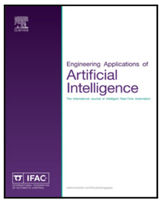
Contents lists available at ScienceDirect

Engineering Applications of Artificial Intelligence

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



Tuners review: How crucial are set-up values to find effective parameter values?

Elizabeth Montero [*], María-Cristina Riff, Nicolás Rojas-Morales

Department of Computer Science, Universidad Técnica Federico Santa María, Avenida España 1680, Valparaíso, Chile

ARTICLE INFO

Keywords:
Parameter setting
Tuning methods
Parameter search space definition

ABSTRACT

ParamILS, I-Race and Evoca are well-known tuning methods designed to search quality parameter calibrations for metaheuristic algorithms. The set-up of parameter search space can strongly affect the performance of tuning methods. In this work we study how the parameter search definitions affect the quality of parameter calibrations delivered by these tuners. An experimental evaluation using two well known metaheuristic algorithms and a real life case is presented. We also provide some guidelines to consider when defining parameters search spaces according to the tuner used in order to obtain the best performance they can find.

1. Introduction

Tuning methods have shown to be effective strategies to improve metaheuristics performance finding appropriate values for their parameters. A set of values for each parameter a metaheuristic defines is usually called a parameter calibration. A typical genetic algorithm usually defines a set of at least three parameters: population size, crossover rate and mutation rate. Moreover, most times these parameters are coupled with some design decisions that drive the designer to define proper selection, mutation and crossover operators. In fact, parameter values can be divided into two categories: categorical and numerical. Categorical parameters are those that have a finite domain with no distance metric or ordering between values, while numerical parameters are those whose domains are subsets of $\mathbb{N}$ or $\mathbb{R}$. Popular recommendations to set numerical values can be found in literature for genetic algorithms (De Jong, 1975; Grefenstette, 1986), but setting categorical parameters can become a harder task that requires the definition of proper "values" for these parameters.

Moreover, when tuning a categorical parameter, most tuning methods search all possible values for the parameter. On the other hand, when tuning a numerical parameter, a finite number of values belonging to its range of values is selected to perform the tuning. The selection of these sets of values is an experience-dependent process that can strongly affect the quality of the tuning process. We consider as quality/performance of a tuning process the quality of solutions obtained by the tuned metaheuristic algorithm when using the calibration obtained by the tuning process and the computational effort involved on the tuning process. In this work we study the performance of three well-known tuning methods considering complex tuning scenarios in order to analyze the influence of parameter search space definitions.

Several tuning methods have been proposed during the last years. These methods can be categorized in four main areas (Eiben and Smit, 2011): Sampling methods, model-based methods, screening methods and meta-evolutionary algorithms. Section 2 gives an overview of state-of-the-art tuning techniques and algorithms. In this work, we focus our attention on three well-known tuners: ParamILS (Hutter et al., 2009), I-Race (Birattari et al., 2010) and Evoca (Riff and Montero, 2013). ParamILS corresponds to an iterated local search algorithm that starting from a default parameter calibration searches its neighborhood looking for parameter calibrations of better quality. The process is iterated until a fixed stopping criterion is met. I-Race follows the framework of iterative screening methods. It constructs candidate calibrations based on a probability model, evaluates the most promising ones and updates the probability model to bias the next sample. Evoca works with a population of parameter calibrations that undergo selection, crossover and mutation operations to improve their quality.

Related to the definition of parameter search spaces ParamILS requires the definition of a discrete set of values for each parameter being tuned in order to describe finite neighborhoods. I-Race also requires a finite set of possible values for each categorical parameter being tuned and a range of values for each numerical parameter. Moreover, I-Race can also use a user-defined set of parameter calibrations to sample its first iteration. Evoca requires the definition of a set of possible values for each categorical parameter and a range of values for each numerical

* Corresponding author.
E-mail address: elizabeth.montero@usm.cl (E. Montero).

https://doi.org/10.1016/j.engappai.2018.09.001
Received 12 February 2018; Received in revised form 4 August 2018; Accepted 6 September 2018
Available online 21 September 2018

parameter being tuned. These tuning methods are described in detail in Section 3.

We have realized that using ParamILS is not an easy task for users/designers because they are asked to explicitly define the parameter search space. This definition requires, in most cases, the selection of a small subset of values each numerical parameter can take. For categorical parameters all possible values are usually considered. On the other hand, the performance of I-Race has shown being strongly dependent of a proper definition of the initial setup of parameter space used. A complex initial setup may discourage some users who find it difficult to select adequate settings for these methods. Unlike these algorithms, Evoca works well with simpler definitions of parameter search spaces. We deepen the motivation of our research in Section 4. Moreover, a performance analysis of ParamILS (FocusedILS), I-Race and Evoca tuners using different setup of parameters space is presented in Section 5. We analyze the difference in the quality of parameter calibrations obtained using various experimental setups. Section 6 presents a summary of main recommendations on the definition of parameter spaces for tuning methods including the analysis of a real-life case. Moreover, we also included some preliminary results on collaborative tuning scenarios between the tuning methods studied. Our conclusions and future work are presented in Section 7.

2. Related work

Tuning methods have shown being effective strategies to find good quality parameter calibrations for metaheuristics algorithms. Several tuning methods have being proposed, starting with Grefenstette' Meta-GA (Grefenstette, 1986) until nowadays where four categories of tuners can be identified (Eiben and Smit, 2011): Sampling methods, model-based methods, screening methods and meta-evolutionary algorithms.

Sampling methods reduce the search effort by cutting the number of parameter calibrations evaluated with respect to a full factorial design. Most sampling methods are used as starting points for model-based methods or as initialization methods. Calibra (Adenso-Diaz and Laguna, 2006) is an example of *iterative* sampling method. After each step, *iterative* sampling methods refine the parameter search area from which new parameter calibrations will be sampled.

Model-based methods construct models of utility landscapes of parameter search spaces. In these methods the total number of experiments is reduced by replacing some of the "real" executions by estimations of the proposed model. SPO (Bartz-Beielstein et al., 2012) is a well-known *iterative* model-based method. It performs a multi-stage procedure, that iteratively generates a set of new parameter calibrations and predicts their quality using the proposed model. Best performing calibrations are tested to prove their quality and then used to update the current model. SMAC (Hutter et al., 2011) method is an improved model-based tuning method. It uses a specially designed comparison mechanism and random forests to model response surfaces. It is able to work with categorical and numerical parameters, and also, with multiple problem instances that show different features.

Screening methods try to identify the best parameter calibrations from a given set performing a fixed number of tests/executions. *Iterative* screening methods, determine a subset of parameter calibrations that deserve further investigation at each step. F-Race (Birattari et al., 2002) is the most known screening tuning method. I-Race (Balaprakash et al., 2007) is an extension of F-Race that at each iteration starts an F-Race procedure with a sampling of parameter calibrations in a large region of the parameter search space. At each step, I-Race uses a multi-variate normal distribution to fit the F-Race surviving parameter calibrations. These calibrations are then used to sample calibrations for a new iteration. This screening and generating procedure is repeated until the computational budget for tuning is reached.

Search-based methods face the tuning problem as an optimization problem, hence a good strategy to find high quality calibrations can be to use a metaheuristic algorithm. ParamILS (Hutter et al., 2009) is an iterated local search algorithm that works searching neighborhoods of parameter calibrations. A version of ParamILS (FocusedILS) uses a comparison method able to increase the number of executions for comparing parameter calibrations when required. FocusedILS version of ParamILS was used in our experiments. Evoca (Riff and Montero, 2013) is an evolutionary-based algorithm that works with a population of calibrations. In Evoca, the population size is computed considering the number of parameters and their domain sizes. The key idea is to include a set of well distributed values for each parameter on its first population. Evoca implements two transformation operators: a wheel crossover and a hill climbing first improvement mutation operator.

