Applied Soft Computing 17 (2014) 39–51

Contents lists available at ScienceDirect

Applied Soft Computing

journal homepage: www.elsevier.com/locate/asoc

A beginner's guide to tuning methods☆

Elizabeth Montero[a,b,*], María-Cristina Riff[a], Bertrand Neveu[c]

[a] Universidad Técnica Federico Santa María, Valparaíso, Chile
[b] Université de Nice, Sophia Antipolis, France
[c] Project Imagine, École des Ponts ParisTech, Paris, France

ARTICLE INFO

Article history:
Received 12 June 2011
Received in revised form 1 April 2012
Accepted 19 December 2013
Available online 3 January 2014

Keywords:
Metaheuristics
Evolutionary algorithms
Tuning methods
Parameter setting problem

ABSTRACT

Metaheuristic methods have been demonstrated to be efficient tools to solve hard optimization problems. Most metaheuristics define a set of parameters that must be tuned. A good setup of that parameter values can lead to take advantage of the metaheuristic capabilities to solve the problem at hand. Tuning strategies are step by step methods based on multiple runs of the metaheuristic algorithm. In this study we compare four automated tuning methods: F-Race, Revac, ParamILS and SPO. We evaluate the performance of each method using a standard genetic algorithm for continuous function optimization. We discuss about the requirements of each method, the resources used and quality of solutions found in different scenarios. Finally we establish some guidelines that can help to choose the more appropriate tuning procedure.

1. Introduction

Performance of metaheuristic methods is strongly related to the definition of proper components and a suitable setting of parameter values. In this sense, we distinguish between two sets of parameters:

- Categorical parameters: They are procedures or functions of the algorithm which can be implemented in different ways. For example, a categorical parameter can be the selection method to use in an evolutionary algorithm. In this case the choice could be made among: roulette wheel, tournament or ranking.
- Numerical parameters: They are real or integer values. Examples of numeric parameters are population size and transformation operators rates in evolutionary algorithms.

The main difference between categorical and numeric parameters is that numeric parameters define *searchable spaces*, i.e. it is possible to define distance measures between two values of the parameter. For categorical parameters it is not possible to measure distance between two values, for example when we compare the roulette wheel with 2-tournament selection procedures.

Parameter setting problem is an important problem when we use metaheuristics. The selection of a suitable set of values for the parameters can lead to a very good performing version of the metaheuristic at hand. The main difficulties related to the parameter setting problem can be summarized as:

- Time consuming task: Parameter setting problem is a time consuming process which usually requires many runs of the metaheuristic with different problem instances and seeds.
- Best parameter values set depends on problem at hand: It is reasonable to set the same parameter values for all instances of a problem? Different instances of the same problem can strongly vary in terms of size and topology of their search spaces.
- Parameters are interrelated: Usually, the behavior of the metaheuristics is related to unknown and complex non-linear interactions among the parameters.

Main features of parameter setting problem turn it into an optimization problem itself.

It is important to note that the stochastic nature of metaheuristics is a key factor to consider when the parameters are being set. Different executions of a metaheuristic using the same parameter values but different seeds can lead to different performances of the metaheuristic algorithm.

In [11] the difference between *tuning* and *control* parameter setting methods is defined. Main features of parameter tuning

☆ Supported by Fondecyt project 1120781 and Postdoctoral Fondecyt project 3130754.
* Corresponding author at: Universidad Técnica Federico Santa María, Valparaíso, Chile.
E-mail addresses: Elizabeth.Montero@inf.utfsm.cl, elizabeth.montero@gmail.com (E. Montero), Maria-Cristina.Riff@acm.org (M.-C. Riff), NeveuB@certis.enpc.fr (B. Neveu).

1568-4946/$ – see front matter © 2013 Elsevier B.V. All rights reserved.
http://dx.doi.org/10.1016/j.asoc.2013.12.017

methods can be summarized as: (1) Those processes are performed before the run of the tuned algorithm and (2) those processes search for parameter values, which will remain fixed during the run of the algorithm. Moreover, tuning processes usually involve multiple runs of the algorithm in order to analyze its performance with different parameter values and considering its stochastic nature. This makes tuning processes high consuming time tasks. On the other hand, *control* methods are processes performed during the execution of the algorithm that allow it to change the parameter values during the run. Control methods are included in the original algorithm possibly modifying its original design and increasing its execution time.

1.1. Parameter tuning

The first approach to perform this process can be defined as a *Brute-force* approach. It consists in determining the best parameter values estimating the performance of each possible parameter configuration (a *full factorial* set) by means of a sufficiently large number of executions and a sufficiently large set of problem instances. In literature, different tuning methods have been proposed. Tuning methods can be classified as:

- **Hand-made tuning**: In this case the metaheuristic algorithm's designer studies the performance of the algorithm in a set of problem instances using appropriate parameter values for the algorithm (according to his own criteria). These parameter values are iteratively modified in the way of achieving improvements in the performance of the algorithm. Hand-made tuning is the typical approach to parameter tuning in the oldest research works in metaheuristic area [11].
- **Tuning by analogy**: In this case, the idea is to follow guidelines that recognized authors have established according to the studies they have performed. Some well known parameter settings were proposed by De Jong [10] and Grefenstette [12]. Common guidelines for evolutionary algorithms are to use a mutation probability of $1/L$, where L is the length of the chromosome and a crossover probability of 0.6 [10].
- **Experimental design based tuning**: We consider as experimental based tuning, those studies based on experimental design to set the parameter values. In [9], the authors perform a statistical exploratory analysis to model the effect and relations between the crossover and mutation probabilities on a standard genetic algorithm.

  Racing methods can also be considered in this category. Racing methods search for good configurations, starting by an initial set of parameter configurations and, iteratively, discarding the worst performing according to statistical tests [7,6,8].

  Sequential Parameter Optimization method [2,15,5] corresponds to a special case of the well known sequential model-based optimization methods like the efficient global optimization (EGO) algorithm [13] and the sequential *kriging* optimization procedure [17]. SPO has been specially adapted to deal with stochastic nature response surfaces. SPO method is a three steps method. First, it performs an experimental analysis of a set of design points (parameter configurations); second, a stochastic process model is used for estimating the algorithm performance; and, finally, it determines new promising design points according to estimations of the model.
- **Search based tuning**: One of the first approaches to search based tuning methods is the meta-level genetic algorithm (*meta-GA*) proposed in [12]. It is itself a standard genetic algorithm as defined in [10]. Parameter values of meta-GA were tuned *by analogy* according to the De Jong guidelines described in [10].

  Revac algorithm [21] is an evolutionary algorithm whose operators are used to estimate parameter's distributions. Revac works with a population of parameter calibrations, at each step, it generates a new calibration using mutation and crossover operations. Revac operators are able to search in the whole range of values previously defined for each operator.

  ParamILS is an iterated local search strategy [14,16]. It starts with a default parameter configuration and iteratively tries to improve its performance by searching in its neighborhood for better parameter calibrations. The Focused ParamILS version implements a discrimination method able to perform robust comparisons between parameter configurations.
- **Hybrid tuning**: In [1] the authors propose *Calibra* method, this method combines experimental design and local search concepts. Calibra uses the Design of Experiments to focus the search promising areas of the search space and local search to vary the parameter values found at each step on the promising areas. Calibra method is able to tune less than five parameters, because it uses the Taguchi's $L_9(3^4)$ design.

In our study we compare four tuning methods: F-Race that corresponds to an adaptation of Racing methods; Revac; ParamILS and SPO. All of these methods are described deeply in Section 2. We describe our experiments in a genetic algorithm for continuous optimization in Section 3. Finally, general conclusions about this work are presented in Section 4 including future perspectives of this work. This study was presented in the thesis work in [18].

2. Tuning methods

Tuning methods are strategies designed to automatically search for the best configuration for parameter values in metaheuristics methods. Given a metaheuristic with k parameters, tuning methods search for a parameter configuration $c^* = \{p_1, \ldots, p_k\}$ that leads the best performance of the tuned algorithm.

2.1. F-Race method

F-Race was proposed in [7] based on the general idea of *racing* methods. Racing methods are based on brute-force approach. The idea is to provide a better allocation of resources among candidate parameter configurations, reducing the computational resources allocated to poor configurations. F-Race is an iterative process, at each step it evaluates a set of candidate configurations in a new problem instance or seed and eliminates from the set those calibrations that show a statistical worse performance. Fig. 1 shows 4 steps of a F-Race process. In the first step, calibrations have not been evaluated, so they are considered to have the same performance. In race i, calibrations white and gray outperform calibration black. At step $i+1$, calibrations white and gray are clearly better than black calibration and, hence, black calibration is eliminated as it can be seen in step $i+2$. The idea of discarding poor parameter configurations is to perform more evaluations of the promising configurations on more instances and, hence, obtain a more reliable estimation of their performance.

