\tau_0 = \tau_{\max}$, where $f(s^{\mathrm{gb}}(0))$ in (4) is determined by a simple estimation. The lower bound is set to

$$\tau_{\min} = \frac{\tau_{\max}}{\gamma \cdot n}, \tag{5}$$

where n is the dimension of the problem, which in the TSP is given by the number of nodes, and γ is a parameter. Another parameter when applying $\mathcal{MM}$AS to the TSP is nn, which determines the size of each node's candidate list (Stützle and Hoos 2000), from which the next node is selected.

We have also studied the *pseudo-random proportional action choice rule* (Dorigo and Gambardella 1997) in $\mathcal{MM}$AS, as this is shown to be promising in Stützle and Hoos (1998). Here, with probability q_0, $0 \leq q_0 < 1$, the ant deterministically chooses as the next node j the node for which $[\tau_{ij}(t)]^{\alpha} \cdot [\eta_{ij}]^{\beta}$ is maximal; with probability $1 - q_0$, the ant probabilistically chooses the next node according to (1).

The range and the default values of the seven studied parameters are listed in Table 1. The default values that we adopted are suggested by the ACOTSP software (Stützle 2002).[1] We have defined three case studies for each number $d \in \{2, 3, 4, 5, 6\}$ of parameters to be tuned, resulting in $3 \times 5 = 15$ case studies. The parameters that were tuned in each of the case studies are listed in Table 2. In each of the case studies, the parameters that were not tuned were fixed to their default values (see Table 1). The goal of each case study was to find a parameter configuration such that $\mathcal{MM}$AS performed well on the TSP within a given time limit, which was set in our experiments to five seconds.

Note that local search is not applied in this study. It is widely recognized that the hybrid with local search is essential in obtaining high performing ACO algorithms (Dorigo and Stützle 2004). However, a previous study (Yuan et al. 2010b) has also shown that, when incorporating local search in ACO algorithms, the landscape of the parameter space becomes

[1] Dorigo and Stützle (2004) proposed $\rho = 0.02$ and $m = n$ as a default parameter setting for $\mathcal{MM}$AS. For TSP instances in the range of the instance size we use, Stützle (1999) uses the parameter setting $\rho = 0.04$ and $m = n/2$. We experimentally compared these two proposed settings with the default setting of the ACOTSP software, and adopted the one suggested by the ACOTSP software as default for this study because, in our context, it resulted in significantly better performance than the other two settings.

rather flat. Therefore, the algorithm performance becomes relatively insensitive to the parameters, which makes it not an ideal testbed for studying tuning algorithms.

The TSP instances were generated by the DIMACS instance generator (Johnson et al. 2001). All instances are Euclidean TSP instances with 750 nodes, where the nodes are uniformly distributed in a square of side length 10 000. 1 000 such instances are generated for the tuning phase, and 300 for the testing phase.

4.2 Particle swarm optimization—Rastrigin functions

As the second set of case studies we have chosen the constricted PSO (cPSO) algorithm (Clerc and Kennedy 2002), a prominent PSO algorithm, as the optimization algorithm to be tuned. The application problems considered here are a family of functions derived from the Rastrigin function, a well-known benchmark function in continuous optimization; details on the family of functions are provided below.

In PSO algorithms, a population of simple agents, called *particles*, moves in the domain of an objective function $f : \Theta \in \mathcal{R}^n$, where n is the number of variables to optimize. Each particle i at iteration t has three vectors associated with it:

1. Current position, denoted by $\mathbf{x}_i^t$: This vector stores the latest candidate solution generated by the particle.
2. Velocity, denoted by $\mathbf{v}_i^t$: This vector represents the particle's search direction.
3. Personal best position, denoted by $\mathbf{b}_i^t$: This vector stores the best solution found by the particle since the beginning of the algorithm's execution.

In addition, a neighborhood relation is defined among the particles through a *population topology* (Kennedy and Mendes 2006; Dorigo et al. 2008), which can be thought of as a graph in which nodes represent particles, and edges represent neighbor relations. The behavior of particles in PSO algorithms is usually influenced by the best neighbor; in the following, $\mathrm{n}(i)$ gives the index of the best neighbor of particle i. Figure 2 shows four topologies with different levels of connectivity for a swarm of nine particles. These topologies are specified by a parameter called *neighborhood radius*, which is the number of particles on each side of a particle if the particles are arranged as in the figure. The radius parameter takes values in the range $[1, \lfloor N/2 \rfloor]$, where N is the size of the swarm. If the radius is equal to one, the resulting topology is known as a ring topology, and if the radius is equal to $\lfloor N/2 \rfloor$, the topology is fully connected.

In the particular PSO variant we use here, the constricted PSO, a particle i moves independently for each dimension j using the following rules:

$$v_{ij}^{t+1} \leftarrow \chi\left(v_{ij}^t + \phi_1 U_1\left(b_{ij}^t - x_{ij}^t\right) + \phi_2 U_2\left(b_{\mathrm{n}(i)j}^t - x_{ij}^t\right)\right), \tag{6}$$

and

$$x_{ij}^{t+1} \leftarrow x_{ij}^t + v_{ij}^{t+1}, \tag{7}$$

where χ is a parameter called *constriction factor*, ϕ_1 and ϕ_2 are two parameters called *acceleration coefficients*, and U_1 and U_2 are two independent, random numbers uniformly distributed in the range $[0, 1)$.

Five parameters are considered in this study, namely, χ, ϕ_1, ϕ_2, N, and the neighborhood radius. The actual range of the neighborhood radius depends on the value taken by N. Thus, we introduce a parameter p that maps linearly the continuous range $[0, 1]$ into the range $[1, \lfloor N/2 \rfloor]$. More details about the parameters used in the experiments are listed in Table 3.

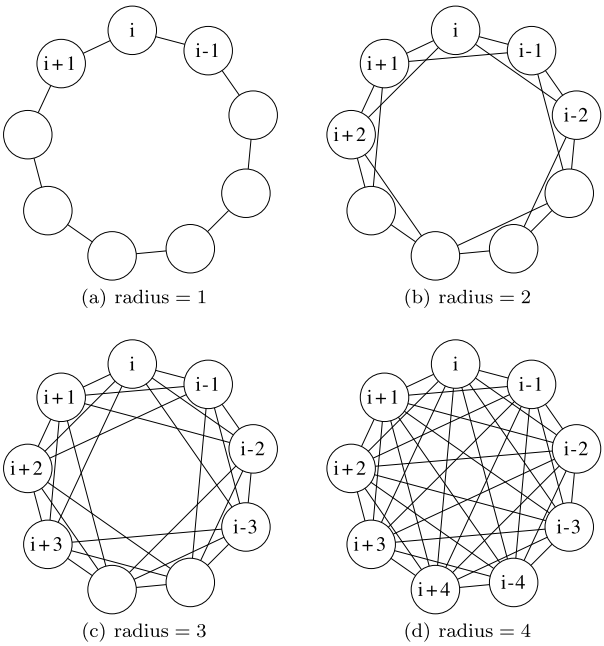

Fig. 2 Examples of population topologies with different connectivity degrees, resulting from neighborhoods of different size. In (**a**), radius = 1, in (**b**), radius = 2, in (**c**), radius = 3, and in (**d**), radius = 4

Table 3 Range and the default value of each parameter of cPSO that we tuned

Parameter	χ	ϕ_1	ϕ_2	N	p
Range	[0.0, 1.0]	[0.0, 4.0]	[0.0, 4.0]	[4, 1000]	[0.0, 1.0]
Default	0.729	2.05	2.05	30	1

Table 4 Parameters that we tuned in each case study of cPSO. We have studied the tuning problem with two to five parameters to be tuned. There are three case studies for two, three, and four parameters, and one case study for five parameters. Hence, there are 10 case studies in total

#parameters			
2	χ ϕ_1	ϕ_1 ϕ_2	N p
3	χ ϕ_1 ϕ_2	ϕ_1 ϕ_2 N	χ N p
4	χ ϕ_1 ϕ_2 N	χ ϕ_1 ϕ_2 p	ϕ_1 ϕ_2 N p
5	χ ϕ_1 ϕ_2 N p		

The default values of the parameters were set as suggested by Poli et al. (2007). Each run is stopped after 10^5 function evaluations. Similar to what we did in Sect. 4.1, we generated three case studies for each number of parameters $d \in \{2, 3, 4\}$, and one for $d = 5$. Thus, in total $3 \times 3 + 1 = 10$ case studies were defined. The parameters that were tuned in each of the case studies are listed in Table 4. In each case study, the parameters that were not tuned were fixed to their default values (see Table 3).

For all the experiments, the cPSO algorithm was applied to a family of functions derived from the Rastrigin function: $nA + \sum_{i=1}^{n} (x_i^2 - A\cos(\omega x_i))$. The difficulty of this function can be adjusted by changing the values of the parameters n, A, and ω. In our experiments,

we set $\omega = 2\pi$, and we varied the amplitude A to obtain functions with different fitness distance correlations (FDC) (Jones and Forrest 1995). We sampled the amplitude 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 a mean of 0.7 and a standard deviation of 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 are set to $[-5.12, 5.12]^n$ and $n = 100$, respectively. A total of 1 000 instances were generated for tuning and another 1 000 instances for testing.