Tuning methods have become an important tool in metaheuristic research area. Their use has been extended to metaheuristic design, i.e., to determine not only the value of numerical parameters of the metaheuristic, but also, of its categorical parameters. In Montero and Riff (2014a) authors use I-Race and Evoca to select the transformation operators of a standard genetic algorithm and the mutation operator of a multi-objective immune-based algorithm. In Bezerra et al. (2014) authors use a large set of both numerical and categorical parameters to perform the component selection of a generic Multi-objective Evolutionary Algorithm (MOEA). Here, I-Race is used to select the population size, population type, offspring size, external archive type and size, selection scheme, removal set-partitioning quality indicator and diversity operators. Moreover, in Radulescu et al. (2013) authors use I-Race to tune the bi-objective problem of finding anytime algorithms. Anytime algorithms are those able to find quality solutions at any time of their execution. Hypervolume indicator is used to define a single-objective tuning problem that searches for an approximation of the Pareto front of quality of solutions versus time. Moreover, they study the anytime behavior of multi-objective evolutionary algorithms where the quality of solutions obtained is also evaluated using the hypervolume indicator.

Nowadays the main focus of tuning methods is related to the optimization of several criteria, also known as multi-objective problems (Augusto et al., 2006). The bi-objective problem that considers the performance over time of metaheuristics is typically considered. Several multi-criterion approaches proposed in literature are based on single-criteria tuning methods as SPRINT-Race (Zhang et al., 2015) that takes inspiration from the *-Race approaches, MO-ParamILS (Blot et al., 2016) that is based on ParamILS. On the other hand, some of these new proposals are inspired in well known MOEAs as M-FETA (Smit et al., 2010) and EMOPaT (Ugolotti and Cagnoni, 2014). Furthermore, the tuning of multi-criterion algorithms optimizing a set of typical multi-objective indicators has attracted the attention of researchers in the area lately (Blot et al., 2017).

3. Tuning methods

This section introduces the three tuning methods: ParamILS, I-Race and Evoca. Here, we explain the main components of tuning processes performed by each tuning method studied in this work. We have selected these three methods because they have been extensively used in the tuning research area and they have implementations available online.

3.1. Parameter iterated local search

The Parameter Iterated Local Search (ParamILS) method (Hutter et al., 2009) works as an iterated local search algorithm. Algorithm 1 shows the structure of ParamILS method. Starting with a user defined parameter calibration, R tries searching randomly for a better calibration are performed. The resulting calibration undergoes local search through the *IterativeFirstImprovement*(c) function in line 8.

At each iteration, lines 9 to 21, ParamILS performs s random perturbations and the best calibration is then improved by the local search process. Its performance is then compared to the best parameter calibration found so far. There is also, a restart probability $p_{restart}$, which allows the method to escape from local optimum.

Algorithm 1: ParamILS tuning method

```
1  $c_0 \leftarrow$ default calibration;
2  for $i \leftarrow 1$ to $R$ do
3      $c \leftarrow$ random calibration;
4      if better($c$, $c_0$) then
5          $c_0 \leftarrow c$
6      end
7  end
8  $c_{ils} \leftarrow IterativeFirstImprovement(c_0)$;
9  while maximum budget is not reach do
10     $c \leftarrow c_{ils}$;
11     for $i \leftarrow 1$ to $s$ do
12         $c \leftarrow$ random calibration;
13     end
14     $c \leftarrow IterativeFirstImprovement(c)$;
15     if better($c$, $c_{ils}$) then
16         $c_{ils} \leftarrow c$;
17     end
18     if $p_{restart}$ then
19         $c_{ils} \leftarrow$ random calibration;
20     end
21 end
22 return $c$;
```

The *IterativeFirstImprovement*(c) process searches randomly in the neighborhood of a parameter calibration for a *better* one. A neighbor calibration is created by changing the value of one parameter in the calibration. FocusedILS defines the dominance concept to compare the performance of two parameter calibrations. A parameter calibration c dominates a calibration c', if and only if, the average performance of the target algorithm using c for solving n seeds is *better* than its performance using calibration c' for solving n' seeds, with $n > n'$.

ParamILS requires the setting of four (meta-)parameters: R, s, $p_{restart}$ and the maximum budget.

ParamILS is able to deal with conditional parameters. In this case, neighborhoods of conditional parameters are searched only when they are active.

Frank Hutter's implementation of ParamILS is available in his website.[1]

3.2. Iterated F-Race

I-Race is an iterative version of F-Race algorithm (Birattari et al., 2002). At each iteration, F-Race is used to identify the statistically best performing parameter calibrations. F-Race uses Friedman's two-ways ANOVA by ranks test to compare parameter configurations, hence no assumptions about the underlying distributions of the performance of tuned algorithms are required. I-Race uses these best parameter calibrations to bias the sample of new candidate calibrations for the next iteration.

Algorithm 2 shows I-Race structure. Considering a fixed maximum computational budget (executions or time), I-Race (Balaprakash et al., 2007; Birattari et al., 2010; López-Ibáñez et al., 2016) determines the number of iterations to perform as: $I = 2 + round(log_2 p)$ with p the number of parameters (line 2). Moreover, the computational budget of each iteration i can be calculated as $budget_i = (maximum\_budget - budget_{used})/(I - i + 1)$ (line 5), with $budget_{used}$ being the budget already spent during the process.

At each iteration, N_i candidate parameter calibrations are sampled, with $N_i = \lfloor budget_i/\mu_i \rfloor$ and μ_i a factor that increases with the number of iterations (line 6). At the first iteration, all candidate calibrations are sampled uniformly at random or starting from a user defined candidates set. Each F-Race execution can be stopped if at most $N_{min} = 2 + round(log_2 p)$ candidate calibrations remain on the set. Once F-Race terminates, the best N_{elite} calibrations are selected, weighted according to their ranks, and used to update the current probability model.

For sampling each new candidate, one elite parameter calibration is chosen with a probability proportional to its weight. Then, a value is sampled for each parameter according to its sampling distribution. The sampling distribution of each parameter depends on whether it is numerical or categorical. A truncated normal distribution is used for numerical parameters and a discrete probability function is used for categorical parameters.

I-Race is also able to deal with conditional parameters. These are sampled only when they are active and their sampling models are updated only when they appear in elite calibrations.

Some important I-Race (meta-)parameters are: the number of elite candidate calibrations, the number of instances evaluated before performing the first elimination test, the type of statistical test and the confidence level (Pérez Cáceres et al., 2014). Moreover, a proper maximum computational budget should be defined to perform the tuning process.

Algorithm 2: I-Race tuning method

```
1  $Set_1 \leftarrow$ User candidates set();
2  $I = 2 + round(log_2 p)$;
3  $i \leftarrow 1$;
4  while $budget_{used} \leq maximum\_budget$ do
5      $budget_i = (maximum\_budget - budget_{used})/(I - i + 1)$;
6      $N_i = \lfloor budget_i/\mu_i \rfloor$;
7      $Set_{new} \leftarrow$ Sample($\Theta$, $\Theta_{elite}$);
8      $Set_i \leftarrow Set_{new} \cup Set_{elite}$;
9      $Set_{elite} \leftarrow$ Race($Set_i$, $budget_i$);
10     $budget_{used} = budget_{used} + budget_i$;
11     $i = i + 1$;
12 end
13 return $Set_{elite}$;
```

I-Race implementation is provided as an R package available from $CRAN$.[2] More details can be found in its website.[3]

3.3. Evolutionary Calibrator

The Evolutionary Calibrator (Evoca) (Riff and Montero, 2013) is itself an evolutionary algorithm that works with a population of parameter calibrations. Its population size is computed according to the number of parameters tuned and their domain sizes. This, in order to include a set of relevant values for each parameter in an independent way on its first population. A set of well distributed values is selected for each parameter being tuned and their values combined in calibrations using Latin hypercube design (line 1). Algorithm 3 shows the Evoca structure. Evoca uses two transformation operators. First, it implements a wheel crossover that constructs one child parameter calibration from the whole population (line 3). Crossed child replaces the worst calibration on the current population (line 5). Second, it uses a hill climbing first improvement procedure as mutation operator (line 6). Mutation tries to improve crossed child by modifying the value of one parameter. In case of a numerical parameter, it will try to randomly take a new value from the initial parameter range of values, regarding it as a continuous interval. The mutated calibration replaces the second worst parameter calibration on current population, only if it performs better than the crossed calibration (line 9).