F-Race is a specific racing method specially adapted to tune stochastic search algorithms. It uses *Friedman two-way analysis of variance by ranks* to compare the performance among the entire set of candidate parameter configurations. This is a non-parametric test based on ranking; thus, hypotheses on the distribution of the observations are not required. F-Race method implements a block design, which considers as sources of variation the different problem instances. The null hypothesis of this test is formulated as: "all possible rankings of the candidates within each block are equally

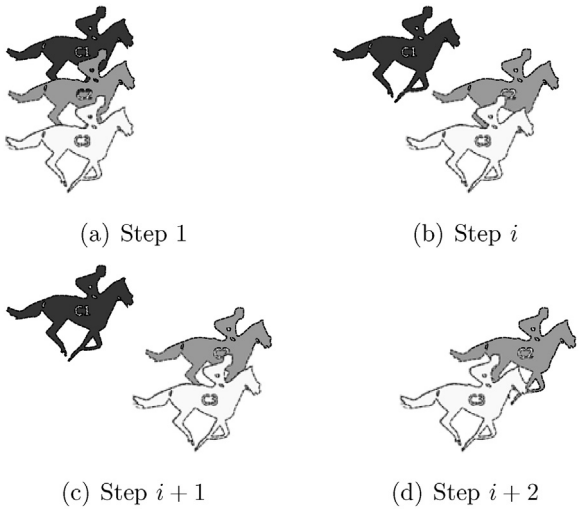


Fig. 1. F-Race process: (a) At the first step all racers are equal. (b) As the race progresses some racers show a better performance than others. Only when this difference is clearly noticeable (c) worst racers can be discarded and the race continues without them (d).

likely". This hypothesis is contrasted using the Friedman test which considers the statistic shown in Eq. (1).

$$T = \frac{(n-1)\sum_{j=1}^{n}(R_j - ((m(n+1))/2))^2}{\sum_{l=1}^{m}\sum_{j=1}^{n}R_{ij}^2 - (mn(n+1)^2)/4} \quad (1)$$

where m is the step of the race with n remaining candidates parameter configurations, R_{lj} is the rank of parameter configuration c_j within block l and $R_j = \sum_{l=1}^{m} R_{lj}$. The statistic T is χ distributed with $n-1$ degrees of freedom. If the observed T exceeds the $(1-\alpha)$ quartile of such a distribution, the null hypothesis is refused at the approximate level α. In this case, the hypothesis that at least one candidate tends to yield a better performance than at least one other is accepted.

A general description of F-Race is shown in Algorithm 1. The procedure starts with a set C of parameter configurations. F-Race requires the definition of a discrete set of values for each parameter to tune. In the first version of F-Race, this initial set includes all possible combinations of discrete sets of parameter values (full factorial set).

At the beginning of the process, F-Race performs r runs for each parameter configuration in C to obtain enough information before any elimination. The information is stored in the array of costs of each parameter configuration ($Cost^r(c_j)$).

At each step, F-Race selects randomly an instance to go forward to the performance comparisons among candidate parameter configurations in C. F-Race evaluates the tuned algorithm on the instance and appends this information to the array of costs of each parameter configuration. When all the arrays of costs are computed, F-Race performs the Friedman test according to Eq. (1). If the null hypothesis is rejected, it can be interpreted as at least one candidate configuration tends to show a better performance than at least other configuration. In this case pairwise comparisons between parameter configurations and the best ranked parameter configuration are performed. The parameter configurations with a statistical lower performance are eliminated from the set of candidates configurations C. The block design used by F-Race allows the normalization of the performances observed for each instance and the use of this statistic. A significance level α should be set for the tests. F-Race process stops either when there is only one parameter configuration remaining, or when some predefined amount of executions is reached.

Algorithm 1. F-Race algorithm.

```
Procedure F-Race
generate set C of candidate configurations
race ← 1
while not terminationcriterion() do
  select randomly instance i from set of instances I
  foreach c_j ∈ C do
    append the cost of executing c_j in i to Cost(c_j)
  endfor
  foreach c_j ∈ C do
    foreach instance l executed do
      R_lj ← ranking of configuration c_j in block l
    endfor
    R_j ← sum of ranks over all instances of configuration c_j
  endfor
  if race > r then
    perform the Friedman test according to Eq. (1)
    if null hypothesis associated to T is rejected then
      c_best ← configuration in C with minimum R
      foreach c_j ∈ C \ c_best do
        perform a pairwise test
        if null hypothesis is rejected then C ← C \ c_j
      endfor
    endif
  endif
  race ← race + 1
end while
return remaining set C
```

F-Race method defines three parameters: the amount of races without elimination of candidate configurations (r), the confidence level of hypothesis tests (α) and the maximum number of executions of the tuned algorithm (budget). These parameters and their typical values are shown in Table 1. F-Race also requires the determination of the discrete levels for each parameter to tune. The amount of levels of all parameters will determine the size of the initial set of candidate parameter configurations.

2.2. Relevance Estimation and Value Calibration method

The Relevance Estimation and Value Calibration of evolutionary algorithms method was proposed in [21]. Revac is defined as an estimation of distribution algorithm [22]. It works with a set of parameter configurations, a population. Fig. 2 shows the representation of Revac population. Each row represents a parameter configuration and each row has k elements, corresponding to the k parameters to tune. Each column represents the distribution of values of one parameter in population, each column has M elements corresponding to the size of the set of configurations.

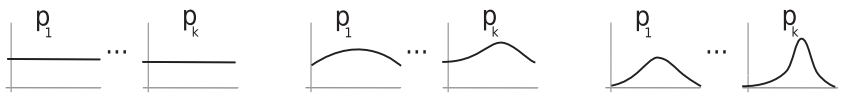
For each parameter, Revac starts the search process with an uniform distribution of values within a given range as it can be seen in Fig. 3. At each step, Revac reduces the range of values of each

Table 1
F-Race parameters.

Parameter	Description	Value
r	Initial races	7
α	Confidence level	0.05
max_execs	Maximum number of executions	1000

$$\left\{\begin{array}{c|cccc} c_1 & p_{1,1} & p_{2,1} & \cdots & p_{k,1} \\ c_2 & p_{1,2} & p_{2,2} & \cdots & p_{k,2} \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ \cdots & \cdots & \cdots & \cdots & \cdots \\ c_M & p_{1,M} & p_{2,M} & \cdots & p_{k,M} \end{array}\right\}$$

Fig. 2. Revac population structure.



(a) Step 1 (b) Step i (c) Final step

Fig. 3. Revac process: (a) At the first step all parameter distributions are uniform. (b) As the search progresses some parameter distributions concentrate around the set of best values for each parameter. (c) Final distributions show peaks in the range of best performing values for each parameter. Narrowness of peaks is considered as an indicator of importance of parameter value on the performance of the algorithm. In this case p_k is considered more important than p_1.

Table 2
Revac parameters.

Parameter	Description	Value
M	Population size	100
N	Crossover operator size	50
$H = N/10$	Mutation operator size	5
max_execs	Maximum number of executions	1000

parameter by using specially designed transformation operators (see Fig. 3).

In Revac, the relevance of each parameter is determined according to the entropy measurement [21]. A parameter that shows a low entropy is considered very useful for the performance of the algorithm, because the best performance was obtained when the value of the parameter was fixed at a small possible set of values. Analogously, a parameter that shows a high entropy value will be classified as a not important one for the performance of the algorithm.

Revac procedure is shown in Algorithm 2. The algorithm starts with a random population of M parameter configurations.

Each parameter configuration is then evaluated. Revac does not define a specialized mechanism to evaluate a parameter configuration quality among a set of instances. Evaluation process considers the execution of the algorithm in the problem instance using just one random seed. Replication is not required as shown in [20].

At each iteration, only one new parameter configuration is created and the child configuration is created through a multi-parent crossover and a mutation transformation. The multi-parent crossover considers the best $N < M$ parameter configurations with uniform scanning. Mutation operator calculates, for each parameter, a mutation interval. For this purpose, all different parameter values present in the population are ordered. The upper and lower interval bounds correspond respectively to the Hth lowest neighbor of the value of parameter i in the child and the Hth highest neighbor of the value of the parameter i in the child. Then a random value is selected from the mutation interval and assigned to the child configuration.

Revac process ends after 1000 parameter configuration executions.

Algorithm 2. Revac procedure.