5 Experiments

This section reports the results of the experiments we ran to compare the tuners. Before presenting the results, we provide some information on the experimental setup.

5.1 Experimental setup

For the tuners that are obtained by combining the sampling algorithms with repeated evaluation, an appropriate number nr of repeated evaluations needs to be chosen. In our experiments, we consider four levels with $nr \in \{5, 10, 20, 40\}$. This choice is made following the study in Yuan et al. (2010b), where the sampling algorithm MADS is used as a tuner. The results showed that the best nr value differed a lot from problem to problem, and ranged from 5 to 40. For F-Race, the default version as described by Birattari (2009) is used.

In our experiments, all parameters are tuned with a precision of two significant digits. This was done by rounding each sampled value. This choice was made because we observed during experimentation that reserving two significant digits in the tuning phase leads to the highest performance of the tuned algorithms. To avoid re-evaluating a same algorithm configuration on a same instance, we store historical evaluation results in an archive. If the same algorithm configuration is sampled again, the results on the evaluated instances are read from the archive instead of rerunning the optimization algorithm.

For each of the 25 case studies mentioned in Sect. 4, we run the tuners with four different levels of the *tuning budget*. By tuning budget we mean the maximum number of times that the underlying optimization algorithm can be run during the tuning phase. Let d be the number of parameters to be tuned. The minimum level of the tuning budget is chosen to be $B_1 = 40 \cdot (2d + 2)$, which results in a budget $B_1 = 240$ when $d = 2$. The setting of B_1 is chosen in this way since BOBYQA needs at least $2d + 1$ 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 other three levels of tuning budget are $B_i = 2^{i-1} \cdot B_1$, $i = 2, 3, 4$, thus doubling the budget from level to level.

Each tuner runs 10 trials on each of the four budget levels of each case study. Each *trial* is an execution of a tuning phase followed by a testing phase. In the testing phase, the final parameter configuration selected in the tuning phase is assessed on a set of test instances.

For the purpose of reducing experimental variance, we adopted the technique of common random instance order and common random seed: in each trial, a fixed random order of the tuning instances is used; additionally, each instance in each trial is assigned a random number, which serves as the common random seed each time this instance is used for evaluation.

The tuners are compared statistically based on the performance the optimization algorithms reach using the tuned configurations on the test instances. The test results of each

tuner are compared with each other using the pairwise Wilcoxon signed rank test with blocking on each instance. In case of multiple comparisons, Holm's adjustment is used for multiple test correction.

The experiments were carried out on computers equipped with two quad-core XEON E5410 CPUs running at 2.33 GHz with 2×6 MB second level cache and 8 GB RAM. Only one CPU core was used for each run due to the sequential implementation of both the ACOTSP software and the PSO code. The computers run under Cluster Rocks Linux version 5.3/CentOS 5.3. The ACOTSP and PSO programs were compiled using gcc-4.1.2-46.

5.2 Restart mechanism of the tuning algorithms

5.2.1 Basic restart mechanism

In our experiments, three sampling algorithms, CMAES, MADS and BOBYQA, are extended by a restart mechanism. The restart is triggered whenever stagnation is detected. Stagnation can be detected when the search coarseness of the mesh or the trust-region radius drops to a very low level; for example, below the adopted precision (here, two significant digits). Considering the number of restarts, it is important to mention that the speed of convergence of the three algorithms is very different. BOBYQA converges much faster than MADS, and MADS, in turn, much faster than CMAES. In fact, faster convergence also leads to a higher number of restarts and, thus, the different algorithms may benefit differently from the possibility of restarting.

One issue about the restart mechanism is how to choose the initial point of each restart. In our experiments, each restart begins from a uniformly randomly sampled point in the parameter space. Other initial points could be taken into account as well. For example, we have tried restarting the algorithms from the best configuration found so far; however, this led to statistically significantly worse results.

Next, we studied what is the appropriate value of *nr* for the tuners that use a restart mechanism. This study was carried out on eight case studies from $\mathcal{MM}$AS. As the tuners we chose BOBYQA and MADS, since they converge the fastest and, hence, have the largest number of restarts. The comparison of the ranks obtained using four *nr* settings for the basic restart mechanism, namely $nr \in \{5, 10, 20, 40\}$, across the eight case studies is shown in the left four boxplots in each of the two plots of Fig. 3. Each of these boxplots gives the distribution of the ranks of a tuner according to its average performance in each case study at each tuning budget level. It is shown that $nr = 10$ appears to be the best setting for both BOBYQA and MADS, followed by $nr = 5, 20, 40$.

5.2.2 Restart mechanism with post-selection by F-Race

In the basic restart mechanism, the best setting of *nr* is relatively small, 10 followed by 5. In such cases, the global best configuration is identified among all the best configurations obtained at each restart by only a small number of evaluations. The small number of evaluations might make the selection unreliable, leading to the danger of over-tuning (Birattari 2009). To avoid this, we propose a restart mechanism with post-selection by F-Race to select the best of the restart best configurations.

The post-selection works as follows. Each restart best configuration is stored, and in the post-selection the best of all restart best configurations is selected by F-Race. The tuning budget reserved for the post-selection F-Race is ω_{post} times the number of restart best configurations. The factor ω_{post} in the repeated evaluation experiments is determined by

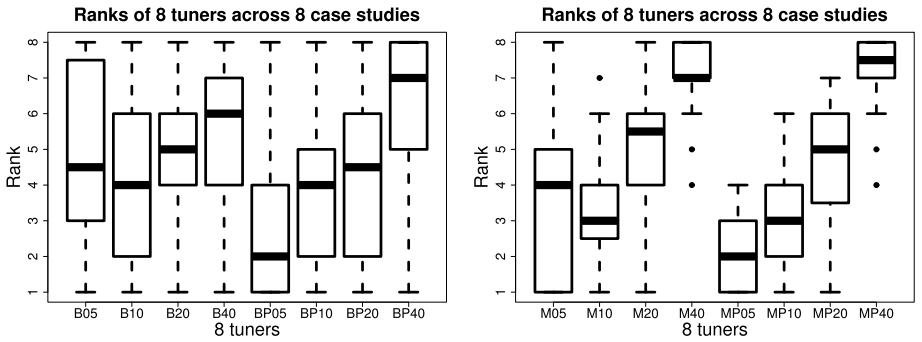

Fig. 3 Ranking comparisons of different settings for the tuners based on BOBYQA (*left*) and MADS (*right*). Each plot shows eight different tuners of two restart versions: without post-selection (the left four plots dubbed "B" or "M" for BOBYQA or MADS respectively) or with post-selection by F-Race (the right four plots dubbed "BP" or "MP" for BOBYQA or MADS, respectively). Each version includes four levels of *nr* values, that is, the number of evaluations at each sampling point; the four levels are 5, 10, 20, 40. Restart with post-selection F-Race and $nr = 5$ turned out to be the best setting

$\omega_{\text{post}} = \max\{5, (20 - nr)\}$. In the post-selection F-Race, we start the Friedman test for discarding candidates from only the $\max\{10, nr\}$th instance, instead of five as in the default F-Race setting; this helps to make the selection more conservative.

We tested post-selection F-Race on tuners BOBYQA and MADS, and compared it with the basic restart mechanism in the same eight case studies of $\mathcal{MM}$AS as used in Sect. 5.2.1. The results with post-selection F-Race for tuners BOBYQA and MADS are shown in the right four boxplots in each of the two plots of Fig. 3. The performance of the restart mechanism with post-selection F-Race and $nr = 5$ is statistically significantly better than all other settings for both, MADS and BOBYQA. Thus, it is the method of choice.

5.3 Combination of F-Race with the sampling algorithms

Next, we examine the performance of the tuners that are obtained by combining the sampling algorithms IRS, URS, MADS and CMAES directly with F-race (see Sect. 3). Recall that the F-Race hybrid with BOBYQA is not feasible, as explained in Sect. 3. Before presenting the results, we discuss some design issues for the combination with F-Race.

5.3.1 F-Race(RSD) and I/F-Race

As explained in Sect. 3.1, URS and IRS are the sampling methods of F-Race(RSD) and I/F-Race (Birattari et al. 2010), respectively. However, F-Race(RSD) and I/F-Rase were never compared to the sampling methods with a fixed number of evaluations and this article fills the gap. The results of this comparison are shown in Fig. 4. The comparison of F-Race(RSD) and URS with fixed number of evaluations $nr \in \{5, 10, 20, 40\}$ is shown in the first row; the comparison of I/F-Race and IRS with fixed number of evaluations is shown in the second row. These results confirm that the sampling methods IRS and URS are better combined with F-race than using a fixed number of evaluations.