Evoca defines three (meta-)parameters: the maximum population size, the number of repetitions to evaluate each parameter calibration (r) and the maximum computational budget. Evoca implementation is available in its authors website.[4]

[1] http://www.cs.ubc.ca/labs/beta/Projects/ParamILS.

[2] http://cran.r-project.org.

[3] http://iridia.ulb.ac.be/irace/.

[4] ecco.informaticae.org.

Algorithm 3: Evoca tuning method

```
Generate initial population ($\mathcal{P}$);
while $maximum\_budget$ is not reach do
    $crossed\_child$ ← wheel-based-crossover($\mathcal{P}$);
    evaluate($crossed\_child$, $r$);
    replace worst($\mathcal{P}$, $crossed\_child$);
    $mutated\_child$ ← hill-climbing($crossed\_child$);
    evaluate($mutated\_child$, $r$);
    if $mutated\_child$ is better than $crossed\_child$ then
        replace second worst($\mathcal{P}$, $mutated\_child$);
    end
end
return $\mathcal{P}$;
```

Table 1
Main features of tuning methods.

Tuning method		ParamILS	I-Race	Evoca
Category	Search-based	×		×
	Screening		×	
Parameter space definition	Discretized	×		
	Non-discretized		×	×
Output	Best calibration	×		
	Set of calibrations		×	×
Conditional parameters		Yes	Yes	No
Meta-parameters		4	6	3
Stop criterion		Maximum computational budget		

3.4. Discussion

This section summarizes the main features of the previously described tuning methods. Table 1 lists some important issues related to the use of tuning methods. First, it classifies tuners according to the process they perform: ParamILS and Evoca are search-based methods while I-Race is an iterative screening method. ParamILS is a local search method that works the entire tuning process with only one parameter calibration, while Evoca works with a population of parameter calibrations. I-Race also works with a set of parameter calibrations that compete and collaborate to construct models of performance of the tuned algorithm according to its parameters.

To perform the tuning process ParamILS requires the definition of a finite set of values each parameter can take, while I-Race and Evoca consider a range definition for numerical parameters. The important open research that we address is this work has been proposed by authors of ParamILS: *"... to enhance ParamILS with dedicated methods for dealing with continuous parameters that do not require discretization by the user"* (page 302 in Hutter et al. (2009)).

ParamILS has shown to be very effective finding good performing parameter values (Montero et al., 2014), but can be uninformative respect to alternative parameter calibrations and features of the space of parameters (Montero et al., 2012). I-Race and Evoca have shown good results as part of a framework for supporting the design process of metaheuristics (Montero and Riff, 2014b).

As output, ParamILS delivers the best parameter calibration found, while I-Race and Evoca deliver a set of good performing parameter calibrations.

Both ParamILS and I-Race have specially designed processes to deal with conditional parameters. On the other hand, Evoca does not differentiate these parameters.

I-Race defines 6 meta-parameters, while ParamILS defines 4 and Evoca 3. Authors recommended values are usually considered for all these parameters. All these tuning methods can consider as stopping criterion a maximum number of executions or the total time spend by the tuning process.

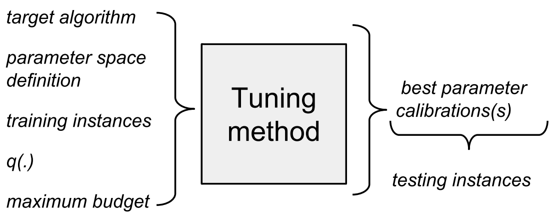


Fig. 1. Structure of tuning process. From a target algorithm, a parameter space definition, a set of training instances, a performance measure and a maximum budget; the tuner method finds a (set of) best parameter calibration(s).

4. Tuning process

Fig. 1 shows the main components of a tuning process. Regardless the tuning method used, all of them require a target algorithm to tune $\mathcal{A}$; a parameter setup file specifying the setup of parameter values to tune; a set of instances to be considered the training set and the output that usually corresponds to a set of one or more well performing parameter calibrations. Moreover, a quality measure to evaluate performance of parameter calibrations, a maximum computational budget limiting the effort invested in the tuning process and proper meta-parameter values for the tuning method must be set.

Formally, the tuning problem can be defined as follows:

Definition 1. Given a metaheuristic algorithm $\mathcal{A}$, an instance of the tuning problem consists in a 4-tuple $P = (\mathcal{A}, \Theta, \Pi, budget_{max})$, where Θ is the calibrations space for $\mathcal{A}$. Π is the set of input problem instances, $budget_{max}$ is a time out after which all runs of $\mathcal{A}$ will be terminated if they are still running.

Any calibration $\theta \in \Theta$ is a candidate calibration of P. The gain of a candidate calibration θ is given by:

$$G_P(\theta) = f_{\pi\in\Pi}(q(\theta,\pi)) \tag{1}$$

where $q(\theta,\pi)$ computes the expected gain (e.g., the quality of the solutions) of running $\mathcal{A}$ using instance $\pi \in \Pi$ when using calibration θ. Also, f evaluates the overall performance of the calibration on Π using any proper indicator as mean, number of runs solving the problem or any other

Most research related to tuning methods has been focused on the way the parameter calibrations are generated, searched or analyzed. Experimental setups are usually presented in order to obtain the best of each tuner, but little attention is paid on how to define these tuning scenarios without expert knowledge about the target algorithm.

Often, the decisions of parameter intervals and default values are taken after extensive experience on the problem: *"Having subsequently gained more experience with SAPS parameters in previous work on parameter calibration for more general problem classes, we choose promising intervals for each parameter, including but not centered at the original default"* (Hutter et al., 2009). In other cases, numerical parameters are discretized based on experience or intuition: *"The Spear scenario requires the calibration of* 26 *parameters of Spear, all of them categorical"* (Pérez Cáceres et al., 2014), where there are actually 4 integer parameters, 12 continuous parameters and 10 categorical parameters (Babić and Hutter, 2008).

In Table 2 we show example setup files required by each tuner here studied. For our example we consider as target algorithm a typical genetic algorithm that requires to tune three parameters. Two continuous and one categorical parameter: crossover rate, mutation rate and crossover operator. We can observe, that the way the parameter search spaces are defined is quite different: ParamILS requires to explicitly determine each value crossover and mutation rates can take during the tuning process, while I-Race and Evoca require only to define the range of values for each probability. In this example, for ParamILS the search

Table 2
ParamILS, I-Race and Evoca setup file examples.

Tuner	Parameter setup
ParamILS	$crossover\_rate$ in $\{0.1, 0.2, 0.3, 0.4, 0.5, 0, 6, 0.7, 0.8, 0.9, 1.0\}[0.5]$ $mutation\_rate$ in $\{0.01, 0.025, 0.05, 0.075, 0.1, 0.15, 0.20\}[0.1]$ $crossover\_operator$ in $\{uniform, one\_point, two\_point\}[uniform]$
I-Race	$crossover\_rate$ “ –cr” r $(0, 1)$ $mutation\_rate$ “ –mr” r $(0, 0.20)$ $crossover\_operator$ “ –co” c $(uniform, one\_point, two\_point)$
Evoca	$crossover\_rate$ 0.0 1.0 r $mutation\_rate$ 0.00 0.20 r $crossover\_operator$ $uniform$ one_point two_point c

for the crossover rate is done considering one decimal precision, while in I-Race the maximum number of decimal positions can be fixed by the user. Using Evoca the precision can be adjusted during the tuning process. Mutation rate is considered belonging to the range of values from 0 to 0.20 for all tuners, but ParamILS requires to define a finite set of values (7 in this case). The categorical parameter related to the crossover operator was considered in the same way by the three tuning methods. In Table 2 we also included the default value of each parameter required by ParamILS.