Procedure Revac
$C_p \leftarrow$ generate M random parameter configurations
evaluate each parameter configuration $\in C_p$
while not max_execs **do**
  $c_{child} \leftarrow$ uniform crossover of N best parameter configurations
  $c_{child} \leftarrow$ mutate c_{child} on interval $2 \cdot H$
  evaluate configuration c_{child}
  replace the oldest parameter configuration $\in C_p$ with c_{child}
  calculate entropy of each parameter $\in C_p$
end while
return range of values and entropy of each parameter $\in C_p$

Revac method defines three parameters: the size of the population of parameter configurations (M), the size of the crossover and mutation operators (N and H, respectively) and the maximum number of executions of the tuned algorithm. These parameters and their values are shown in Table 2.

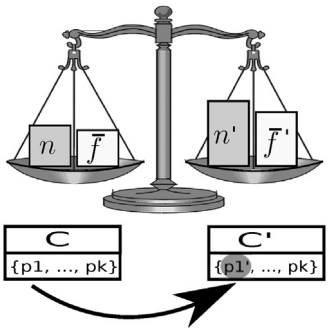


Fig. 4. FocusedILS comparison process: c and c' calibrations are considered equivalent unless that one of them is both better in fitness and has been tested in a higher number of seeds. Considering a maximization problem, in this case c will be replaced by c'.

2.3. Parameter Iterated Local Search method

The ParamILS method was proposed in [14]. It works as an iterated local search algorithm and was inspired by the *hand-made* tuning process. ParamILS starts with a default parameter configuration and iteratively improves its performance searching in its neighborhood. The neighborhood is defined as the change of the value of just one parameter. Fig. 4 outlines the process performed at each iteration. Comparison between parameter configurations are based on both the performance of the parameter configurations and the number of seeds used to calculate their fitness.

ParamILS process is shown in Algorithm 3. The algorithm starts with a default parameter configuration, usually based on the experience of the user. Then, the method performs R tries searching for a new configuration with a better quality than the default parameter configuration. This new configuration found follows a local search procedure.

At each iteration, ParamILS performs s random perturbations to the configuration at hand, then a local search process ($IterativeFirstImprovement(c, N)$) is performed using the best configuration obtained and then it compares its performance to the best parameter configuration found so far. There is also a restart probability ($p_{restart}$), which allows the algorithm to escape from local optimum.

Algorithm 3. ParamILS.

Procedure ParamILS
$c_0 \leftarrow$ default parameter vector
for $i \leftarrow 1$ to R **do**
  $c \leftarrow$ random parameter vector
  if better(c, c_0) **then** $c_0 \leftarrow c$
endfor
$c_{ils} \leftarrow IterativeFirstImprovement(c_0, \mathcal{N})$
while not $terminationcriterion()$ **do**
  $c \leftarrow c_{ils}$
  for $i \leftarrow 1$ to s **do**
    $c \leftarrow$ random parameter vector in $\mathcal{N}(c)$
  $c \leftarrow IterativeFirstImprovement(c, \mathcal{N})$
  if better(c, c_{ils}) **then** $c_{ils} \leftarrow c$
  if $p_{restart}$ **then** $c_{ils} \leftarrow$ random parameter vector
endwhile
return overall best c

Table 3
ParamILS parameters.

Parameter	Description	Value
R	Random solutions in first phase	10
s	Random solutions at each iteration	3
$p_{restart}$	Probability of restarting the search	0.01
max_execs	Maximum number of executions	1000

The *IterativeFirstImprovement*(c, N) process used by ParamILS is a procedure which searches randomly in the neighborhood of a parameter configuration for a *better* one. This procedure is shown in Algorithm 4.

Algorithm 4. ParamILS *IterativeFirstImprovement* procedure.

```
Procedure IterativeFirstImprovement(c, N)
repeat
  c' ← c
  foreach c'' ∈ N(c') in randomized order do
      if better(c'', c') then c ← c''
      break
until c' = c
return c
```

Alternative ParamILS algorithm versions are: BasicILS and FocusedILS. These versions differ in the *better*(c, c') procedure they implement. BasicILS considers a fixed amount N of seeds to compare the performance of configurations c and c' while the *better*(c, c') procedure implemented by FocusedILS works defining the dominance concept. In this case, it is possible to say that c dominates c', if and only if, the average performance of the algorithm using c for solving n seeds is *better* than the performance obtained using c' configuration for solving n' seeds, with $n > n'$.

ParamILS method defines four parameters; these parameters and their values are shown in Table 3. The use of ParamILS requires the definition of the interval levels for each parameter and an initial parameter configuration.

2.4. Sequential Parameter Optimization method

The Sequential Parameter Optimization method is based on the combination of Design of Experiments (DOE) and Design and Analysis of Computer Experiments (DACE), coupled with specially designed techniques to tackle randomness of stochastic nature algorithms. The combination of DOE and DACE has been extensively used and the first approaches to use them in stochastic algorithms were performed in [4,3]. The term *sequential parameter optimization* was introduced in [2].

The method can be defined as a search heuristic to optimize the performance of stochastic algorithms. SPO is a method that iteratively performs an experimental analysis of a set of design points (parameter configurations), then it estimates the performance of the algorithm to tune by means of a stochastic process model and finally determines new design points.

SPO is shown in Algorithm 5. It starts constructing a set of initial design points through a Latin Hypercube Sampling (LHS) design over the range of parameter values defined. The LHS design is used because it provides a good coverage of the design space. To determine n design points, the range of each parameter must be divided into n equally sized intervals. Then one parameter value is randomly chosen for each range defined in order to obtain n possible values.

Due to the stochastic nature of the search algorithms considered here, performance for each design point is evaluated by means of several repeats. The best design points from the previous iteration are included in the set of points of the current iteration and reevaluated, thereby increasing its number of repeats. SPO enforces fair comparison of the current best design points; the new generated design points are executed as many times as the best design point has been executed.

Table 4
SPO parameters.

Parameter	Description	Value
n	Initial design size	10
$samples$	Initial samples	2
cp	Number of candidates design points	2
r	Rate of old and new design points	1/3
max_execs	Maximum number of executions	1000

Table 5
Numeric parameters of SGA.

Parameter	Values
Crossover rate (c_r)	$c_r \in \mathbb{R}$, where $0.0 \leq c_r \leq 1.0$
Mutation rate (m_r)	$m_r \in \mathbb{R}$, where $0.0 \leq m_r \leq 0.25$
Population size (p_s)	$p_s \in \mathbb{N}$, where $5 \leq p_s \leq 100$

The set of design points is used to estimate the algorithm performance by means of a stochastic process model defined by Eq. (2).

$$Y(x) = \sum_{j=1}^{k} \beta_j \cdot f_j(x) + Z(x), \tag{2}$$

where $Z(\cdot)$ is a random process with mean zero and covariance $V(u, v) = \sigma^2 \mathcal{R}(c, u, v)$ and using the correlation shown in Eq. (3).

$$\mathcal{R}(c, u, v) = \prod_{j=1}^{d} \exp(-c_j \cdot (u_j - v_j)^2). \tag{3}$$

The set of design points of the next iterations includes the best points found so far and a set of expected good designs, whereby expectation is based on the model created in previous step. For this purpose, a new set of (cp) candidate points are also created using the LHS design. This set is typically much larger than the initial set of design points. The generalized expected improvement criterion is determined computing the candidates model value. This criterion takes into account that we are uncertain about unknown design points. Thus, it estimates the probability of a candidate of being better than the known best so far, by taking the modeling error into account. Therefore, the points with the highest probability of being better than the current best design point are selected as candidates for the next iteration.

Algorithm 5. Sequential Parameter Optimization procedure.

```
Procedure Sequential Parameter Optimization
C ← n LHS designed configurations
while not terminationcriterion() do
  for each i ∈ C do
    run c_i samples times
    Cost(c_i) ← average performance of samples
  construct model M according to Cost(C) using (2) and (3)
  C_old ← select r·n best configurations from C
  C_candidates ← cp LHS designed configurations
  C_new ← select (1 − r)·n best promising configurations
      from C_candidates according to model M
  C ← C_old + C_new
  updatesamplesize
endwhile
return overall best c ∈ C
```

Table 4 shows the four parameters defined by SPO and their typical values. The method requires the definition of a range of values for each parameter to tune.

Table 6
Test suite.