5.3.2 MADS/F-Race and the incumbent protection mechanism

MADS/F-Race (Yuan et al. 2010b) uses F-Race to identify the incumbent point among the old incumbent point and the new sample points in each MADS iteration. The basic version of

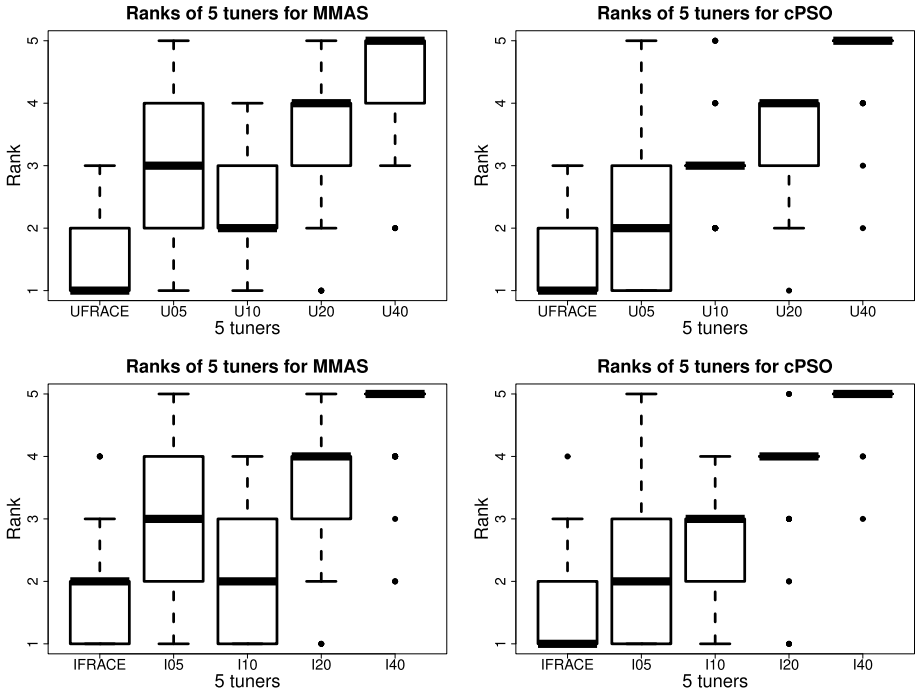

Fig. 4 Ranking comparisons of two settings of tuners, direct hybrid with F-Race and the fixed number of evaluations, on two sampling algorithms, namely, URS (*first row*) and IRS (*second row*). *The left column* shows comparisons across 15 case studies of $\mathcal{MM}$AS, and *the right column* shows comparisons across 10 case studies of cPSO. In each plot, the leftmost box shows the tuner using the direct hybrid with F-Race, namely, F-Race(RSD) or I/F-Race (indicated with UFRACE and IFRACE respectively in the plots); and the other four boxes show the tuners using a fixed number of evaluations $nr \in \{5, 10, 20, 40\}$

MADS/F-Race is extended here with an incumbent protection mechanism, which is inspired by the intensification mechanism used in SPO$^+$ (Hutter et al. 2009a). In each iteration, if an incumbent p^* that has been evaluated on N^* instances would be eliminated in F-Race by another point p with $n < N^*$ instances, we keep p^* in the race and double the number of evaluated instances $n := \min(2n, N^*)$. This is done until either the new point p is eliminated from the race, or n reaches N^*. Then we free the protection and continue the race normally. The incumbent protection mechanism is done to avoid the incumbent being eliminated by some inferior but "lucky" point with few evaluations. Experimental results show that MADS/F-Race with the incumbent protection mechanism results in a statistically significant improvement over the basic MADS/F-Race.

5.3.3 CMAES/F-Race

CMAES is a (μ, λ) evolutionary strategy. F-Race can be integrated in each iteration to select μ elite points out of λ points. The race stops when either at most μ points survive, or a budget of $10 \times \lambda$ evaluations has been reached. In order to keep track of the best-so-far point, the best point of each iteration is stored, and a post-selection phase is implemented using F-Race to identify the global best point from the iteration best points. The setting of post-selection is the same as described in Sect. 5.2.2 by assigning $nr = 5$.

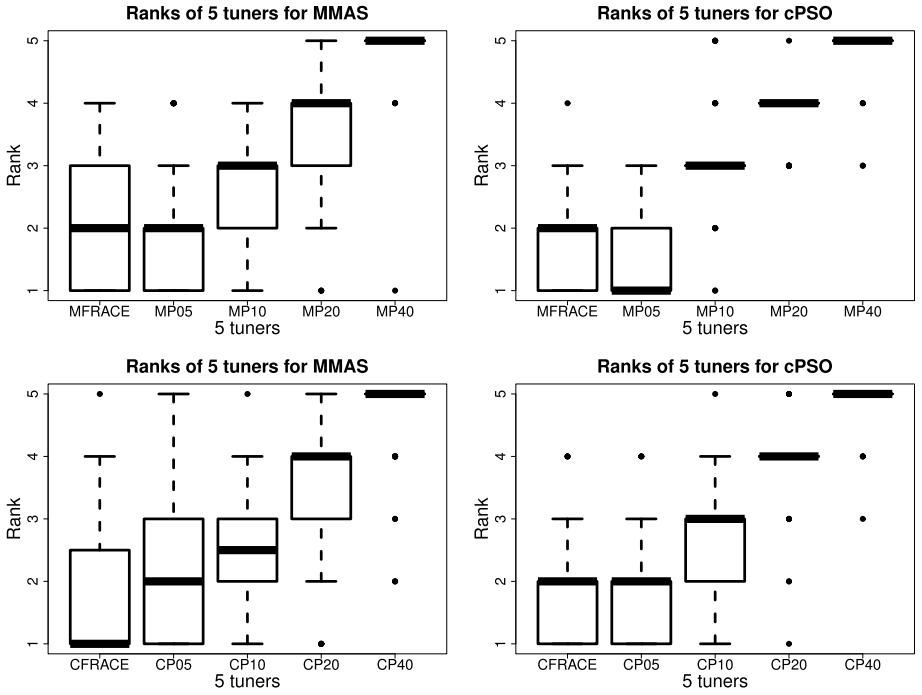

Fig. 5 Ranking comparisons of two settings of tuners, direct hybrid with F-Race and the restart with post-selection F-Race, on two sampling algorithms, namely, MADS (*first row*) and CMAES (*second row*). *The left column* shows comparisons across 15 case studies of $\mathcal{MM}$AS, and *the right column* shows comparisons across 10 case studies of cPSO. In each plot, the leftmost box shows the tuner using the direct hybrid with F-Race, namely, MADS/F-Race or CMAES/F-Race (indicated with MFRACE and CFRACE respectively in the plots); and the other four boxes show the tuners using post-selection F-Race with $nr \in \{5, 10, 20, 40\}$

5.3.4 Comparisons of F-Race hybrids to restart with post-selection F-Race

In Fig. 5, the direct F-Race hybrids are compared with their corresponding post-selection F-Race on two sampling algorithms, namely, MADS in the first row and CMAES in the second row. Each plot on the left column shows the ranking comparisons across 15 case studies of $\mathcal{MM}$AS, and each plot on the right shows comparisons across 10 case studies of cPSO. Firstly, in all plots, the direct hybrid F-Race significantly outperforms restart with post-selection and $nr \in \{10, 20, 40\}$. Comparing the F-Race hybrid to the restart with post-selection and $nr = 5$, MADS/F-Race performs slightly, yet statistically significantly worse than MADS with post-selection. In the case of CMAES, the performance of CMAES/F-Race and the version with post-selection and $nr = 5$ are similar. The former is shown to be significantly better in the case studies of $\mathcal{MM}$AS, but significantly worse in the cPSO case studies. We explain the observation that the direct F-Race hybrids are inferior to post-selection with $nr = 5$ for MADS and for the cPSO case studies with CMAES as follows: In F-Race, by default each candidate configuration is evaluated on at least 5 instances before the Friedman test is used to discard candidate configurations; hence, many candidate configurations receive more evaluations than the case $nr = 5$. When solving a tuning problem, it appears to be a good strategy to quickly have a set of good configurations based on few

instances and then to select carefully from this set, instead of evaluating each point carefully from the very beginning.

5.4 Comparisons of all tuning algorithms

Here we compare the performance of the five sampling algorithms mentioned in Sect. 3. In the following, we compare the sampling algorithms using the setting of restart with post-selection and $nr = 5$. This is done because (i) BOBYQA does not have a F-Race hybrid and (ii) the restart mechanism with post-selection F-Race and $nr = 5$ works well in comparison with the F-Race hybrids as shown in the previous section. However, for URS and IRS no post-selection is applied due to the missing restart option; still, a fixed number of evaluations, $nr = 5$, is used.

Each plot in Fig. 6 shows the average cost of each of the five tuners on one of the 15 case studies derived from the application of $\mathcal{MM}$AS to the TSP; the results for the 10 case studies derived from the application of cPSO to the family of Rastrigin functions are shown in Fig. 7. Each case study includes four levels of tuning budget.

The main conclusions from this analysis are the following. For the case studies with two to four parameters, BOBYQA is in general the best performing algorithm. However, BOBYQA's performance degrades with an increase of the number of parameters to be tuned. The performance of CMAES is relatively robust across the number of parameters to be tuned, and across the budget levels tested. This conclusion can be further and better observed in Fig. 8. Each box plot shows the ranks of the five tuners in case studies of a fixed number of parameters; the first row shows the ranks of the tuners across all numbers of parameters. The large range of the boxplots for BOBYQA reveals that the performance of BOBYQA is quite variant, even within problems with the same number of parameters. Clear examples are $\mathcal{MM}$AS with two parameters and cPSO with four parameters. The most extreme cases are two cPSO case studies with four parameters: BOBYQA performs the best in the case with χ, ϕ_1, ϕ_2, p to be tuned, but the worst in the case with ϕ_1, ϕ_2, N, p to be tuned. Finally, all the tuners tested clearly outperformed URS in most case studies (the same also holds if URS is combined with F-Race).