5. Evaluation of different parameter search space definitions on tuners

In this section we evaluate the influence of the setup of parameter search space on the capacity of tuners for obtaining good quality calibrations. For our experiments, we have selected two target algorithms:

- ACOTSP: An implementation of Ant Colony Optimization algorithms for the Traveling Salesman Problem (Stützle, 2002)
- Spear: A tree search algorithm for solving SATisfiability problems (Babić and Hutter, 2008).

Both metaheuristic algorithms define categorical and numerical parameters and a subset of conditional parameters. We will first present the results on ACOSTP and then the results on Spear.

5.1. ACOTSP algorithm

ACOTSP is a software package that implements several Ant Colony Optimization algorithms for solving the Traveling Salesman Problem. ACOTSP has 11 parameters: three categorical, four integer and four real. Categorical parameter called “algorithm” determines the ACO algorithm selected to solve the problem. Categorical parameter called “localsearch” determines the use local search and the type of local search to apply. Continuous parameters α, β and ρ are typical parameters required by all ACO algorithms. Five of the remaining parameters are conditionals on the use of local search and the ACO algorithm. Table 3 shows a list of ACOTSP parameters, their description, possible values and if they are conditional parameters, the values of parameters that activate them.

Distance to optimum is used as performance criterion for each run of this algorithm.

Benchmark instances. In our experiments, the training set was composed by ten randomly selected Euclidean traveling salesman problem instances from sets of 1000, 1500, 2000, 2500 and 3000 cities as in Hutter et al. (2009). The test set was composed of 250 TSP instances: 50 instances from each of the previously listed sets.

Tuning scenarios. For these experiments, we set a cutoff time of five seconds per execution and allowed each tuning procedure to execute the target algorithm a maximum computational budget of 5000. Table 4 summarizes the calibration scenarios.

In the following experiments, the same scenarios specifications were considered. From each tuning process the best parameter calibration was obtained. For each scenario, 20 different parameter calibrations were obtained from 20 different executions of the tuning processes using different seeds. These calibrations were then executed on the test instances.

5.1.1. Analysis of ACOTSP results

ParamILS. To define the parameter search space in ParamILS for each parameter we must define a finite set of values. In this case, we consider six different definitions of parameter search space. In each case, we select at most N evenly distributed values belonging to the domain of each parameter. These selections are applied to both numerical and categorical parameters. For these experiments the value of N ranges from 3 to 1000: ParamILS3, ParamILS5, ParamILS8, ParamILS10, ParamILS100 and ParamILS1k. We considered a minimum number of 3 values for each parameter in order to define not too small search spaces. A maximum of 1000 values was considered in order to include in the parameter search space of this scenario each possible value the parameter with the largest range of values can take. If a numerical parameter have less possible values than N, then only its maximum number of values are considered in the definition of its parameter search space.

I-Race. To analyze I-Race behavior, we evaluate two different definitions of parameter spaces. The first one considers a typical specification of each parameter according to its nature: I-RaceCatNum. CatNum stands for using categorical and numerical parameter definitions, i.e, three categorical, four integer and four real valued parameters. The second scenario considers the same parameter setup, but it also includes a file of randomly generated calibrations to start I-Race tuning process: I-RaceCatNumC. We have evaluated this scenario in order to ensure variability of initial candidate calibration sets and to perform reproducible experiments.

Evoca. One set-up of parameter space was used to evaluate Evoca: Each parameter according to its nature as Evoca specifies: EvocaCatNum.

Discussion. Graph in Fig. 2 shows the performance of each tuner in each scenario. For each scenario, the best 20 parameter calibrations, obtained from 20 different tuning processes were tested on each of 250 instances in testing set. Here we can notice that the performance obtained using ParamILS depends strongly on the setup of parameter search space considered. Table 5 summarizes the Friedman tests comparing the results of this section. Values listed here show the mean rank of the performance obtained by each tuning scenario when compared to all the others scenarios listed. Mean results over the set of instances were used to perform these tests. For a designer that uses ParamILS the best option is to use ParamILS8, i.e., a parameter search space definition that takes 8 evenly distributed values for each parameter. When the parameter definition considers large sets of values for parameters it is difficult for ParamILS to obtain good quality parameter calibrations. The same situation appears when too few values are considered.

These two I-Race versions studied show a similar performance, this because the definition of initial candidate sets of calibrations were generated randomly in both cases. Evoca version shows the best performance among all the scenarios.

It is important here to notice that Evoca does not require to invest time explicitly define the parameter search space as ParamILS does. Evoca by itself is able to conduct its tuning search to valuable regions of a large parameter search space.

5.2. Spear algorithm

Spear is a tree search algorithm for solving SAT problems. Spear is a state-of-the-art SAT solver for industrial instances, and with appropriate parameter settings, it is the best available solver for certain types of

Table 3
ACOTSP parameters.

Name	Description	Possible values	Conditioned to
Algorithm	ACOTSP algorithm	{as, mmas, eas, ras, acs}	–
Localsearch	Use of local search	{0: no local search, 1: 2-opt, 2: 2.5-opt, 3: 3-opt}	–
α	Influence of pheromone trails	$[0.01, 5.00]$	–
β	Influence of heuristic information	$[0.01, 10.00]$	–
ρ	Pheromone trail evaporation	$[0.01, 1.00]$	–
ants	Number of ants	$[5, 100]$	–
nnls	Number of nearest neighbors for local search	$[5, 50]$	Localsearch $\neq 0$
q0	Probability of best choice in tour construction	$[0.00, 1.00]$	Algorithm in {acs}
dlb	Use do not look bits in local search	$\{0, 1\}$	Localsearch $\neq 0$
rasrank	Number of ranks	$[1, 100]$	Algorithm in {ras}
Elitistants	Number of elitist ants	$[1, 750]$	Algorithm in {eas}

Table 4
Tuning scenarios.

Algorithm	Parameters	Training set	Test set	Objective	Cut off time	Maximum budget
ACOTSP	11	50	250	Quality	5 [s]	5000 runs
Spear	26	1000	1000	Runtime	5 [s]	5 [h]

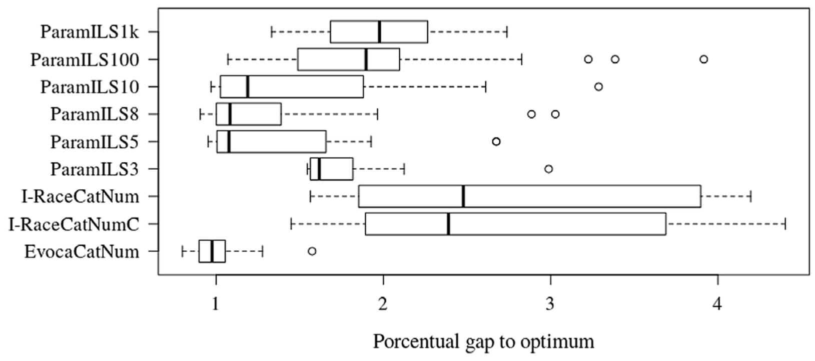


Fig. 2. Performance comparison of different parameter definitions obtained by: ParamILS, I-Race and Evoca for ACOTSP experiments. X axis shows percentage gap to optimum.