Function	Formulation	Interval
f_1: Sphere	$\sum_{i=1}^{3} {x_i}^2$,	$-5.12 \le x_i \le 5.12$
f_2: Rosenbrock	$100 \cdot ({x_2} - {x_1}^2)^2 + (1 - x_1)^2$,	$-2.048 \le x_i \le 2.048$
f_3: Step	$30 + \sum_{i=1}^{5} \lfloor x_i \rfloor$,	$-5.12 \le x_i \le 5.12$
f_4: Schaffer	$0.5 + \frac{(\sin(\sqrt{x_1^2 + x_2^2}))^2 - 0.5}{(1 + 0.0001 \cdot (x_1^2 + x_2^2))^2}$,	$-100 \le x_i \le 100$
f_5: Rastrigin	$10 \cdot D + \sum_{i=1}^{D} ({x_i}^2 - 10 \cdot \cos(2 \cdot \pi \cdot x_i))$,	$-5.12 \le x_i \le 5.12$
f_6: Schwefel	$418.9829 \cdot D + \sum_{i=1}^{D} -1 \cdot x_i \cdot \sin(\sqrt{\lvert x_i \rvert})$,	$-500 \le x_i \le 500$
f_7: Griewank	$\sum_{i=1}^{D} \frac{x_i^2}{4000} - \prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + 1$,	$-100 \le x_i \le 100$
f_8: Ackley	$-20 \cdot \exp\left(-0.2 \cdot \sqrt{\frac{1}{D} \cdot \sum_{i=1}^{D} x_i^2}\right) - \exp\left(\frac{1}{D} \sum_{i=1}^{D} \cos(2 \cdot \pi \cdot x_i)\right) + 20 + e$,	$-32 \le x_i \le 32$

3. Experiments

In this section we compare the results of the four tuning methods and the BRS algorithm. Here we are interested in analyzing their advantages and disadvantages in terms of quality of results, effort required, information delivered, usability, user-friendliness and behavior on different scenarios.

For our experiments we used the Frank Hutter implementation of ParamILS available in his website,[1] the SPOT implementation of SPO method available in the SPOT website,[2] the Alvaro Fialho implementation of F-Race and our own REVAC implementation, last both available in our website.[3] We selected a standard genetic algorithm for continuous function optimization as the algorithm to tune. This source code is also available in our website.[3]

3.1. Standard genetic algorithm for continuous function optimization

We consider a standard genetic algorithm (SGA) with Gray code representation. At each iteration an entire new population is constructed according to a selection, crossover and mutation operators. Elitism is always used. Parameters associated to the SGA are listed in Table 5. SGA uses roulette wheel as selection operator, single-point crossover operator and bit-flip mutation operator.

The SGA is used to optimize the functions presented in Table 6. First three functions are classical test functions included in De Jong's test suite. Here we have functions Sphere, Rosembrock and Step. We also considered Schaffer's f_6 function (f_4 in this paper). Moreover, considering the unimodal feature of these functions we included in our tests suite four multimodal functions: Rastrigin, Schwefel, Griewank and Ackley functions ($f_5, \ldots, f_8$). In this case, D corresponds to the number of dimensions. In our tests we fixed D to 5 dimensions for functions f_5, f_6 and f_8 and set to 2 for function f_7. This is because previous tests show that the time required by SGA to reach the optimum for f_7 was too high and we wanted to keep this time low considering the cost of tuning processes.

The fitness criterion of a SGA execution is the amount of evaluations to find the optimum. For functions f_1, f_2 and f_3 the optimum value was fixed at 10^{-10} and for function f_4 the optimum value was fixed at 10^{-6}. For functions f_5 and f_8 we consider an optimum value of 10^{-4} and for functions f_6 and f_7 the optimum was fixed at 10^{-3}. Again, these values were fixed considering the hardness of the function being solved and, specially, keeping low the effort (time) to solve it.

Each execution of the SGA performs a maximum amount of 10^5 function evaluations. According to previous experimentation this amount of evaluations performs well for the test suite considered, i.e. these amount of evaluations allow the SGA find the optimum in most cases for most functions. Moreover, these amounts of evaluations needed around to 1.3 s for each SGA execution. When the optimum cannot be reached the fitness of the SGA execution is set to the maximum amount of function evaluations fixed.

3.2. Experimental setup

We have organized our experiments in three different scenarios. Each scenario is formed by a set of training instances and a set of testing instances. Training instances are used during the tuning process in order to measure the quality of the parameter calibrations the tuning method evaluates. Testing instances are used to measure the quality of the configurations found by the tuning methods using the training instances. Disjoint training and testing sets have been considered in order to avoid over-fitting effects. Table 7 shows the training set and testing sets considered in each scenarios.

It is important to note that F-Race and ParamILS include in their formulations specific directions to tune algorithms considering a set of instances in the training set. Revac and SPO were all adapted considering the same criteria used by ParamILS to use the entire training set during the tuning process. In all these cases, the average performance of the parameter calibration in the training set is considered the quality of each parameter configuration.

For our experiments we have considered a maximum budget of 4000 executions of the SGA for each tuning method. This maximum budget was fixed according to the Revac specifications detailed in Section 2.2. Revac considers 1000 parameter calibration evaluations. Moreover, each evaluation of a parameter configuration requires, in this case, four SGA executions, one with each problem instance considered in the training set. Hence, the F-Race, ParamILS and SPO methods were able to perform a maximum of 4000

Table 7
Testing scenarios.

	Training set	Testing set
Scenario 1	$\{f_1, f_2, f_3, f_4\}$	$\{f_5, f_6, f_7, f_8\}$
Scenario 2	$\{f_5, f_6, f_7, f_8\}$	$\{f_1, f_2, f_3, f_4\}$
Scenario 3	$\{f_1, f_2, f_5, f_6\}$	$\{f_3, f_4, f_7, f_8\}$

[1] http://www.cs.ubc.ca/labs/beta/Projects/ParamILS.
[2] http://cran.r-project.org/web/packages/SPOT/index.html.
[3] http://www.inf.utfsm.cl/~emontero/tuners.

Table 8
Search space for numeric parameters.

Parameter	Values	Set size
F-Race		
$crossover_{rate}$	{0.0, 0.25, 0.5, 0.75, 1.0}	5
$mutation_{rate}$	{0.00, 0.05, 0.1, 0.15, 0.20, 0.25}	6
$population_{size}$	{5, 10, 25, 50, 100}	5
Revac		
$crossover_{rate}$	[0.0, 1.0]	Undefined
$mutation_{rate}$	[0.0, 0.25]	Undefined
$population_{size}$	[5, 100]	Undefined
ParamILS		
$crossover_{rate}$	{0.0, 0.10, 0.20, 0.30, 0.40, 0.50, 0.60, 0.70, 0.80, 0.90, 1.00}[0.50]	11
$mutation_{rate}$	{0.0, 0.01, 0.03, 0.05, 0.07, 0.09, 0.11, 0.15, 0.19, 0.23, 0.25}[0.11]	11
$population_{size}$	{5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 100}[50]	11
SPO		
$crossover_{rate}$	[0.0, 1.0]	Undefined
$mutation_{rate}$	[0.0, 0.25]	Undefined
$population_{size}$	[5, 100]	Undefined

executions of the SGA in order to establish a fair comparison between tuning methods.

Revac method used in this study has been designed to work in searchable spaces; hence it is not able to tune categorical parameters. Therefore, we compare the methods tuning only numeric parameters of the SGA.

When using F-Race and ParamILS methods to tune numeric parameters it is necessary to define discrete sets of values for each one. For F-Race, the range levels considered were determined by considering a trade-off between the initial amount of races performed by F-Race (r) and the size of initial the set of candidate configurations. Moreover, we tried to maximize the levels points considered for each parameter. In this case, we have considered between 5 and 6 possible values for each numeric parameter and $r = 5$ initial races without elimination of candidate configurations. For ParamILS, we have considered a maximum of 11 level values for each numeric parameter. This maximum was established to control the neighborhood size of the local search operator. The range levels used for each tuning method are shown in Table 8.

For comparison, we have also implemented a *Blind Random Search (BRS)* method. It is an iterative process, which at each step generates a random parameter calibration by selecting each parameter value randomly from its range of possible values. The random calibration is executed using a random seed and its performance is compared with the best performance found so far. The output corresponds to the best performing calibration found.

The four tuning methods and the BRS algorithm were executed 5 times considering the stochastic nature of both the SGA and the tuning methods.

Moreover, the performance of each parameter configuration found is measured after executing the SGA algorithm 100 times using different random seeds. Performance of each execution is measured as the number of function evaluations required to reach the optimum considering the precision described in Section 3.1.

3.3. Scenario 1

This scenario considers as training set four unimodal functions: $\{f_1, f_2, f_3$ and $f_4\}$ and its testing set is conformed by four multimodal functions: $\{f_5, f_6, f_7$ and $f_8\}$. Table 9 shows the final parameter configurations found by each tuning method for Scenario 1. F-Race was unable to discard parameter configurations during its process $F3$.

In this scenario F-Race found twice the same parameter calibration $c = \{crossover_{rate} = 0.0, mutation_{rate} = 0.05, population_{size} = 5\}$. These values correspond to the lower bound for $crossover_{rate}$, $mutation_{rate}$ and $population_{size}$ in their fixed initial range. Revac has found similar parameter calibrations for this Scenario considering its 5 different runs. ParamILS fixed always $population_{size}$ at 5. Parameter configurations found by SPO are very similar to those found by Revac. BRS method found five different calibrations with $population_{size}$ ranging from 5 to 13, $crossover_{rate}$ ranging from 0.11 to 0.4 and $mutation_{rate}$ ranging from 0.02 to 0.06.