Figure 9 shows the average ranks of the five tuners grouped by the number of parameters to be tuned averaged across all budget levels for the $\mathcal{MM}$AS (left) and cPSO (right). These two figures confirm the observation that BOBYQA is the best for tuning small numbers of parameters, while CMAES shows rather robust performance.

5.5 Further comparisons

5.5.1 Comparison between the tuned and the default configurations

We compared the performance of the tuned parameter configurations to that of the default parameter configurations for $\mathcal{MM}$AS and cPSO. The average cost on the test instances obtained by the default parameter configurations (see Tables 1 and 3, respectively) for $\mathcal{MM}$AS is 2.54×10^7, and for cPSO is 589. Taking into account the objective ranges on the y-axis in the plots of Figs. 6 and 7, we can conclude that in all 25 case studies all tuners at even the lowest tuning budget level obtained parameter configurations that resulted in significantly better average costs than using the default parameter configurations.

As an example, we list the average percentage improvement of the parameter configurations obtained by restart CMAES with post-selection and $nr = 5$ over the default parameter configurations of $\mathcal{MM}$AS and cPSO in Tables 5 and 6, respectively. In the case studies of

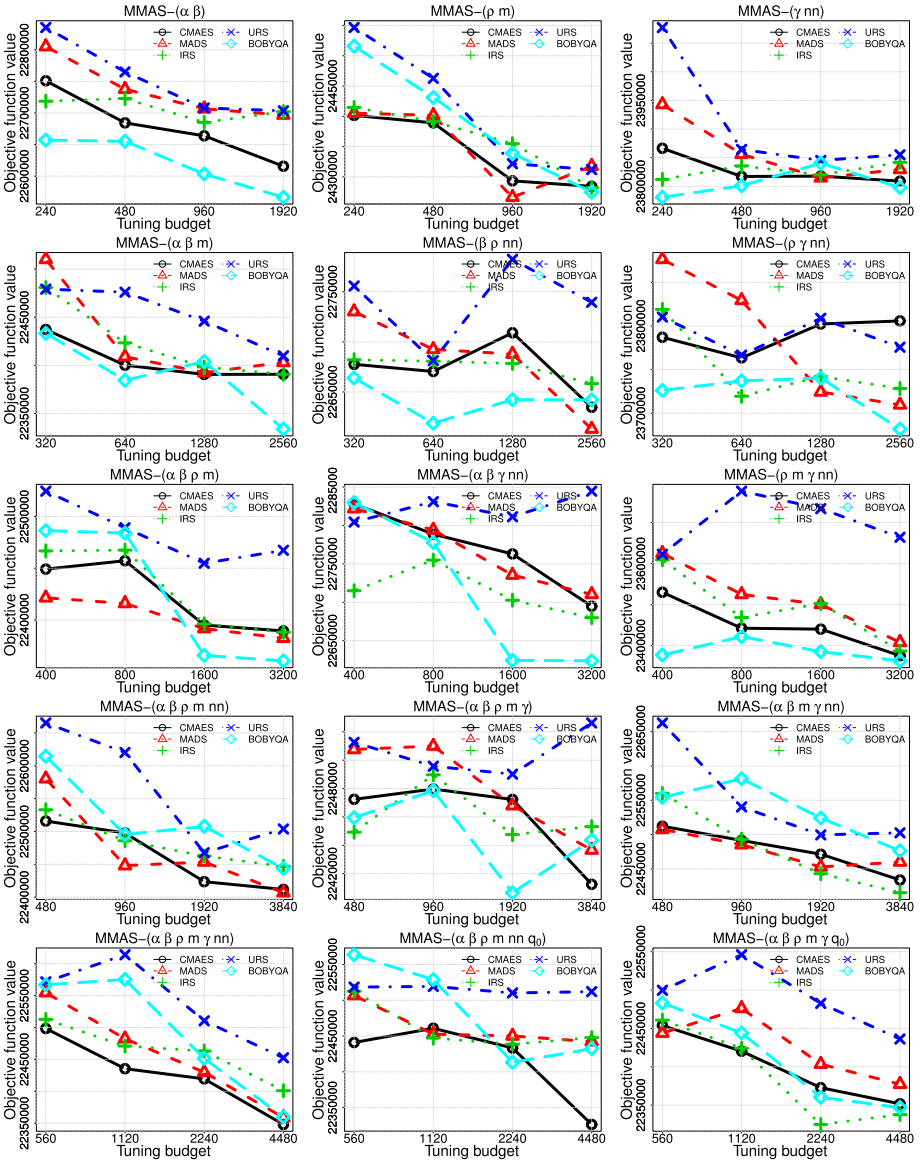

Fig. 6 Average performance over budget level of five tuners on the 15 case studies of $\mathcal{MM}$AS. Each of the five tuners uses one of the sampling methods: BOBYQA, CMAES, MADS, IRS, or URS. Each tuner handles the stochasticity using $nr = 5$; BOBYQA, CMAES, and MADS use restart and post-selection. Each plot shows the average cost measured on the test instances with respect to four budget levels in each case study. Each row presents results for a same number of parameters, from two (*top*) to six (*bottom*)

$\mathcal{MM}$AS, the tuned configurations improve over the default configuration by around 10% on average; the average improvement for cPSO is often more than 40%. Note that the default parameter configurations of $\mathcal{MM}$AS and cPSO are based on years of extensive studies by experienced researchers in the field of swarm intelligence. Our results here confirm that by

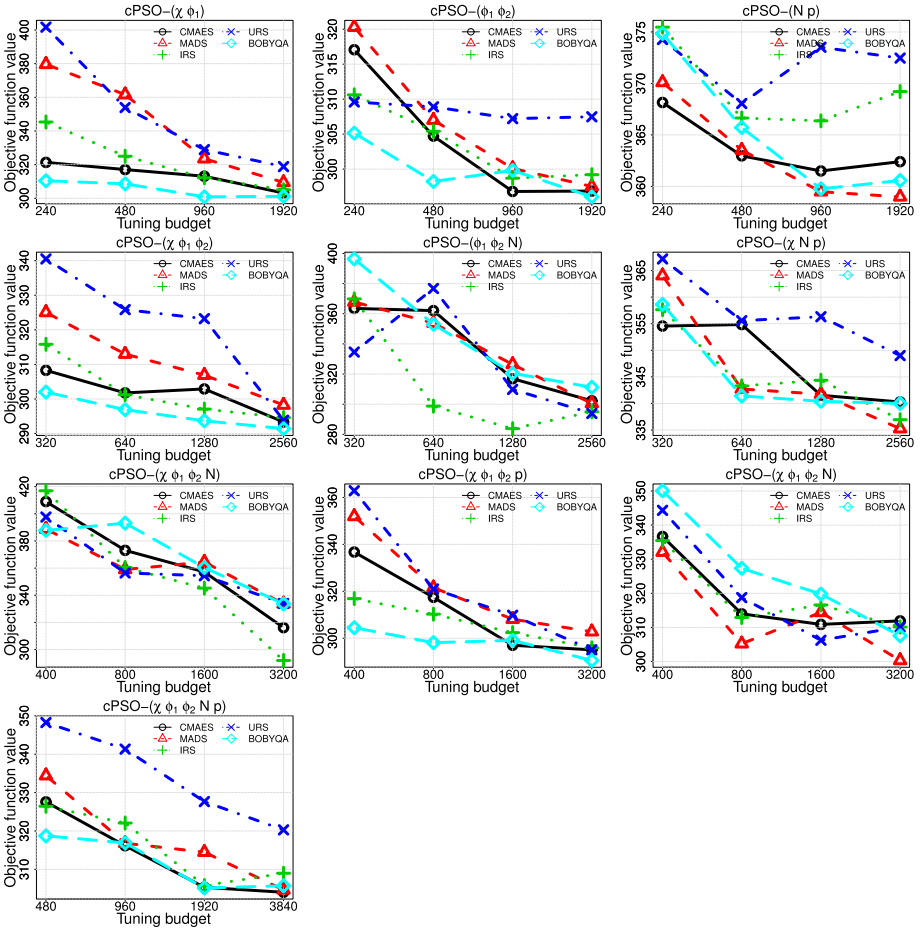

Fig. 7 Average performance over budget level of five tuners on the 10 case studies of cPSO. Each of the five tuners uses one of the sampling methods: BOBYQA, CMAES, MADS, IRS, or URS. Each tuner handles the stochasticity using $nr = 5$; BOBYQA, CMAES, and MADS use restart and post-selection. Each plot shows the average cost measured on the test instances with respect to four budget levels in each case study. Each row presents results for a same number of parameters, from two (*top*) to five (*bottom*)

using automated tuning procedures one can obtain much better performing algorithms than by setting parameters based on rules of thumb and human expertise.