Table 5
Friedman test ACOTSP - $N = 20, \chi^2 = 88.05, df = 8, p = 0.000$.

Setting	Mean rank
EvocaCatNum	1.90
ParamILS8	3.15
ParamILS5	3.40
ParamILS10	4.00
ParamILS3	5.25
ParamILS100	5.70
ParamILS1k	6.25
I-RaceCatNumC	7.65
I-RaceCatNum	7.70

hardware and software verification instances (Hutter et al., 2007). Spear defines 26 parameters including ten categorical, four integer, and twelve continuous parameters, and their default values were hand-tuned by its developer (Babić and Hutter, 2008). The categorical parameters mainly control heuristics for variable and value selection, clause sorting, resolution ordering, and enable or disable optimizations, such as the pure literal rule. The continuous and integer parameters mainly deal with activity, decay, and elimination of variables and clauses, as well as with the interval of randomized restarts and percentage of random choices. Table 6 shows a list of the 26 Spear parameters, their description, possible values and if they are conditional parameters, the values of parameters that activate them.

Spear has been used to exhibit ParamILS capabilities. In Hutter et al. (2009) appropriate discretization decisions of integer and continuous parameters have been made: "*between three and eight discrete values spread relatively uniformly across the resulting intervals. The number of discrete values was determined according to intuition of experts about the importance of the parameters for Spear algorithm*".

Benchmark instances. We applied Spear to solve the SAT-encoded graph-coloring problems based on small world graphs: SWGCP. The set has 2000 instances, divided randomly into training and test sets of 1000 instances each one as in Hutter et al. (2009).

Tuning scenarios. In our tests using Spear, we set cutoff times of five seconds per execution. The maximum computational budget for tuning was fixed to five CPU hours. Table 4 summarizes the tuning scenarios.

5.2.1. Analysis of spear results

ParamILS. In this case, we consider eight different setup of parameter spaces. For this scenario, the value of N ranges from 3 to 10 000: ParamILS3, ParamILS5, ParamILS8, ParamILS10, ParamILS100, ParamILS1k and ParamILS10k. In this case, the maximum number of possible values a parameter of Spear can take is 10 000, hence the ParamILS10k parameter space definition considers each value this parameter can take. In our experiments, we also evaluated the performance of the parameter search space definition proposed by experienced users (Hutter et al., 2009), that considers from 3 to 8 values for numerical parameters: ParamILSAuthors.

I-Race. We evaluate three different setups of parameter space definitions. The first one considers a typical specification of each parameter according to its nature: I-RaceCatNum. The second one includes the definition of a set of randomly generated calibrations to start I-Race tuning process: I-RaceCatNumC. The third scenario considers the setup of parameter space introduced in Hutter et al. (2009) and used in the experiments performed in Pérez Cáceres et al. (2014). In this setup, all

Table 6
Spear parameters.

Name	Description	Possible values	Conditioned to
sp-var-dec-heur	Variable decision heuristic	20 possible heuristics[a]	–
sp-learned-clause-sort-heur	Heuristic for sorting learned clauses	20 possible heuristics[a]	–
sp-orig-clause-sort-heur	Heuristic for sorting original clauses	20 possible heuristics[a]	–
sp-res-order-heur	Heuristic for ordering the sequence of resolutions	20 possible heuristics[a]	sp-resolution in $\{1, 2\}$
sp-clause-del-heur	Clause deletion heuristic	$\{0, 1, 2\}$	–
sp-phase-dec-heur	Phase decision heuristic	$\{0, 1, 2, 3, 4, 5, 6\}$	–
sp-resolution	Type of resolution	$\{0, 1, 2\}$	–
sp-variable-decay	Variable activity decay	$[0.00, 10.00]$	–
sp-clause-decay	Clause activity decay	$[1.00, 10.00]$	–
sp-restart-inc	Restart period multiplier	$[1.00, 10.00]$	–
sp-learned-size-factor	Initial limit on the number of learned clauses as a percentage of the size of the number of original clauses	$[0.10, 10.00]$	–
sp-learned-clauses-inc	Multiplier of the limit of the number of learned clauses	$[1.10, 2.00]$	–
sp-clause-activity-inc	Clause activity increment	$[-10.00, 10.00]$	–
sp-var-activity-inc	Variable activity increment	$[-10.00, 10.00]$	–
sp-rand-phase-dec-freq	Random phase decision frequency	$[0.00, 1.00]$	sp-phase-dec-heur in $\{0, 1, 3, 4, 5, 6\}$
sp-rand-var-dec-freq	Random variable decision frequency	$[0.00, 1.00]$	–
sp-rand-var-dec-scaling	Random variable decision frequency scaling factor	$[0.00, 10.00]$	sp-rand-var-dec-freq $\neq 0$
sp-rand-phase-scaling	Random phase decision frequency scaling factor	$[0.00, 10.00]$	sp-rand-phase-dec-freq $\neq 0$
sp-max-res-lit-inc	Factor by which resolution is allowed to increase the number of literals	$[0.10, 10.00]$	sp-resolution in $\{1, 2\}$
sp-first-restart	Number of conflicts to the first restart	$[10, 10000]$	–
sp-res-cutoff-cls	Maximal allowed number of clauses in which a variable can occur in order to be considered for resolution	$[1, 20]$	sp-resolution in $\{1, 2\}$
sp-res-cutoff-lits	Maximal number of literals in the clause set of any phase of a variable in order to be considered for resolution	$[10, 10000]$	sp-resolution in $\{1, 2\}$
sp-max-res-runs	Max number of resolution runs	$[0, 1000]$	sp-resolution in $\{1, 2\}$
sp-update-dec-queue	Re-sort decision queue after each restart	$\{0, 1\}$	–
sp-use-pure-literal-rule	Use pure literal rule	$\{0, 1\}$	–
sp-clause-inversion	Enable inversion of learned clauses	$\{0, 1\}$	sp-learned-clause-sort-heur in $\{19\}$

[a] Prefer the most/less active/frequent/asymmetric between other heuristics.

parameters were defined as categorical and their possible values were selected according to Hutter et al. (2009): I-RaceCat.

Evoca. We consider two different setup of parameter space definitions. The first one defines each parameter according to its nature as Evoca specifies: EvocaCatNum. The second one considers all parameters as categorical taking the same set of possible values used in Hutter et al. (2009): EvocaCat.

Discussion. Graph in Fig. 3 shows the performance of each tuner in each scenario. For each scenario the best 20 parameter calibrations, obtained from 20 different tuning processes were tested on each of 1000 instances in testing set.

Clearly, we can notice that the performance obtained using ParamILS strongly depends of the setup of parameter space considered. The best results of ParamILS were obtained using the values suggested by authors in Hutter et al. (2009): ParamILSAuthors. Table 7 summarizes the Friedman tests comparing the results of this section. For a designer that uses ParamILS without an in-depth analysis the best option is to use ParamILS8 (the same as for ACOTSP). When the parameter search space grows, the parameter calibrations ParamILS performs worse compared to the ACOTSP scenario.

I-Race method also benefits from using an experienced-defined parameter search space. We can observe this behavior when numerical parameter values are discretized: I-RaceCat.

As ParamILS and I-Race, Evoca also shows a difference between the performance obtained when using a standard parameter search space definition versus using a discretized one. This confirms that using the knowledge of the specific algorithm to define the setup of the tuner strongly improves its performance. We also observe that when I-Race discretizes numerical parameter values, it obtains better parameter calibrations. These results allow us to show the relevance of the setup of parameter search space definitions for ParamILS and I-Race. In this case, the use of the parameter search space defined by Spear authors demonstrated its benefits on all tuning methods studied. The method that most benefits of using this parameter search space is ParamILS that also obtains good performance selecting between five and ten evenly distributed values.