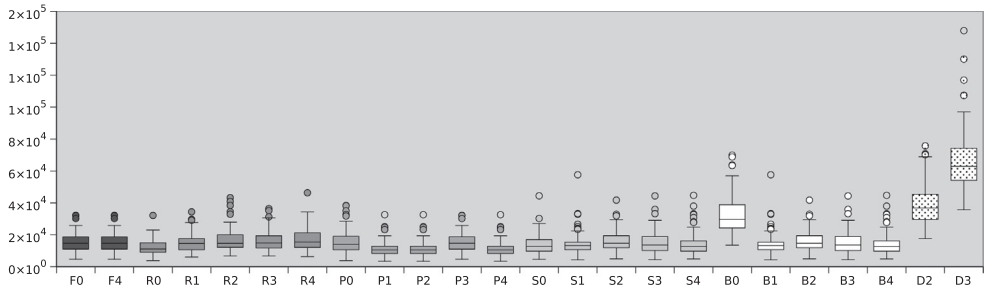
Table 9
Parameters – Scenario 1.

Method	Id	$crossover_{rate}$	$mutation_{rate}$	$population_{size}$
F-Race	F0	0.00	0.05	5
	F1	0.00	0.10	25
	F2	0.00	0.10	100
	F4	0.00	0.05	5
Revac	R0	0.20	0.03	6
	R1	0.25	0.03	7
	R2	0.36	0.02	9
	R3	0.21	0.02	10
	R4	0.18	0.02	11
ParamILS	P0	0.50	0.03	5
	P1	0.00	0.03	5
	P2	0.00	0.03	5
	P3	0.00	0.05	5
	P4	0.00	0.03	5
SPO	S0	0.19	0.04	5
	S1	0.19	0.03	7
	S2	0.06	0.04	6
	S3	0.34	0.04	5
	S4	0.15	0.03	7
BRS	B0	0.16	0.06	8
	B1	0.40	0.02	5
	B2	0.12	0.03	13
	B3	0.11	0.05	9
	B4	0.18	0.05	7

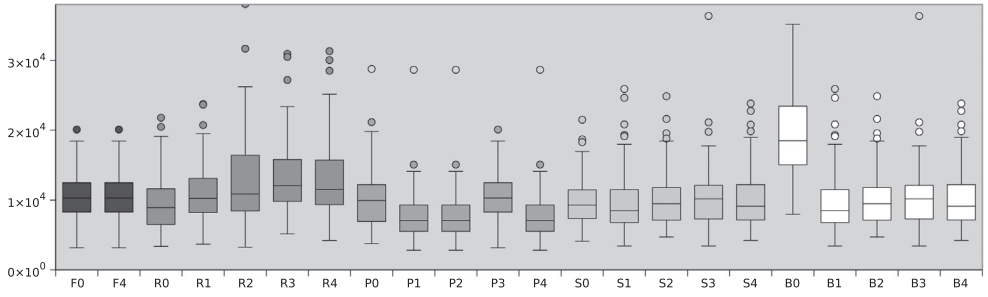
Graphs in Fig. 5 show the performance of parameter calibrations in Tables 9 and 10 when solving the test set. We have also evaluated the performance of three well known default parameter configurations proposed by De Jong [10] and Grefenstette [12]. We list those parameter configurations in Table 10.

Table 10
Default parameter configurations by [10,12].

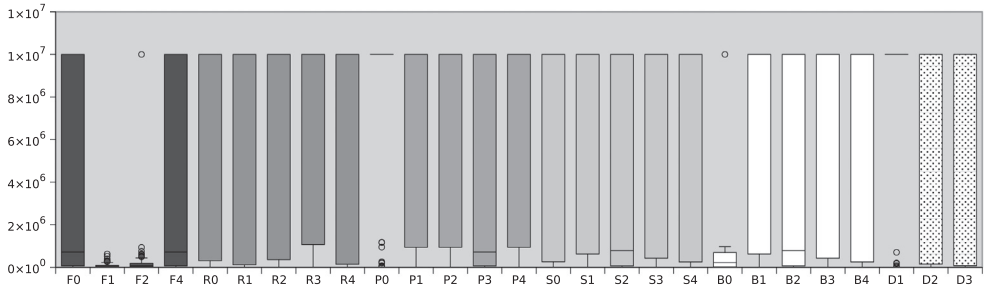
Id	$crossover_{rate}$	$mutation_{rate}$	$population_{size}$
D1	0.600	0.001	50
D2	0.950	0.010	30
D3	0.450	0.010	80



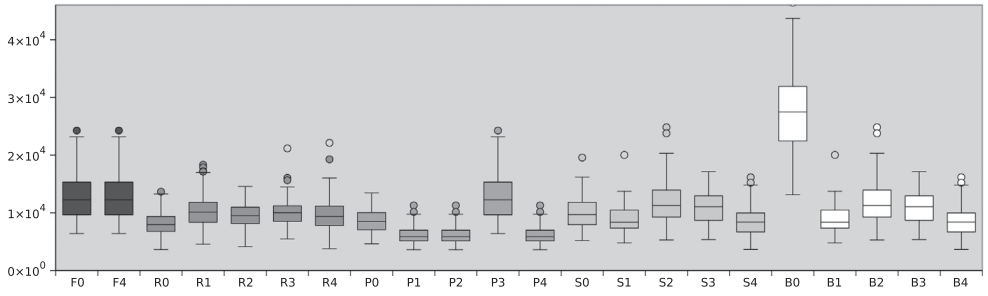
(a) f5



(b) f6



(c) f7



(d) f8

Fig. 5. Performance Scenario 1: boxplots showing the performance of the best parameter calibrations from Tables 9 and 10.

In these figures we use box-plots[4] of 100 independent executions of the SGA.

Performance of configurations *F*1 and *F*2 has shown the worst ones. We can observe that it is always a good option to use a tuning method. Even a simple method as BRS is able to get in most cases a better performance than those obtained using parameter tuning by analogy (*D*2 or *D*3). Moreover, considering the performance obtained from calibration *B*0, we can conclude that the use of more sophisticated tuning methods like Revac, ParamILS and SPO deliver more reliable parameter configurations.

A very interesting situation in this scenario is the performance of parameter calibrations for function f_7. Fig. 5(c) shows that the best results are obtained using calibrations *F*1, *F*2 and *B*0. These calibrations consider a high $mutation_{rate}$ value.

3.4. Scenario 2

Scenario 2 considers as training set the four multimodal functions: $\{f_5, f_6, f_7$ and $f_8\}$ and as testing set the four unimodal

[4] The lower and upper border of the box represents the lower and upper quartile of the performance distribution. The median value is drawn on it. The end of the lines represents the maximum and minimum performance value of the sampling and circles represent outliers.