5.5.2 The number of parameters to be tuned

In practice, when using tuning tools, we are often facing the question of how many parameters should be tuned. Our computational results in Tables 5 and 6 indicate that there is a tradeoff between the number of the parameters to be tuned and the tuning budget. If we consider the largest available tuning budget in the case studies of $\mathcal{MM}$AS, it is clear that the more parameters are tuned, the better is the performance. In fact, in Table 5 the tuned performance generally improves as the number of parameters increases. A similar (but less

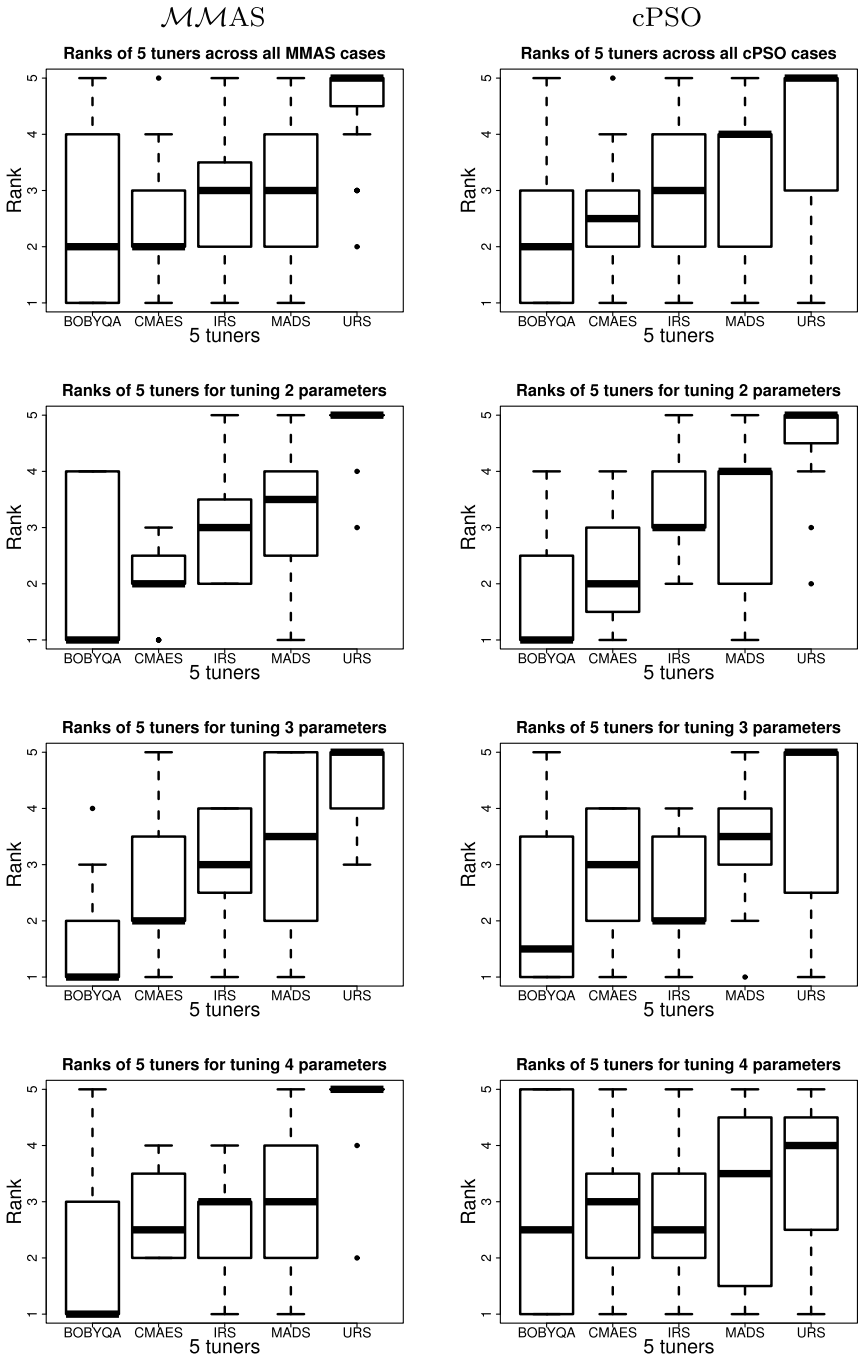

Fig. 8 Ranking comparison of five tuners for tuning $\mathcal{MM}$AS (*left column*) and cPSO (*right column*). Each of the five tuners uses one of the sampling methods, namely BOBYQA, CMAES, MADS, IRS, and URS. Each tuner handles the stochasticity using $nr = 5$; BOBYQA, CMAES, and MADS use restart and post-selection. *The first row* shows the comparisons across all possible case studies; *from the second to the sixth row*, it shows the results on the problems of a certain number of parameters from two to six

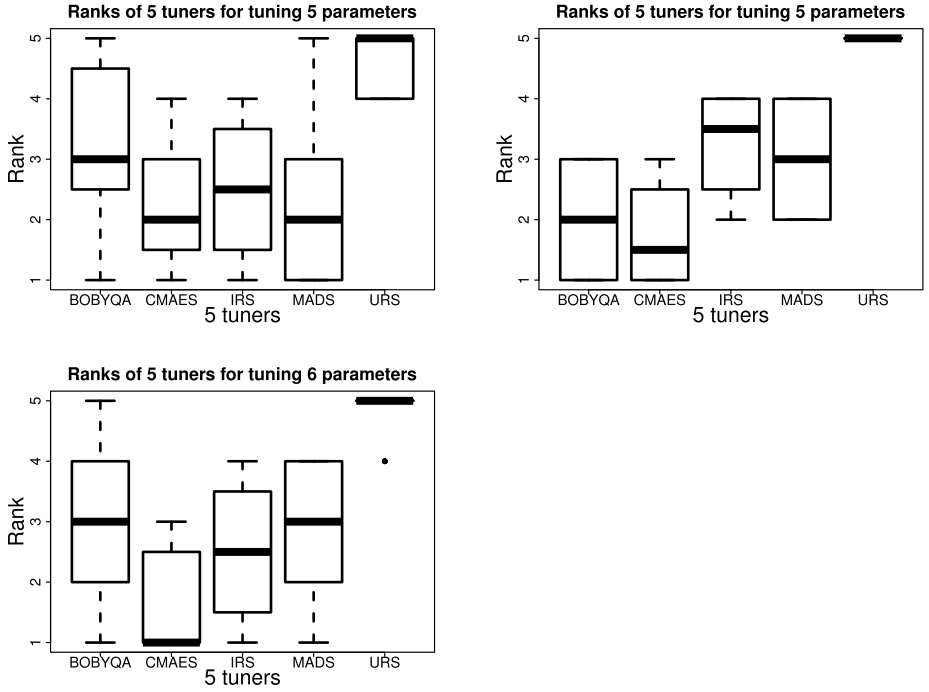

Fig. 8 (*Continued*)

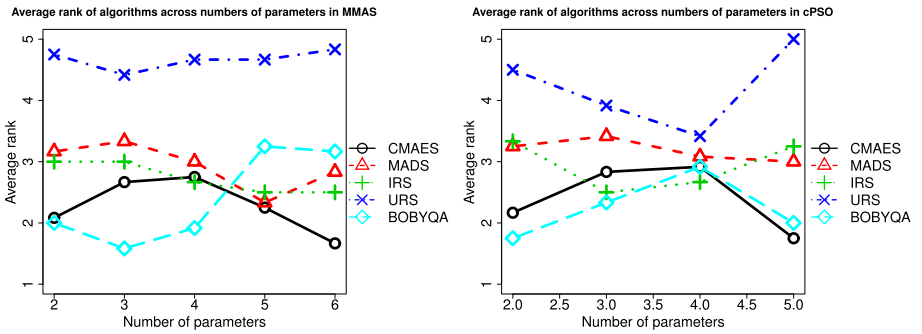

Fig. 9 Average ranks over numbers of parameters of five tuners on case studies of $\mathcal{MM}$AS (*left*) and cPSO (*right*). Each of the five tuners uses one of the sampling methods BOBYQA, CMAES, MADS, IRS, or URS. Each tuner handles the stochasticity using restart and post-selection with $nr = 5$

strong) trend can be observed from Table 6 in the case studies of cPSO. Differently from the $\mathcal{MM}$AS case studies, in the cPSO case studies the best tuned performance is not obtained by the largest number of free parameters (five), but in the case of (χ ϕ_1 ϕ_2). If we consider limitations on the tuning budget in the case studies of $\mathcal{MM}$AS of, say, no more than 1 000 runs, the best results in Table 5 would be obtained by tuning only three parameters, namely, α, β, and m. Therefore, as maybe expected, if the tuning budget is limited, less parameters should be tuned.