Table 7
Friedman test spear - $N = 20, \chi^2 = 167.36, df = 12, p = 0.000$.

Setting	Mean rank
ParamILSAuthors	2.15
ParamILS8	2.90
ParamILS5	3.55
EvocaCat	4.25
ParamILS10	6.10
EvocaCatNum	6.60
ParamILS3	6.70
ParamILS100	7.55
I-RaceCat	7.95
ParamILS10k	9.35
ParamILS1k	9.70
I-RaceCatNumC	12.00
I-RaceCatNum	12.20

From the experimental analysis here performed, in next section we elaborate a set of recommendations on the use of tuners algorithms and the relevance of a proper definition of parameter search spaces.

6. Recommendations on the use of tuners

This section presents a set of recommendations to have in mind when using tuners and specially when defining parameter search spaces for tuners. We have organized our recommendations according to the time available for tuning process, the programming skills of the user/designer, its experience with the target algorithm and the goals of the algorithm. This last feature is related to its desired competitiveness and understanding. Table 8 summarizes the main features of the tuning scenarios and the suitability of the tuner.

Programming skills. For the programming skills of users we consider two categories: expert and intermediate. An expert programmer will not have any problem installing, compiling and executing any of these tuners. Each tuner requires the installation of software to be executed: ParamILS was programmed on `Ruby` and requires to install a `Ruby`

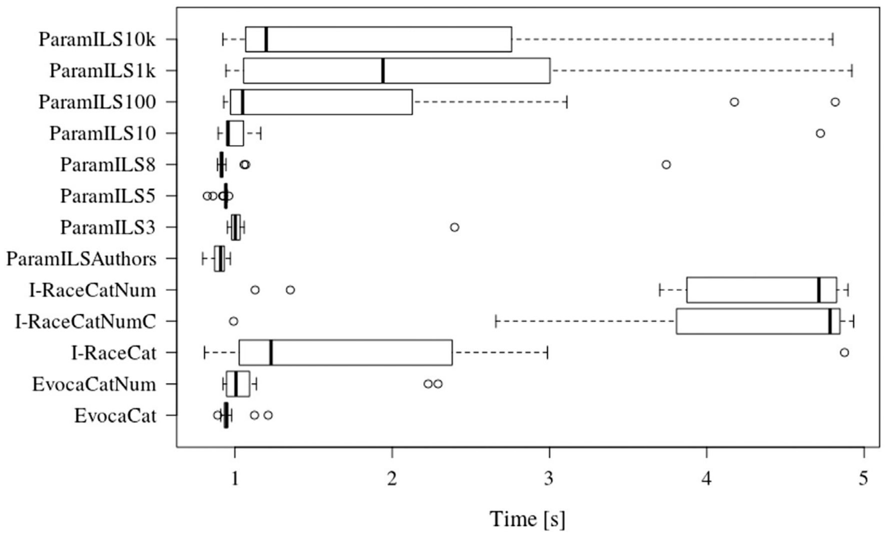


Fig. 3. Performance comparison of different parameter definitions obtained by: ParamILS, I-Race and Evoca for Spear scenario. X axis shows the time to solve a problem instance.

Table 8
Tuners suitability.

Features		ParamILS	I-Race	Evoca
Programming skills	Beginner	×		×
	Expert		×	
User experience with target algorithm	Beginner		×	×
	Expert	×		
Goals of the algorithm	Competitiveness	×		×
	Understanding		×	×
Available time to tune	Critical	×		×
	Plenty of time		×	

interpreter. Evoca was programmed in C++ and requires a C++ compiler. Most Linux systems provide these software, hence, we consider them easier to use. I-Race implementation was implemented on R and can be installed as an R package on any system. Tuning environments defined by these tuners share some key features: All require to define the set of training instances, the list of parameters to tune, their respective parameter search space and an interpreter to communicate the tuner and the target algorithm. These three tuners have support available to teach new users how to format the required files. In this sense, we recommend to use ParamILS that has a lot of good examples available and requires a simpler and more intuitive definition of required files.

User experience with the target algorithm. Experience of the user with the target algorithm is key when selecting the tuning method. An experienced user can easily determine a reduced set of values each parameter can take during the tuning process. From our results on Spear we observed that the best parameter calibrations were obtained searching on an experienced-defined parameter search space. Although this information was useful for all tuners, the most benefited with this information was ParamILS.

For a user without experience Evoca and I-Race can be a better option than ParamILS. Mainly because the definition of discrete possible values for numerical parameters is not an easy task. Evoca and I-Race work with range definitions of numerical parameter search spaces. From our experiments, the performance of Evoca demonstrated being better than the performance of I-Race, but I-Race is able to obtain more confident results because of its statistical analyses.

Goals of the tuning process. The goals of the tuning processes respect to the target algorithm are also important. In this sense we define two options: to obtain a competitive target algorithm or to understand the design of the target algorithm. A competitive metaheuristic algorithm requires only one parameter calibration that allows it to find quality results. Hence, if the objective of an experienced user is to obtain the best performing algorithm we recommend to use ParamILS, that demonstrated their capabilities finding the best parameter calibrations for Spear. If the user has no experience, Evoca can be used it is easier to set up.

On the other hand, if the goal of tuning process is to better understand the algorithm, then we recommend to use Evoca or I-Race, whose output is a set of well performing parameter calibrations. Alternative parameter calibrations provide information about the relevance of parameters values and components of the algorithm—through the tuning of categorical parameters (Montero and Riff, 2014a; Bezerra et al., 2014). Also, they can provide more informative outputs based on the set of calibrations the tuners manage in any step of the process (Montero and Riff, 2014b).

Tuning execution time. Regarding the time involved in tuning processes, ParamILS can be considered a better option when tuning target algorithms where the parameter values influence their execution times. For example, when tuning Spear algorithm, the quality of each execution was measured as the time to solve the problem instance. High quality parameter calibrations will take lower executions times, hence, ParamILS will perform faster tuning processes. On the other side, in their aim to diversify the parameter calibrations analyzed, both I-Race and Evoca spend more time evaluating low quality calibrations. I-Race uses a prediction model that can help to reduce this effect, while Evoca must evaluate all these calibrations.

Furthermore, when using ParamILS the tuning process could, eventually, being stopped in any moment and the parameter calibration delivered would always be a high quality calibration. On the other side, I-Race and Evoca require higher computational budgets to learn about the

Table 9
Performance Wilcoxon test RASON ($Z = -1.75, p = 0.079$).

Ranks		N	Mean rank	Sum of ranks
ParamILS8-ParamILS21	Negative ranks	14	10.86	152.00
	Positive ranks	6	9.67	58.00
	Ties	0		
	Total	20		

calibration being analyzed. They clearly perform diversification steps at the beginning of their tuning process, and only once these processes finish the tuners work with high quality parameter calibrations.

Next, we analyze how these recommendations can be applied on a real life case.

6.1. A real life case study

In order to evaluate if our recommendations could be used for tuning algorithms that solve real world problems, we use RASON algorithm, recently proposed in Riff et al. (2016) to solve real world scheduling radiotherapy problems.

Roughly speaking, in a scheduling radiotherapy problem, a set of patients from three different categories (urgent, palliative and radical) must be assigned to a set of treatment sessions. Each category has associated a priority. Constraints in these problems are related to a minimum number of days between sessions, the use of specific radiotherapy machines, supervision of specific specialists during the sessions, free weekends, among others.

6.1.1. RASON

RASON uses three integer parameters: $HC1$, $HC2$ and $HC3$. These parameters define the number of moves that the algorithm performs during each of its three hill climbing phases. It uses the sum of the delays of all scheduled patients as a quality criterion for each execution. In their work authors tuned RASON using ParamILS considering 21 values for each parameter. According to our results from Spear and ACOTSP our recommendation is to use ParamILS with 8 values equally spaced from each parameter range.