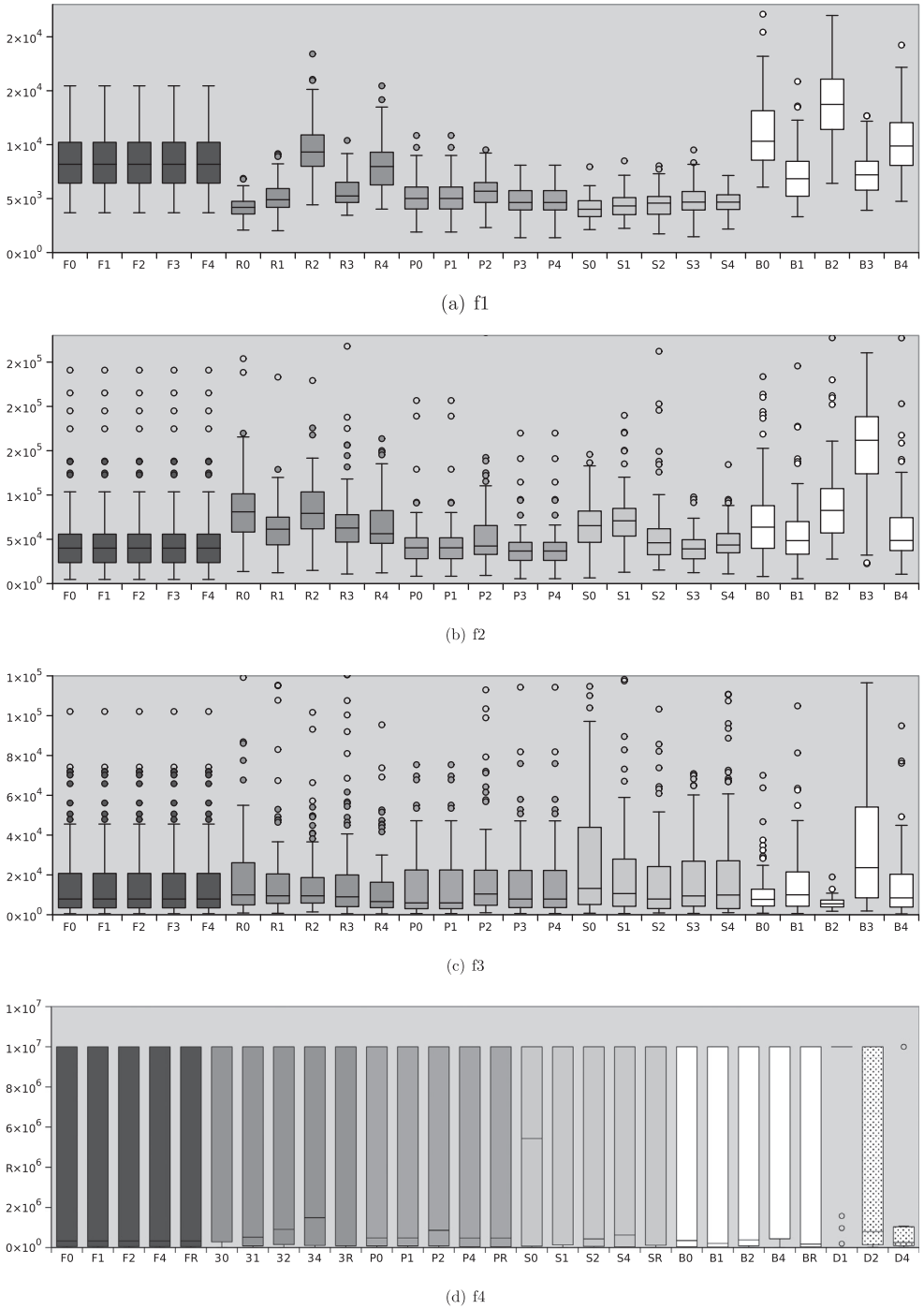
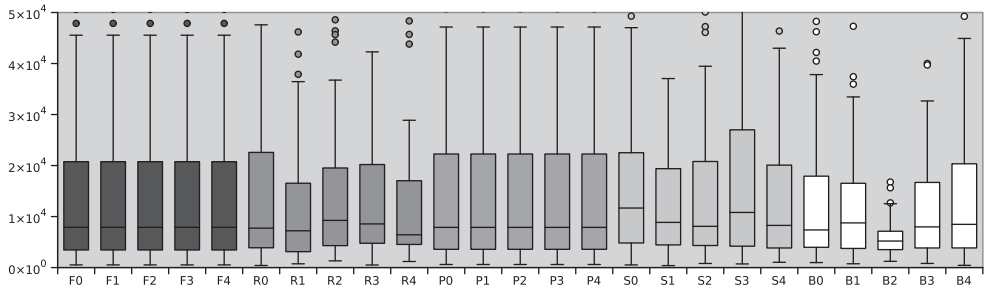


Fig. 6. Performance Scenario 2: boxplots showing the performance of the best parameter calibrations from Tables 11 and 10.

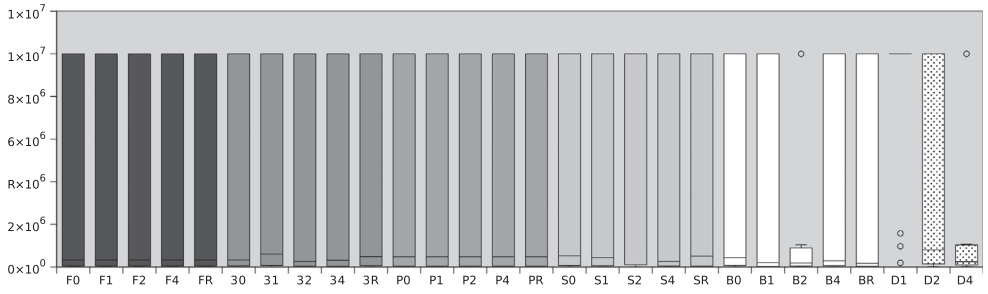
functions: $\{f_1, f_2, f_3$ and $f_4\}$. Table 11 shows the final parameter configurations found by each tuning method for Scenario 2.

In this scenario F-Race found five times the same parameter calibration $c = \{crossover_{rate} = 0.0, mutation_{rate} = 0.05, population_{size} = 5\}$. These values correspond to the minimum possible value for $crossover_{rate}$, $mutation_{rate}$ and $population_{size}$ respectively in their set. Revac selected similar parameter calibrations for this scenario considering its 5 different runs. As in Scenario 1, parameter configurations found by SPO are very similar to those calibrations found by Revac. In this case, $mutation_{rate}$ took values between 0.02 and 0.03, $crossover_{rate}$ took values between 0.07 and 0.16 and $population_{size}$ fixed at 5. In both scenarios their parameter calibrations only differ in the value of $crossover_{rate}$. For this training set $crossover_{rate}$ shows to be a relevant operator when solving functions: $f_5 - f_8$. This is also confirmed by the results obtained by BRS method that found $crossover_{rate}$ ranges from 0.04 to 0.95.

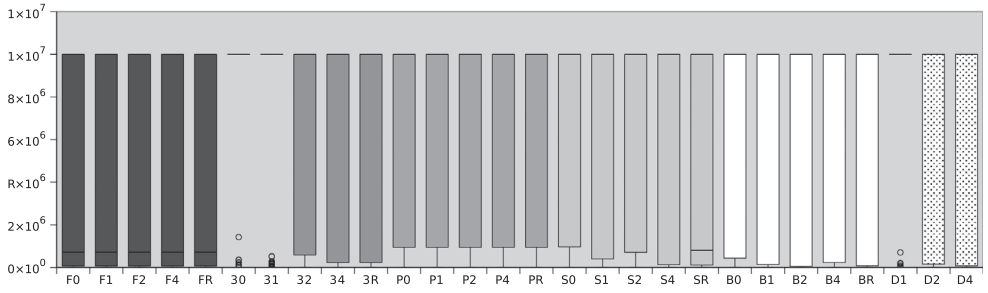
Graphs in Fig. 6(a)–(c) show the performance of the best performing parameter configurations found. We can observe that for function f_1 the best performance was achieved by ParamILS and SPO configurations, while Revac and F-Race configurations can be considered as third and fourth positions respectively. For function f_1 the use of the BRS's configurations deliver the worst performance. For function f_2 a similar behavior can be described, but in this case, F-Race configurations found the best performance, followed



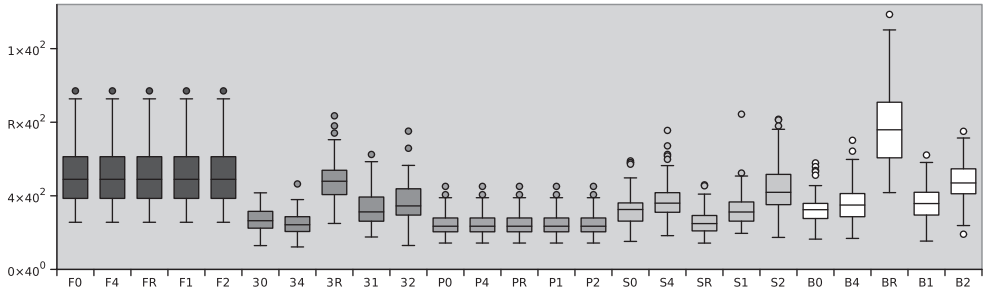
(a) f3



(b) f4



(c) f7



(d) f8

Fig. 7. Performance Scenario 3: boxplots showing the performance of the best parameter calibrations from Tables 12 and 10.

by ParamILS, SPO and Revac configurations. Again, the worst performance was achieved by BRS configurations. In the case of f_3 that is the simplest function in test suite the performance of all parameter configurations are very similar.

Note that the performance of configurations found by BRS is unstable and it is not clear which is the best one.

As in Scenario 1, in Scenario 2 we can observe that it is always a good option to use a tuning method, even a simple method as BRS is able to get in most cases a better performance than those obtained using parameter tuning by analogy.

In this scenario, function f_4 shows a very different behavior compared to the rest of functions. As we see in Fig. 6(d), the best performance for this function was found by calibration $D3 = \{crossover_{rate} = 0.45,\ mutation_{rate} = 0.01,\ population_{size} = 80\}$. This parameter calibration fixes a $mutation_{rate}$ value similar to those set by the other parameter calibrations, but $crossover_{rate}$ and $population_{size}$ values are different to those fixed by tuning methods. For the tuning methods, small $population_{size}$ and $crossover_{rate}$ values were better for the training set; hence, tuning processes converge to these parameter values.