Table 5 Average percentage improvement of the parameter configurations tuned by CMAES with post-selection and $nr = 5$ compared to the default parameter configuration of $\mathcal{MM}$AS applied to the TSP

Budget	(α β)	(ρ m)	(γ nn)	Mean
240	10.43	3.931	6.04	6.80
480	10.69	3.980	6.23	6.97
960	10.77	4.356	6.23	7.12
1920	10.96	4.392	6.26	7.21
Budget	(α β m)	(β ρ nn)	(ρ γ nn)	Mean
320	11.66	10.72	6.35	9.58
640	11.81	10.75	6.44	9.67
1280	11.85	10.59	6.29	9.58
2560	11.85	10.89	6.28	9.67
Budget	(α β ρ m)	(α β γ nn)	(ρ m γ nn)	Mean
400	11.62	10.13	7.36	9.70
800	11.59	10.28	7.71	9.86
1600	11.83	10.38	7.72	9.98
3200	11.85	10.65	7.97	10.16
Budget	(α β ρ m nn)	(α β ρ m γ)	(α β m γ nn)	Mean
480	11.35	11.53	11.37	11.42
960	11.43	11.50	11.45	11.46
1920	11.72	11.53	11.53	11.59
3840	11.76	11.76	11.68	11.73
Budget	(α β ρ m γ nn)	(α β ρ m γ q_0)	(α β ρ m nn q_0)	Mean
560	11.42	11.60	11.65	11.56
1120	11.67	11.73	11.57	11.66
2240	11.73	11.92	11.68	11.78
4480	12.02	12.00	12.10	12.04

However, if less parameters are tuned, the selection of which parameters to tune becomes critical. For example, if the important parameters ϕ_1 and ϕ_2 of cPSO are not among the tuned ones, such as in case studies (N p) and (χ N p), or when tuning $\mathcal{MM}$AS without the most influential parameters α and β, such as in case studies (ρ m) and (γ nn), relatively small improvements over the default settings are obtained. In other words, knowledge on the impact of the parameters on algorithm performance is particularly useful in a situation of strongly limited tuning budget. Nevertheless, if a significantly large tuning budget is available, we would recommend to leave more parameters free for tuning. This incurs less risk regarding a wrong selection of the parameters to be tuned. An additional reason for leaving more parameters to be tuned is that the tuning algorithm can potentially better take parameter interactions into account.

Table 6 Average percentage improvement of the parameter configurations tuned by CMAES with post-selection and $nr = 5$ compared to the default parameter configuration of cPSO applied to a family of Rastrigin functions

Budget	$(\chi\ \phi_1)$	$(\phi_1\ \phi_2)$	$(N\ p)$	Mean
240	45.45	46.17	37.50	43.04
480	46.17	48.27	38.38	44.27
960	46.85	49.61	38.62	45.03
1920	48.54	49.60	38.47	45.54
Budget	$(\chi\ \phi_1\ \phi_2)$	$(\phi_1\ \phi_2\ N)$	$(\chi\ N\ p)$	Mean
320	47.66	38.28	39.80	41.91
640	48.77	38.53	39.75	42.35
1280	48.57	46.20	42.02	45.60
2560	50.19	48.68	42.23	47.03
Budget	$(\chi\ \phi_1\ \phi_2\ N)$	$(\chi\ \phi_1\ \phi_2\ p)$	$(\phi_1\ \phi_2\ N\ p)$	Mean
400	30.56	42.83	42.84	38.74
800	36.66	46.11	46.69	43.15
1600	39.36	49.58	47.22	45.39
3200	46.32	49.91	47.04	47.76
Budget	$(\chi\ \phi_1\ \phi_2\ N\ p)$			Mean
480	44.38			44.38
960	46.32			46.32
1920	48.16			48.16
3840	48.38			48.38

6 Parameter landscape analysis

We have performed a parameter landscape analysis for the swarm intelligence algorithms considered in this article. This study helps to understand better the tuning problem, and enriches our knowledge about the studied swarm intelligence algorithms.

6.1 The parameter landscape for two parameters

We first examine four case studies with two parameters to be tuned: $(\alpha\ \beta)$, $(\rho\ m)$ of $\mathcal{MM}$AS, and $(\chi\ \phi_1)$, $(N\ p)$ of cPSO. For each case, we define a 100×100 grid on the parameter space. Each grid vertex, which corresponds to a parameter configuration, is evaluated on the same 25 randomly selected instances, and the average evaluation value is computed. For each instance, a common random seed is used in order to reduce variance.

The parameter landscapes of the four case studies are visualized in the contour plots given in Fig. 10. They contain a single global *optimal region*, where by *optimal region* we refer to the first two levels of the contours, where all the good points lie. In the cPSO-$(N\ p)$ case, there appear to be multiple local optima within the optimal region, but this may be caused by noise due to the small number of evaluations for each parameter configuration. This unimodality of parameter landscapes for two parameters has also been observed in Steinmann et al. (1997), and shown analytically for a one-parameter space (Mengshoel 2008).

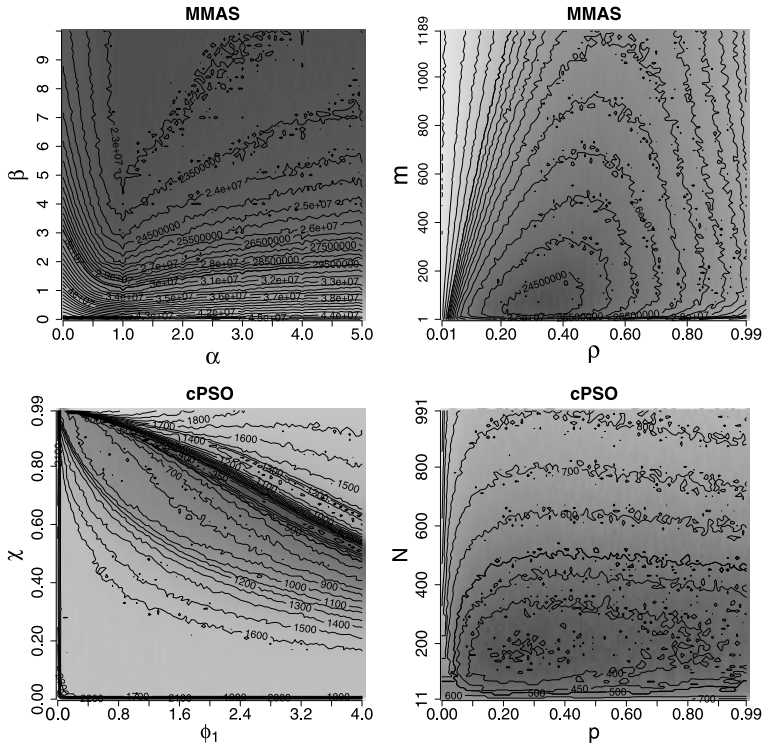

Fig. 10 Contour plots for the parameter landscapes of four case studies with two parameters to be tuned. *Top-left*: $\mathcal{MM}$AS-(α β); *top-right*: $\mathcal{MM}$AS-(ρ m); *bottom-left*: cPSO-(χ ϕ_1); *bottom-right*: cPSO-(N p). *The darker* the region is, the better those points are. Note that in the plot of cPSO-(χ ϕ_1), the region close to the χ-axis and close to the ϕ_1-axis appears to be dark because of the contour lines. It is the same case for the region close the ρ-axis in the plot of $\mathcal{MM}$AS-(ρ m)

The size of the optimal region of the parameter space, depends on the particular problem. For example, if we compare the contour plots of cPSO-(χ ϕ_1) with cPSO-(N p), the optimal region of the former lies in a very narrow valley, where ϕ_1 takes values from 2 to 4 and χ takes values from 0.5 to 0.8; the latter has a large optimal region, where p takes values between 0.1 and 0.8 and N takes values between 100 and 300. $\mathcal{MM}$AS-(α β) has a large optimal region at the top triangle area between the three points (0.75, 10), (1, 5) and (3, 10); for $\mathcal{MM}$AS-(ρ m), the optimal region is relatively harder to locate at the corner where ρ takes values between 0.2 and 0.45 and m takes small values no more than 150.

The contour plots also give an indication of the relative influence of the parameters on algorithm performance. A typical example is $\mathcal{MM}$AS-(α β): most of the contour lines are parallel to the α-axis, which shows that the parameter β is more influential than α: as long as α takes a value greater than or equal to 1, the algorithm performance is much more sensitive to the variation of β. The same conclusion can be drawn in case study cPSO-(N p): as long as p takes a value between 0.1 and 0.8, the algorithm performance is more sensitive to the variation of parameter N, therefore N is more influential than p in this case study.

Finally, we also observed an interesting parameter correlation in case study cPSO-(χ ϕ_1), where the optimal region appears to be linearly correlated: the higher the acceleration factor ϕ_1, the smaller the constriction factor χ should be. In fact, here we fixed the parameter ϕ_2

Table 7 The best parameter configuration found by CMAES with post-selection and $nr = 5$, variation coefficient, and fitness distance correlation for each case study of $\mathcal{MM}$AS

	$(\alpha\ \beta)$	$(\rho\ m)$	$(\gamma\ nn)$
Best	(1 8)	(0.35 77)	(0.57 5)
VC/FDC	0.22/0.70	0.082/0.49	0.051/0.79
	$(\alpha\ \beta\ m)$	$(\beta\ \rho\ nn)$	$(\rho\ \gamma\ nn)$
Best	(1 7 180)	(9 0.6 17)	(0.42 4.8 7)
VC/FDC	0.20/0.74	0.37/0.60	0.13/0.62
	$(\alpha\ \beta\ \rho\ m)$	$(\alpha\ \beta\ \gamma\ nn)$	$(\rho\ m\ \gamma\ nn)$
Best	(1 9 0.49 210)	(1 8 3 22)	(0.6 150 2.6 5)
VC/FDC	0.20/0.57	0.41/0.52	0.21/0.79
	$(\alpha\ \beta\ \rho\ m\ nn)$	$(\alpha\ \beta\ \rho\ m\ \gamma)$	$(\alpha\ \beta\ m\ \gamma\ nn)$
Best	(1 9 0.77 180 17)	(1 8 0.52 260 0.35)	(1 7.7 270 2.2 17)
VC/FDC	0.44/0.50	0.21/0.53	0.44/0.54
	$(\alpha\ \beta\ \rho\ m\ \gamma\ nn)$	$(\alpha\ \beta\ \rho\ m\ \gamma\ q_0)$	$(\alpha\ \beta\ \rho\ m\ nn\ q_0)$
Best	(1.1 9 0.47 310 3.7 12)	(1 8.5 0.56 330 3.1 0.41)	(1 7 0.72 270 21 0.54)
VC/FDC	0.44/0.35	0.14/0.50	0.30/0.35

to 2.05; a linear correlation between χ and $\phi_1 + \phi_2$ can be observed from Fig. 11 and it is further discussed in Sect. 6.2.