To evaluate the performance of each tuner set-up we used a training set of 60 instances as in Riff et al. (2016). These instances correspond to real world scenarios of radiotherapy scheduling with scheduling periods from 30 to 60 days and two different distributions of the urgent, palliative and radical patients.

We obtained 20 different parameter calibrations from 20 executions of the tuning processes using different seeds.

6.1.2. Analysis of RASON results

ParamILS. Here we compare the performance of the tuning processes performed by ParamILS using two parameter search space definitions: ParamILS21 and ParamILS8. Fig. 4 compares the performance of ParamILS8 and ParamILS21 and Table 9 gives the Wilcoxon test results. X axis shows the sum of delays of all planned patients.

Although the quality of parameter calibrations obtained from ParamILS8 parameter search space definition looks better than those obtained from the ParamILS21, there is no significant difference between their performance.

We also compared the tuning computational effort required to find these parameter calibrations. Boxplots in Fig. 5 show that ParamILS8 requires less effort than ParamILS21 to find their parameter calibrations with a median close to 350 RASON executions, while ParamILS21, is around 600 executions. A significant difference with a 95% confidence level was determined between the number of RASON executions of ParamILS8 and those of ParamILS21 (Table 10).

Moreover, we measured the execution time of each tuning process. ParamILS8 needs an average of 400 executions of RASON and 196 min to find its best parameter calibrations while ParamILS21 needs 600 runs

Table 10
Budget Wilcoxon test RASON ($Z = -1.98, p = 0.048$).

Ranks		N	Mean rank	Sum of ranks
ParamILS8-ParamILS21	Negative ranks	15	10.53	158.00
	Positive ranks	5	10.40	52.00
	Ties	0		
	Total	20		

Table 11
Friedman test collaborative approaches for ACOTSP - $N = 20, \chi^2 = 12.28, df = 4, p = 0.015$.

Setting	Mean rank
E+PILS 50%	2.30
E+PILS 90%	2.75
Evoca	2.80
E+PILS 10%	3.20
ParamILS8	3.95

and 290 min. In terms of the size of the search space, ParamILS21 has a size of 9261 parameter calibrations while ParamILS8 has a size of 512 parameter calibrations. Therefore, as expected from Section 6, reducing the number of ParamILS from 21 to 8 not only allows to obtain comparable results in term of the RASON performance, but also reduce the tuning computational cost. Clearly, ParamILS is an efficient tool for calibrating metaheuristic algorithms and its results strongly depends of its initial set-up.

6.2. Collaboration scenarios

In this section we evaluate some ideas of collaborative schemes between tuners. For this we have designed an experiment that defines the parameter search space of ParamILS based on the parameter search space obtained by Evoca. We have used the ACOSTP algorithm to perform these tests.

In order to perform fair comparisons we have designed three collaborative scenarios that divide the available budget for tuning process into the two steps:

- E+PILS 10%: Allocates 10% of the budget to Evoca and the remaining 90% to ParamILS.
- E+PILS 50%: Allocates 50% of the budget to Evoca and the remaining 50% to ParamILS.
- E+PILS 90%: Allocates 90% of the budget to Evoca and the remaining 10% to ParamILS.

Graph in Fig. 6 shows the performance comparison of these three collaborative scenarios and the isolated performance of Evoca and ParamILS. Table 11 shows the Friedman comparison of these tuning scenarios. From those is it clear that, despite the budget allocated to each step, collaborative scenarios shown always a better performance than ParamILS. Moreover, the only case when Evoca performance was better than one of these collaborative scenarios was for the E+PILS 10% version. In this case, a very low percentage of budget was allocated to Evoca step that probably does not allow it to converge to promising areas of the parameter search space.

7. Conclusions

In this work we have analyzed the behavior of three well-known tuners: I-Race, ParamILS and Evoca when using different parameter search space definitions. We have tested these three methods using different parameter search spaces and evaluated the quality of the parameter calibrations obtained for ACOTSP and Spear algorithms on these scenarios. The tests allow us to conclude that the quality of the final calibrations obtained by ParamILS and I-Race strongly depends on their initial set-up values. Furthermore, Evoca shown being less sensitive

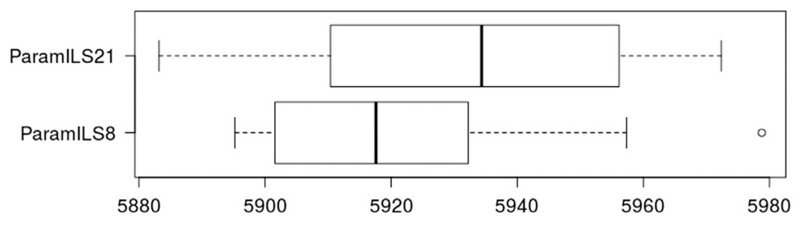


Fig. 4. Performance comparison of different parameter definitions obtained by ParamILS for RASON experiments. X axis shows the total delay of all planned patients.

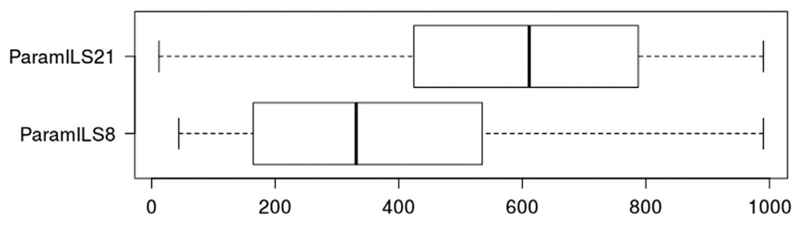


Fig. 5. Budget usage of different parameter search space definitions obtained by ParamILS for RASON experiments. X axis shows the number of RASON executions to the best parameter calibration found.

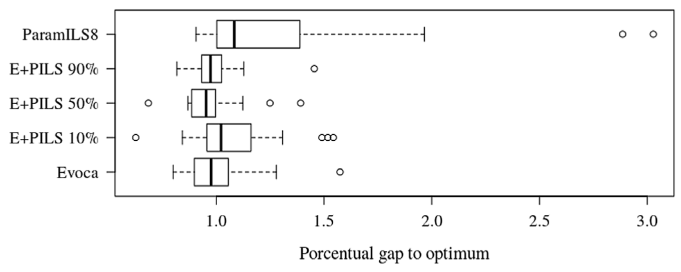


Fig. 6. Performance comparison of different parameter definitions obtained by collaboration between Evoca and ParamILS. X axis shows percentage gap to optimum.

to the initial set-up, obtaining quality parameter values without prior knowledge of the calibration space. Moreover, we also included a set of recommendations on the use of tuners an definition of proper parameter search spaces that were evaluated on a real life case. For future work we are interested in studying collaboration schemes of tuners in order to identify the best parameter calibration of metaheuristics without requiring an in-depth analysis from the user to set-up the tuners. Some preliminary experiments on the ACOSTP metaheuristic approach were also included. Moreover, some more complex hybrid approaches could be studied (Liu and Ranjithan, 2010).

Acknowledgments

Funding First and third author want to acknowledge the funding of the FONDECYT Project 11150787. Second author acknowledges the support of the FONDECYT Project 1191940 and Centro Científico Tecnológico de Valparaíso (CCTVal) FB0821.