Table 11
Parameters – Scenario 2.

Method	Id	$crossover_{rate}$	$mutation_{rate}$	$population_{size}$
F-Race	F0	0.00	0.05	5
	F1	0.00	0.05	5
	F2	0.00	0.05	5
	F3	0.00	0.05	5
	F4	0.00	0.05	5
Revac	R0	0.14	0.02	5
	R1	0.16	0.02	6
	R2	0.21	0.01	13
	R3	0.08	0.01	8
	R4	0.17	0.01	10
ParamILS	P0	0.10	0.03	5
	P1	0.10	0.03	5
	P2	0.30	0.03	5
	P3	0.00	0.03	5
	P4	0.00	0.03	5
SPO	S0	0.07	0.02	5
	S1	0.16	0.02	5
	S2	0.12	0.02	5
	S3	0.10	0.03	5
	S4	0.11	0.02	5
BRS	B0	0.54	0.03	9
	B1	0.04	0.02	9
	B2	0.09	0.01	20
	B3	0.95	0.01	6
	B4	0.04	0.03	12

3.5. Scenario 3

Scenario 3 considers as training set two unimodal and two multimodal functions: $\{f_1, f_2, f_5, f_6\}$ and as testing set the other two unimodal and two multimodal functions: $\{f_3, f_4, f_7$ and $f_8\}$. Table 12 shows the final parameter configurations found by each tuning method for Scenario 3.

In this scenario F-Race found five times the same parameter calibration $c = \{crossover_{rate} = 0.0, mutation_{rate} = 0.05, population_{size} = 5\}$. Moreover, this is the same parameter calibration it found for training set of Scenario 2. That shows the importance of functions f_5 and f_6 used in both tuning processes. Revac selected similar parameter calibrations for this Scenario considering its 5 different runs.

Table 12
Parameters – Scenario 3.

Method	Id	$crossover_{rate}$	$mutation_{rate}$	$population_{size}$
F-Race	F0	0.00	0.05	5
	F1	0.00	0.05	5
	F2	0.00	0.05	5
	F3	0.00	0.05	5
	F4	0.00	0.05	5
Revac	R0	0.16	0.03	5
	R1	0.12	0.03	5
	R2	0.22	0.02	14
	R3	0.15	0.02	9
	R4	0.10	0.02	10
ParamILS	P0	0.00	0.03	5
	P1	0.00	0.03	5
	P2	0.00	0.03	5
	P3	0.00	0.03	5
	P4	0.00	0.03	5
SPO	S0	0.04	0.02	9
	S1	0.47	0.02	7
	S2	0.11	0.03	5
	S3	0.03	0.03	7
	S4	0.05	0.04	7
BRS	B0	0.09	0.02	9
	B1	0.04	0.03	9
	B2	0.24	0.02	22
	B3	0.14	0.02	10
	B4	0.04	0.03	12

Table 13
Tuning time [min].

	F-Race	Revac	ParamILS	SPO	BRS
Scenario 1	1107.5	57.0	65.2	59.3	97.5
Scenario 2	154.2	64.2	59.4	90.0	129.5
Scenario 3	121.5	48.3	56.0	54.8	121.9
Average	127.7	56.5	60.2	68.0	116.3

For this scenario ParamILS delivered always the same parameter configuration $c' = \{crossover_{rate} = 0.0, mutation_{rate} = 0.05, population_{size} = 5\}$. Parameter configurations found by SPO are very similar to those calibrations found by Revac. BRS method found five different calibrations with $population_{size}$ ranging from 9 to 22.

Graphs in Fig. 7 show the performance of parameter configurations found for these functions.

All tuning methods have a quite good performance for solving function f_3. In function f_8, the performance of ParamILS calibrations can be considered the best one, followed by Revac and SPO, a very unstable BRS and, finally, F-Race.

For functions f_4 and f_7 were useless. In the case of function f_4 there is one parameter configuration $B2 = \{crossover_{rate} = 0.24, mutation_{rate} = 0.02, population_{size} = 22\}$ found by BRS that is able to perform the best performance. Moreover, default parameter configuration $D3 = \{crossover_{rate} = 0.45, mutation_{rate} = 0.01, population_{size} = 80\}$ was also able to obtain a good performance. Both configurations $B2$ and $D3$ have higher $crossover_{rate}$ and $population_{size}$ than the other configurations.

4. Conclusions

In this section we summarize the main global conclusions about our comparison of these parameter tuning methods.

First, it is clear that in terms of performance it is difficult to determine if there exists a tuning method able to find always the best performing parameter calibration. Hence, we briefly enumerate the main features of tuning methods which can determine their usability in different tuning scenarios.

- **About tuning by analogy**: It is clear that to use a tuning method instead of using a set of recommended parameter values is a good choice. For all the functions, there is at least one parameter configuration that performs better than the default and the set of random parameter configurations.
- **About the use of tuning methods**: It is clear that the use of a blind random search approach to find a good parameter configuration has been a good option for this algorithm. The average performance obtained by its calibrations is for all functions comparable to the performance obtained by tuning methods. Moreover, in most cases, the average performance of the calibrations obtained by the blind random search is significantly better compared to the average performance obtained by the default and random configurations.

  It is important to note that the only result that can be obtained from the blind random search process is a calibration that performs well with one seed, whereas the result obtained by ParamILS is a configuration that is statistically better than all others studied considering a significant set of seeds. It is also important to note that tuning process performed by the blind random search strategy is, on average, more than twice time consuming compared to the process performed by ParamILS or SPO for getting equivalent quality results. Table 13 shows the average tuning time spent by tuning methods in each scenario.

- **Parameters nature**: It is important to note that Revac is not able to search categorical parameters, because its transformation process is done on a continuous search space for the parameters. The categorical parameters have categorical values; therefore, there are not numerical relations among their *values*, as is required by this methods. F-Race, ParamILS and SPO are able to tune numeric and categorical parameters. Firstly, both require the definition of discrete set of values for the parameters, which in some cases can be restrictive.
- **Budget of the process**: In our experiments, we considered a fixed budget of 1000 SGA executions, because this is the budget established by Revac for its tuning processes. As can be seen in experimental results presented, the amount of executions required for each tuning process depends on the problem instance to tune and the tuning method that is being used. In general, it is not possible to determine this value before the process; hence to keep this value fixed is not a good option.

  From a practical point of view, methods like ParamILS and SPO have the advantage that at each step of the tuning process they are able to report a solution; hence, the process can be stopped at any point of the search. After the first M SGA executions of Revac and $r^* size(C)$ SGA executions of F-Race, they could also be stopped and they will be able to deliver results.
- **Stochastic nature**: Revac, ParamILS and SPO are clearly stochastic nature methods; hence, we have run each method 5 times for each function to tune. From the results we can note that, in general, all are able to find similar results in different executions, but there are some cases, where the results of some methods clearly depend on the seed used. The most clear examples are the results of F-Race in Scenario 1. Even when F-Race is not a stochastic nature method, its evaluations depend on the seeds selected to execute the algorithm to tune.
- **Input requirements**: The definition of discrete sets of values for F-Race is a hard task, because their sizes are strongly related to the time the tuning process will take. ParamILS also requires the definition of discrete sets of values for parameters, but the size of these sets has shown not being determinant on the quality of the tuning process. On the other hand, Revac and SPO just require the definition of ranges of values to search.
- **Parameter interactions**: F-Race, ParamILS and SPO work with the notion of configuration more than just parameter values, i.e. they put more attention in the interaction between parameter values compared to the process performed by Revac where each parameter range of values is iteratively reduced.
- **Output expected**: Related to the output of each method, we define three kinds of users. The first one is a user who wants to compare its new metaheuristic with respect to other metaheuristics methods. In this case, the user is interested in finding *one* parameter calibration able to obtain the best performance of the metaheuristic method. The second kind of user wants to explore and analyze the capabilities of his metaheuristic. Finally, the third kind of user is who tries to quickly find a good configuration for the problem.

  For the first user ParamILS and SPO can be good options to tune his/her algorithm. The search performed by Revac, on the other hand, is oriented to find good ranges of values for each parameter. The problem with this approach can appear when more than one range gives a good performance of the algorithm to tune. In these cases, Revac can converge to a wide range of values which can include not good parameter values.

  For the second kind of user, the recommendation is to spend more time using F-Race. The final set of configurations delivered by F-Race is an important source of information which can be used to detect not effective or redundant components [19]. This study has proved that F-Race is useful to assist the design process.

Table 14
SGA executions.