6.2 The parameter landscape of all case studies

In the following, we extended our parameter landscape analysis to all 25 case studies from $\mathcal{MM}$AS and cPSO. To avoid the exponential growth of the number of grid points with the number of parameters, we uniformly at random sampled 4999 points in the parameter space. The 5000th point is the best parameter configuration out of the ten trials at the highest budget level of CMAES with post-selection and $nr = 5$. Each of the sampled points is evaluated on 25 instances (as in Sect. 6.1), and the average evaluation value over the 25 instances is computed.

In Tables 7 and 8 we show the best parameter configurations found by CMAES with post-selection and $nr = 5$ for problem class $\mathcal{MM}$AS and cPSO, respectively,[2] as well as the *variation coefficient* (VC) and the *fitness distance correlation* (FDC). The VC is a scale-invariant measure for the variability computed as the ratio of the standard deviation over the mean. High values of it often indicate that the algorithm performance is very sensitive to the setting of parameter values, and a careful tuning is important; low values often indicate lower sensitivity. The FDC (Jones and Forrest 1995) is the correlation coefficient between

[2]Note that the best parameter configuration found by CMAES in most of the case studies differs from the best one out of 5000 sampled points. Nevertheless, we list the former instead of the latter as the "best known" parameter configuration in Tables 7 and 8 because the former is selected from the tuning process based on many more than 25 instances.

Table 8 The best parameter configuration found by CMAES with post-selection and $nr = 5$, variation coefficient, and fitness distance correlation for each case study of cPSO

	(χ ϕ_1)	(ϕ_1 ϕ_2)	(N p)
Best	(0.53 3.8)	(3.3 1.6)	(180 0.2)
VC/FDC	0.29/0.52	0.40/0.47	0.29/0.68
	(χ ϕ_1 ϕ_2)	(ϕ_1 ϕ_2 N)	(χ N p)
Best	(0.68 0.63 4)	(0.43 3.8 10)	(0.78 14 0.84)
VC/FDC	0.32/0.37	0.39/0.11	0.45/0.18
	(χ ϕ_1 ϕ_2 N)	(χ ϕ_1 ϕ_2 p)	(ϕ_1 ϕ_2 N p)
Best	(0.66 1.9 3.7 13)	(0.67 3.7 1.7 0.76)	(0.54 3.8 6 0.59)
VC/FDC	0.38/0.18	0.36/0.17	0.32/0.46
	(χ ϕ_1 ϕ_2 N p)		
Best	(0.68 3.8 1.2 110 0.81)		
VC/FDC	0.41/0.072		

the fitness of the sampled points and their distance to the best points. It is usually used as a measure of the search difficulty of a problem. In minimization problems, a high positive FDC coefficient means that the lower the cost of a solution, the closer it is, on average, to the global optima. In the following, we discuss some conclusions from these results.

Observing the best parameter configurations in $\mathcal{MM}$AS, it is noteworthy that the best values for α and β usually take integer values (see Table 7). More specifically, α usually takes value 1 (with one exception where it is 1.1) and β takes values 7, 8, or 9 (with two exceptions where it takes values 7.7 and 8.5). In fact, there is a reason why α and β are usually integers: exponentiation by integers is typically handled by compilers as multiplications while exponentiation by non-integers uses a computationally much heavier Taylor series expansion. Since the term $\tau_{ij}^{\alpha} \cdot \eta_{ij}^{\beta}$ of (1) is computed very frequently, using integer values leads to a significant speedup of $\mathcal{MM}$AS, which, experimentally, can be up to 40%.

Noteworthy effects for other parameters are as follows. Firstly, the candidate list size parameter nn has a strong interaction with β: the correlation coefficient is 0.83. If β takes its default value 2, then nn takes a very small value; especially in case studies (γ nn) and (ρ m γ nn), it takes the value of its tuning lower bound, which is 5. This corresponds to a strong restriction of the size of the candidate list to compensate for the lack of greediness caused by small β. In other case studies, where β takes a large value, the best nn value increases to around its default value 20. The best values for parameter m appear to be much larger than the default setting 25.

Observing the best parameter configurations in cPSO from Table 8, the most remarkable insight is that there is a strong negative linear correlation between the best values of parameter χ and $\phi_1 + \phi_2$ as shown in Fig. 11 (correlation coefficient is -0.86). This means that the higher the sum of the acceleration factors, the lower the constriction factor should be.

Observing the VC values, in $\mathcal{MM}$AS, the parameter landscape is particularly variant when parameters β and nn are simultaneously tuned, which is indicated by the high VC value of these cases in Table 7. This means that the algorithm performance is very sensitive to the combination of β and nn. In general, we observed the tendency that the value of VC increases as the number of parameters increases, that is, the parameter landscape becomes more variant as more free parameters enter the tuning process. However, there are exceptions. In case studies (α β ρ m γ q_0) and (α β ρ m nn q_0), the introduction of parameter q_0

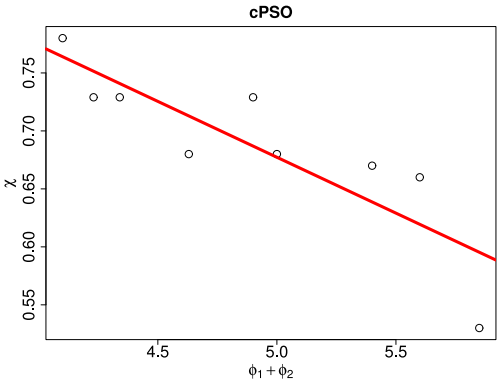

Fig. 11 Correlation plots between $\phi_1 + \phi_2$ and χ of cPSO. The correlation coefficient is -0.86

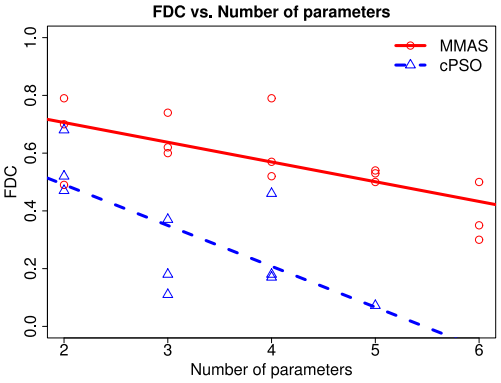

Fig. 12 Correlation plots between the FDC and the number of parameters in case studies of $\mathcal{MM}$AS and cPSO. The correlation coefficient is -0.70 for $\mathcal{MM}$AS, and -0.71 for cPSO

results in smaller VC than when fixing q_0 to 0. Our interpretation is that the introduction of ACS's pseudo-random proportional action choice rule into $\mathcal{MM}$AS by setting $q_0 > 0$ compensates the lack of exploitation if other parameters take poor values, making $\mathcal{MM}$AS's performance more robust. It is also worth noting that, in general, the VC in the case studies of cPSO is higher than in $\mathcal{MM}$AS, since nine out of 15 case studies of $\mathcal{MM}$AS have lower VC than the lowest value 0.29 in cPSO. This indicates a more variant parameter landscape for cPSO.

The FDC values, in general, as depicted in Fig. 12, decrease as the number of parameters increases. The correlation coefficient is -0.70 for $\mathcal{MM}$AS, and -0.71 for cPSO. This indicates that the tuning problems become more difficult with more parameters to be tuned. Moreover, the FDC values are lower in the cPSO case studies than in the $\mathcal{MM}$AS case studies (five out of 10 case studies in cPSO have lower FDC values than the lowest value 0.35 in $\mathcal{MM}$AS); this suggests that cPSO is more difficult to tune than $\mathcal{MM}$AS.

7 Conclusions

In this paper, we have studied a number of algorithms for tuning numerical parameters. We have compared three state-of-the-art algorithms for continuous optimization, CMAES, BOBYQA and MADS, together with uniform random sampling (URS) and the underlying sampling mechanism in iterated F-Race (IRS). These continuous optimization algorithms

are improved by a restart mechanism, and CMAES is extended with a uniform random sampling in the first iteration. All the above mentioned continuous optimization algorithms adopt a post-selection mechanism. The basic idea of applying the post-selection mechanism is to first identify a set of elite configurations through a small number of evaluations on each configuration, and then use F-Race in the final phase of the tuning process to carefully select best configuration among the set of elite configurations. Our experiments have proved that the post-selection mechanism is very effective in handling the stochastic nature of the tuning problem.