References

Adenso-Diaz, B., Laguna, M., 2006. Fine-tuning of algorithms using fractional experimental designs and local search. Oper. Res. 54 (1), 99–114.
Augusto, O., Rabeau, S., Dépincé, P., Bennis, F., 2006. Multi-objective genetic algorithms: A way to improve the convergence rate. Eng. Appl. Artif. Intell. 19 (5), 501–510. http://dx.doi.org/10.1016/j.engappai.2006.01.010.
Babić, D., Hutter, F., 2008. Spear theorem prover. In: Proceedings of the SAT 2008 Race.
Balaprakash, P., Birattari, M., Stützle, T., 2007. Improvement strategies for the F-Race algorithm: sampling design and iterative refinement. In: Proceedings of the 4th International Conference on Hybrid Metaheuristics. Springer-Verlag, Berlin, Heidelberg, pp. 108–122.
Bartz-Beielstein, T., Flasch, O., Zaefferer, M., 2012. Sequential parameter optimization for symbolic regression. In: Genetic and Evolutionary Computation Conference, GECCO '12, Philadelphia, PA, USA, July 7-11, 2012, Companion Material Proceedings, pp. 495–496. http://dx.doi.org/10.1145/2330784.2330861.
Bezerra, L.C.T., López-Ibáñez, M., Stützle, T., 2014. Automatic design of evolutionary algorithms for multi-objective combinatorial optimization. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (Eds.), Parallel Problem Solving from Nature–PPSN XIII: 13th International Conference, Ljubljana, Slovenia, September 13-17, 2014. Proceedings. Springer International Publishing, Cham, pp. 508–517.
Birattari, M., Stützle, T., Paquete, L., Varrentrapp, K., 2002. A racing algorithm for configuring metaheuristics. In: Proceedings of the Genetic and Evolutionary Computation Conference. Morgan Kaufmann, pp. 11–18.
Birattari, M., Yuan, Z., Balaprakash, P., Stützle, T., 2010. F-Race and iterated F-race: An overview. In: Bartz-Beielstein, T., Chiarandini, M., Paquete, L., Preuss, M. (Eds.), Experimental Methods for the Analysis of Optimization Algorithms. Springer Berlin Heidelberg, pp. 311–336.
Blot, A., Hoos, H.H., Jourdan, L., Kessaci-Marmion, M.-É., Trautmann, H., 2016. MO-ParamILS: A Multi-objective automatic algorithm configuration framework. In: Festa, P., Sellmann, M., Vanschoren, J. (Eds.), Learning and Intelligent Optimization: 10th International Conference, LION 10, Ischia, Italy, May 29 –June 1, 2016, Revised Selected Papers. Springer International Publishing, Cham, pp. 32–47.
Blot, A., Pernet, A., Jourdan, L., Kessaci-Marmion, M.-E., Hoos, H.H., 2017. Automatically configuring multi-objective local search using multi-objective optimisation. In: 9th International Conference on Evolutionary Multi-Criterion Optimization - Volume 10173, EMO 2017. Springer-Verlag New York, Inc., New York, NY, USA, pp. 61–76.
De Jong, K., 1975. An Analysis of the Behaviour of a Class of Genetic Adaptive Systems. (Ph.D. thesis), University of Michigan.
Eiben, A.E., Smit, S.K., 2011. Parameter tuning for configuring and analyzing evolutionary algorithms. Swarm Evol. Comput. 1 (1), 19–31.
Grefenstette, J., 1986. Optimization of control parameters for genetic algorithms. IEEE Trans. Syst. Man Cybern. 16 (1), 122–128.
Hutter, F., Babic, D., Hoos, H.H., Hu, A.J., 2007. Boosting verification by automatic tuning of decision procedures. In: Proceedings of the Formal Methods in Computer Aided Design. IEEE Computer Society, Washington, DC, USA, pp. 27–34.
Hutter, F., Hoos, H.H., Leyton-Brown, K., 2011. Sequential model-based optimization for general algorithm configuration. In: Coello Coello, C.A. (Ed.), Learning and Intelligent Optimization. In: Lecture Notes in Computer Science, vol. 6683, Springer Berlin Heidelberg, pp. 507–523.
Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T., 2009. ParamILS: An automatic algorithm configuration framework. J. Artificial Intelligence Res. 36, 267–306.

Liu, L., Ranjithan, S.R., 2010. An adaptive optimization technique for dynamic environments. Eng. Appl. Artif. Intell. 23 (5), 772–779. http://dx.doi.org/10.1016/j.engappai.2010.01.007. advances in metaheuristics for hard optimization: new trends and case studies.

López-Ibáñez, M., Cáceres, J.D.-L.L.P., Stützle, M.B.T., Birattari, M., 2016. The irace package: Iterated racing for automatic algorithm configuration. Oper. Res. Perspect. 3, 43–58. http://dx.doi.org/10.1016/j.orp.2016.09.002.

Montero, E., Riff, M.-C., 2014a. Towards a method for automatic algorithm configuration: a design evaluation using tuners. In: Bartz-Beielstein, T., Branke, J., Filipič, B., Smith, J. (Eds.), Proceedings of the Parallel Problem Solving from Nature–PPSN XIII: 13th International Conferenference. Springer International Publishing, Cham, pp. 90–99.

Montero, E., Riff, M.-C., 2014b. Towards a method for automatic algorithm configuration: a design evaluation using tuners. In: Parallel Problem Solving from Nature - PPSN XIII. In: Lecture Notes in Computer Science, vol. 8672, Springer International Publishing, pp. 90–99.

Montero, E., Riff, M.-C., Neveu, B., 2014. A beginner's guide to tuning methods. Appl. Soft Comput. 17, 39–51.

Montero, E., Riff, M.-C., Pérez-Caceres, L., Coello Coello, C.A., 2012. Are state-of-the-art fine-tuning algorithms able to detect a dummy parameter?. In: Parallel Problem Solving from Nature - PPSN XII. In: Lecture Notes in Computer Science, vol. 7491, Springer Berlin Heidelberg, pp. 306–315.

Pérez Cáceres, L., López-Ibañez, M., Stützle, T., 2014. An analysis of parameters of irace. In: Blum, C., Ochoa, G. (Eds.), Evolutionary Computation in Combinatorial Optimisation. In: Lecture Notes in Computer Science, vol. 8600, Springer Berlin Heidelberg, pp. 37–48.

Radulescu, A., López-Ibáñez, M., Stützle, T., 2013. Automatically improving the anytime behaviour of multiobjective evolutionary algorithms. In: Purshouse, R.C., Fleming, P.J., Fonseca, C.M., Greco, S., Shaw, J. (Eds.), Evolutionary Multi-Criterion Optimization: 7th International Conference, EMO 2013, Sheffield, UK, March (2013) 19-22. Proceedings. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 825–840.

Riff, M.-C., Cares, J.P., Neveu, B., 2016. RASON: A new approach to the scheduling radiotherapy problem that considers the current waiting times. Expert Syst. Appl. 64, 287–295. http://dx.doi.org/10.1016/j.eswa.2016.07.045.

Riff, M.-C., Montero, E., 2013. A new algorithm for reducing metaheuristic design effort, In: IEEE Congress on Evolutionary Computation (CEC 2013), Cancún, México, pp. 3283–3290.

Smit, S.K., Eiben, A.E., Szlávik, Z., 2010. An MOEA-based method to tune EA parameters on multiple objective functions. In: Filipe, J., Kacprzyk, J. (Eds.), IJCCI (ICEC). SciTePress, pp. 261–268.

Stützle, T., 2002. ACOTSP: a software package of various ant colony optimization applied to the symetric salesman problem, http://iridia.ulb.ac.be/mdorigo/ACO/aco-code/.

Ugolotti, R., Cagnoni, S., 2014. Analysis of evolutionary algorithms us- ing multi-objective parameter tuning. In: Proceedings of the 302014 Annual Conference on Genetic and Evolutionary Computation, GECCO 14, Vol. 114. ACM, New York, NY, USA, pp. 1343–1350. http://dx.doi.org/10.1145/2576768.2598226.

Zhang, T., Georgiopoulos, M., Anagnostopoulos, G.C., 2015. SPRINT multi-objective model racing. In: Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation, GECCO '15. ACM, New York, NY, USA, pp. 1383–1390.