	F-Race	Revac	ParamILS	SPO	BRS
Scenario 1	3170.4	3186.4	2433.4	2465.6	1115.2
Scenario 2	3210.4	1946.4	1474.8	2243.2	2012.0
Scenario 3	3032.8	2074.4	730.4	1448.0	2036.0
Average	3137.9	2402.4	1546.2	2052.3	1721.1

  For the last kind of user, the recommendation is to use ParamILS which performs a strongly intensified search on the parameter search space and can quickly find good performing parameter configurations able to solve the problem at hand. Performance of ParamILS calibrations shown always a very good performance in almost all cases studied.
- **Time**: In terms of time, it is important to note that the quality criterion of the SGA tuned in this study is the amount of evaluations to the optimum. Therefore, in our case, a good parameter calibration determines, always, a low time consuming SGA execution. Table 13 shows a summary of the tuning time (in minutes) measured in Section 3. Here we can note that ParamILS, Revac and SPO take, on average, half the time required by F-Race and BRS to tune the same scenario. This is due to the fact that ParamILS and SPO work with a small set of configurations and, usually, can quickly improve their quality. In this case, Revac performance can be helped by a set of good quality configurations which determine its efficient behavior even when it works with a large population. On the other hand, F-Race is forced to test its entire large set of initial configurations. Note that these results show that ParamILS and SPO methods are tuning process strongly focused on finding good parameter calibrations.

  Table 14 shows the average number of SGA executions required by each tuning method to reach its final parameter calibration. Here we can observe that F-Race is the most expensive method, this due to the execution of the initial number of runs without elimination of calibrations. We remark that F-Race is not able to deliver a result until it performs its initial phase of testing runs. Revac looks like the second more runs-consuming method. This condition is mainly due to the size of the final range of parameter values. Here, we arbitrarily decided that Revac process can be considered finished when all the parameter ranges have been reduced to their 5%. However, some users can consider that a wide range of values is sufficiently good. SPO and ParamILS look like the more efficient tuning methods using just a 40–50% of the total budget assigned.

  Blind random search method is very competitive considering the number of SGA executions it required to find its best parameter configuration. However, it is important to emphasize that the time required for the SGA executions is always higher than the time required by the other tuning processes.
- **Stop criterion**: From the methods used, the only method able to stop the search process by itself is F-Race. It is able to detect only one configuration left in the set of candidate configurations and then stop the process. Some possible criteria to consider can be related to the convergence of the tuning process and/or the percentage of reduction of the initial range of values.

  Moreover, it is important to remark that tuning processes performed by ParamILS, SPO and even Revac can be stopped at almost any point of the search and they will be able to deliver a good quality parameter calibration.
- **Training and testing sets**: From the results of the three scenarios the difference that can produce the selection of the training sets in the final results of the tuning methods is not clear. In our experiments we considered the *multimodality* of instances as a feature to design the training and testing sets, but apparently this is not a feature that differentiates the instances for the SGA. According

Table 15
Features of tuning techniques.

Method	F-Race	Revac	ParamILS	SPO
Type	Experimental Design	Search Based	Search Based	Experimental Design
Initial requirements	Discrete set of values	Interval of values	Discrete set of values	Interval of values
Expected output	Set of best performing configurations	One value and one interval of values for each parameter	Best performing configuration	Best performing configuration
Number of parameters	3	4	4	5
Stop criterion	Number of executions or one configuration left	Number of executions	Number of executions/time	Number of executions/time

to bibliography, diverse training set should be considered while tuning parameter for an algorithm in order to avoid over-fitting effects. A good idea can be to incorporate *boostrap* methods in tuning methods to reduce these problems.

Table 15 summarizes the main features of the tuning methods discussed previously.

4.1. Future work

It is important to emphasize the importance of using tuning methods to estimate parameter values.

In this study we have compared the capabilities of four well-known tuning methods proposed in the literature. All these methods present advantages and disadvantages and the idea of this study is to propose guidelines to select the tuning method to use according to the user requirements.

As future work we propose the development of an automated strategy to select the best tuning approach according to the algorithm and problem instance at hand based on the idea of portfolio of tuning strategies.

References

[1] B. Adenso-Diaz, M. Laguna, Fine-tuning of algorithms using fractional experimental designs and local search, Operations Research 54 (1) (2006) 99–114.
[2] T. Bartz-Beielstein, C.W.G. Lasarczyk, M. Preuss, Sequential parameter optimization, in: Proceedings of the IEEE Congress on Evolutionary Computation, vol. 1, 2005, pp. 773–780.
[3] T. Bartz-Beielstein, K.E. Parsopoulos, M.N. Vrahatis, Analysis of particle swarm optimization using computational statistics, in: Proceedings of the International Conference Numerical Analysis and Applied Mathematics, 2004, pp. 34–37.
[4] T. Bartz-Beielstein, K.E. Parsopoulos, M.N. Vrahatis, Design and analysis of optimization algorithms using computational statistics, Applied Numerical Analysis and Computational Mathematics (ANACM) 1 (2) (2004) 413–433.
[5] T. Bartz-Beielstein, Sequential parameter optimization – an annotated bibliography. CIOP Technical Report 04/10, Research Center CIOP (Computational Intelligence, Optimization and Data Mining), Cologne University of Applied Science, Faculty of Computer Science and Engineering Science, April 2010.
[6] S. Becker, J. Gottlieb, T. Stützle, Applications of racing algorithms: an industrial perspective, in: Proceedings of the International Conference on Artificial Evolution, vol. 3871, 2006, pp. 271–283.
[7] M. Birattari, T. Stützle, L. Paquete, K. Varrentrapp, A racing algorithm for configuring metaheuristics, in: Proceedings of the Genetic and Evolutionary Computation Conference, 2002, pp. 11–18.
[8] M. Birattari, Z. Yuan, P. Balaprakash, T. Stützle, F-race and iterated F-race: an overview. Technical Report TR/IRIDIA/2009-018, IRIDIA, Université Libre de Bruxelles, June 2009.
[9] A. Czarn, C. MacNish, K. Vijayan, B. Turlach, R. Gupta, Statistical exploratory analysis of genetic algorithms, IEEE Transactions on Evolutionary Computation 8 (4) (2004) 405–421.
[10] K.A. De Jong, An analysis of the behaviour of a class of genetic adaptive systems (Ph.D. thesis), University of Michigan, 1975.
[11] G. Eiben, M.C. Schut, New ways to calibrate evolutionary algorithms, in: Advances in Metaheuristics for Hard Optimization, Springer, Berlin, Heidelberg, 2008, pp. 153–177.
[12] J. Grefenstette, Optimization of control parameters for genetic algorithms, IEEE Transactions on Systems, Man and Cybernetics 16 (1) (1986) 122–128.
[13] D. Huang, T.T. Allen, W.I. Notz, N. Zeng, Global optimization of stochastic black-box systems via sequential kriging meta-models, Journal of Global Optimization 34 (March (3)) (2006) 441–466.
[14] F. Hutter, H.H. Hoos, T. Stützle, Automatic algorithm configuration based on local search, in: Proceedings of the Conference on Artificial Intelligence, 2007, pp. 1152–1157.
[15] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Kevin Murphy, Time-bounded sequential parameter optimization, in: Proceedings of the 4th International Conference on Learning and Intelligent Optimization, LION'10, Springer-Verlag, 2010, pp. 281–298.
[16] Frank Hutter, Holger H. Hoos, Kevin Leyton-Brown, Kevin P. Murphy, An experimental investigation of model-based parameter optimisation: SPO and beyond, in: Proceedings of the 11th Annual Conference on Genetic and Evolutionary Computation, GECCO'09, ACM, 2009, pp. 271–278.
[17] Donald R. Jones, Matthias Schonlau, William J. Welch, Efficient global optimization of expensive black-box functions, Journal of Global Optimization 13 (4) (1998) 455–492.
[18] E. Montero, Calibration strategies for bio-inspired population-based algorithms that solve combinatorial optimization problems (Ph.D. thesis), Université de Nice, Sophia-Antipolis, 2011.
[19] E. Montero, M.C. Riff, B. Neveu, New requirements for off-line parameter calibration algorithms, in: Proceedings of the Congress on Evolutionary Computation, 2010, pp. 1–8.
[20] V. Nannen, A.E. Eiben, Efficient relevance estimation and value calibration of evolutionary algorithm parameters, in: IEEE Congress on Evolutionary Computation, IEEE, 2007, pp. 103–110.
[21] V. Nannen, A.E. Eiben, Relevance estimation and value calibration of evolutionary algorithm parameters, in: Proceedings of the International Joint Conference for Artificial Intelligence, 2007, pp. 975–980.
[22] M. Pelikan, D.E. Goldberg, F.G. Lobo, A survey of optimization by building and using probabilistic models, Computational Optimization and Applications 21 (1) (2002) 5–20.