The experiments show that BOBYQA performs best in the case studies with two or three parameters to be tuned, but its performance degrades with more parameters. CMAES appears to be a rather robust algorithm across all numbers of parameters that we considered.

In future work, we plan to integrate continuous optimization algorithms with tuning algorithms that work on categorical parameters. The hybrid tuner that would result from this integration can be very effective to handle complex tuning problems. In addition, we will also analyze more thoroughly the proposed post-selection mechanism and we will explore the possibility of devising alternative ones.

Acknowledgements We thank Michael J.D. Powell for providing the Fortran code of BOBYQA. We also thank the anonymous referees for the detailed comments that helped to significantly improve the paper. This work was supported by the E-SWARM project, funded by an ERC Advanced Grant, and by the Meta-X project, 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

Adenso-Díaz, B., & Laguna, M. (2006). Fine-tuning of algorithms using fractional experimental designs and local search. *Operations Research*, *54*(1), 99–114.

Ansótegui, C., Sellmann, M., & Tierney, K. (2009). A gender-based genetic algorithm for the automatic configuration of solvers. In I. P. Gent (Ed.), *LNCS: Vol. 5732. Principles and practice of constraint programming—CP 2009* (pp. 142–157). Heidelberg: Springer.

Audet, C., & Dennis, J. E. (2006). Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on Optimization*, *17*(1), 188–217.

Auger, A., Hansen, N., Zerpa, J. M. P., Ros, R., & Schoenauer, M. (2009). Experimental comparisons of derivative free optimization algorithms. In J. Vahrenhold (Ed.), *LNCS: Vol. 5526. Experimental algorithms, 8th international symposium, SEA 2009* (pp. 3–15). Heidelberg: Springer.

Bartz-Beielstein, T. (2006). *Experimental research in evolutionary computation–the new experimentalism*. Berlin: Springer.

Birattari, M. (2009). *Tuning metaheuristics: a machine learning perspective*. Berlin: Springer.

Birattari, M., Stützle, T., Paquete, L., & Varrentrapp, K. (2002). A racing algorithm for configuring metaheuristics. In W. B. Langdon, et al. (Eds.), *GECCO 2002: proceedings of the genetic and evolutionary computation conference* (pp. 11–18). San Francisco: Morgan Kaufmann.

Birattari, M., Zlochin, M., & Dorigo, M. (2006). Towards a theory of practice in metaheuristics design: a machine learning perspective. *Theoretical Informatics and Applications*, *40*(2), 353–369.

Birattari, M., Yuan, Z., Balaprakash, P., & Stützle, T. (2010). F-Race and iterated F-Race: An overview. In T. Bartz-Beielstein, et al. (Eds.), *Experimental methods for the analysis of optimization algorithms* (pp. 311–336). Berlin: Springer.

Clerc, M. (2006). *Particle swarm optimization*. London: ISTE.

Clerc, M., & Kennedy, J. (2002). The particle swarm–explosion, stability, and convergence in a multidimensional complex space. *IEEE Transactions on Evolutionary Computation*, *6*(1), 58–73.

Dorigo, M. (2007). Ant colony optimization. *Scholarpedia*, *2*(3), 1461.

Dorigo, M., & Gambardella, L. (1997). Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, *1*(1), 53–66.

Dorigo, M., & Stützle, T. (2004). *Ant colony optimization*. Cambridge: MIT Press.

Dorigo, M., Maniezzo, V., & Colorni, A. (1996). Ant system: optimization by a colony of cooperating agents. *IEEE Transactions on Systems, Man and Cybernetics. Part B. Cybernetics*, *26*(1), 29–41.

Dorigo, M., Birattari, M., & Stutzle, T. (2006). Ant colony optimization. *IEEE Computational Intelligence Magazine*, *1*(4), 28–39.

Dorigo, M., Montes de Oca, M. A., & Engelbrecht, A. P. (2008). Particle swarm optimization. *Scholarpedia*, *3*(11), 1486.

Fukunaga, A. S. (2008). Automated discovery of local search heuristics for satisfiability testing. *Evolutionary Computation*, *16*(1), 31–61.

Hansen, N. (2006). The CMA evolution strategy: a comparing review. In J. Lozano, et al. (Eds.), *Studies in fuzziness and soft computing: Vol. 192. Towards a new evolutionary computation* (pp. 75–102). Berlin: Springer.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Murphy, K. P. (2009a). An experimental investigation of model-based parameter optimisation: SPO and beyond. In F. Rothlauf (Ed.), *Genetic and evolutionary computation conference, GECCO 2009* (pp. 271–278). New York: ACM Press.

Hutter, F., Hoos, H. H., Leyton-Brown, K., & Stützle, T. (2009b). ParamILS: an automatic algorithm configuration framework. *Journal of Artificial Intelligence Research*, *36*, 267–306.

Johnson, D. S., McGeoch, L. A., Rego, C., & Glover, F. (2001). 8th DIMACS implementation challenge. http://www.research.att.com/~dsj/chtsp/.

Jones, T., & Forrest, S. (1995). Fitness distance correlation as a measure of problem difficulty for genetic algorithms. In L. J. Eshelman (Ed.), *Proc. of the 6th international conference on genetic algorithms* (pp. 184–192). San Francisco: Morgan Kaufmann.

Kennedy, J., & Eberhart, R. (1995). Particle swarm optimization. In *Proc. of IEEE international conference on neural networks* (pp. 1942–1948). Piscataway: IEEE Press.

Kennedy, J., & Mendes, R. (2006). Neighborhood topologies in fully informed and best-of-neighborhood particle swarms. *IEEE Transactions on Systems, Man and Cybernetics. Part C, Applications and Reviews*, *36*(4), 515–519.

Kennedy, J., Eberhart, R., & Shi, Y. (2001). *Swarm intelligence*. San Francisco: Morgan Kaufmann.

Mengshoel, O. (2008). Understanding the role of noise in stochastic local search: analysis and experiments. *Artificial Intelligence*, *172*(8–9), 955–990.

Nannen, V., & Eiben, A. E. (2007). Relevance estimation and value calibration of evolutionary algorithm parameters. In *Proc. of IJCAI 2007* (pp. 975–980). Menlo Park: AAAI Press/IJCAI.

Oltean, M. (2005). Evolving evolutionary algorithms using linear genetic programming. *Evolutionary Computation*, *13*(3), 387–410.

Poli, R., Kennedy, J., & Blackwell, T. (2007). Particle swarm optimization. An overview. *Swarm Intelligence*, *1*(1), 33–57.

Powell, M. J. D. (2006). The NEWUOA software for unconstrained optimization. In *Nonconvex optimization and its applications: Vol. 83. Large-scale nonlinear optimization* (pp. 255–297). Berlin: Springer.

Powell, M. J. D. (2009). *The BOBYQA algorithm for bound constrained optimization without derivatives* (Technical Report NA2009/06). Department of Applied Mathematics and Theoretical Physics, University of Cambridge, Cambridge, UK.

Steinmann, O., Strohmaier, A., & Stützle, T. (1997). Tabu search vs. random walk. In G. Brewka, C. Habel, & B. Nebel (Eds.), *LNAI: Vol. 1303. KI-97: advances in artificial intelligence* (pp. 337–348). Heidelberg: Springer.

Stützle, T. (1999). *DISKI: Vol. 220. Local search algorithms for combinatorial problems: analysis, improvements, and new applications*. Sankt Augustin: Infix.

Stützle, T. (2002). Software ACOTSP. http://iridia.ulb.ac.be/~mdorigo/ACO/aco-code/public-software.html.

Stützle, T., & Hoos, H. H. (1998). MAX-MIN ant system and local search for combinatorial optimization problems: Towards adaptive tools for combinatorial global optimization. In S. Voss, et al. (Eds.), *Metaheuristics: advances and trends in local search paradigms for optimization* (pp. 313–329). Dordrecht: Kluwer Academic.

Stützle, T., & Hoos, H. (2000). $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generations Computer Systems*, *16*(8), 889–914.

Torczon, V. (1997). On the convergence of pattern search algorithms. *SIAM Journal on Optimization*, *7*(1), 1–25.

Yuan, Z., Montes de Oca, M., Birattari, M., & Stützle, T. (2010a). Modern continuous optimization algorithms for tuning real and integer algorithm parameters. In M. Dorigo, et al. (Eds.), *LNCS: Vol. 6234. Proceedings of ANTS 2010, the seventh international conference on swarm intelligence* (pp. 204–215). Heidelberg: Springer.

Yuan, Z., Stützle, T., & Birattari, M. (2010b). MADS/F-Race: mesh adaptive direct search meets F-race. In M. Ali, et al. (Eds.), *LNAI: Vol. 6096. Proceedings of IEA-AIE 2010* (pp. 41–50). Heidelberg: Springer.

Zlochin, M., Birattari, M., Meuleau, N., & Dorigo, M. (2004). Model-based search for combinatorial optimization: a critical survey. *Annals of Operations Research*, *131*(1–4), 373–395.