Tools for the Automatic Tuning of Parameterized Algorithms

D.E.A. dissertation by

Zhi Yuan

supervised by:

Thomas Stützle
Mauro Birattari

September 13, 2010

Abstract

Algorithms for solving hard optimization problems typically have several parameters that need to be set appropriately such that some aspect of performance is optimized. In this thesis, we review *F-Race*, a racing algorithm for the task of automatic algorithm configuration. *F-Race* is based on a statistical approach for selecting the best configuration out of a set of candidate configurations under stochastic evaluations. We review the ideas underlying this technique and discuss an extension of the initial *F-Race* algorithm, which leads to a family of algorithms that we call iterated *F-Race*. Experimental results comparing one specific implementation of iterated *F-Race* to the original *F-Race* algorithm confirm the potential of this family of algorithms. The hybrid of *F-Race* with two other continuous optimization algorithm also enriches the family of iterated *F-Race*.

Contents

Chapter 1

Introduction

Many state-of-the-art algorithms for tackling computationally hard problems have a number of parameters that influence their search behavior. Such algorithms include exact algorithms such as branch-and-bound algorithms, algorithm packages for integer programming, and approximate algorithms such as stochastic local search (SLS) algorithms. The parameters can roughly be classified into numerical and categorical parameters. Examples of numerical parameters are the tabu tenure in tabu search algorithms or the pheromone evaporation rate in ant colony optmization (ACO) algorithms. Additionally, many algorithms can be seen as being composed of a set of specific components that are often interchangeable. Examples are different branching strategies in branch-and-bound algorithms, different types of crossover operators in evolutionary algorithms, or different types of local search algorithms in iterated local search. These interchangable components are often well described as categorical parameters of the underlying search method.

Research has clearly shown that the performance of parameterized algorithms depends strongly on the particular values of the parameters and the choice of an appropriate setting of these parameters is itself a difficult optimization problem [1, 15, 17]. Given that typically not only the setting of numerical parameters but also that of categorical parameters needs to be determined, we call this problem also the *algorithm configuration problem*. An important aspect of this problem is that it is typically a stochastic problem. In fact, there are two main sources of stochasticity. The first is that often the algorithm itself is stochastic because it uses some randomized decisions during the search. In fact, this stochasticity is typical for SLS algorithms [31]. However, even if an algorithm is deterministic, its performance and search behavior depends on the particular instance to which it is applied. In fact, the particular instance being tackled can be seen as having been drawn according to some underlying, possibly unknown probability distribution,

introducing in this way a second stochastic factor.

In our research, we have developed a method, called *F-Race*, which is particularly well suited for dealing with this stochastic aspect. It is a method that is inspired from racing algorithms in machine learning, in particular Hoeffding races [37, 38, 39]. The essential idea of racing methods, in general, and ours, in particular, is to evaluate a given set of candidate configurations iteratively on a stream of instances. As soon as enough statistical evidence is gathered against some candidate configurations, these are eliminated and the race continues only with the surviving ones. In our case, this method uses after each evaluation round of the candidate configurations, the non-parametric Friedman test as a family-wise test: it checks whether there is evidence that at least one of the configurations is significantly different from others. If the null hypothesis of no differences is rejected, Friedman post-tests are applied to eliminate those candidate configurations that are significantly worse than the best one.

In this article, we first formally describe the algorithm configuration problem, following [17, 15]. Next, in Section 3, we give details on *F-Race*. Section 4 discusses considerations on the sampling of candidate configurations, proposes a family of iterated *F-Race* algorithms, and defines one specific iterated *F-Race* algorithm, which extends over an earlier version published in [7]. Computational results with this new variant, which are presented in Section 4.5, confirm its advantage over other ways of generating the candidate configurations for *F-Race*. Also presented are the hybrids of *F-Race* with two state-of-the-art continuous optimization algorithms, i.e. mesh adaptive direct search [3] and BOBYQA [48], which also give promising results. We end this article by an overview of available *F-Race* applications and outline ideas for further research.

Chapter 2

The algorithm configuration problem

F-Race is a method for the offline configuration of parameterized algorithms. In the *training phase* of offline tuning, an algorithm configuration is to be determined in a limited amount of time that optimizes some measure of algorithm performance. The final algorithm configuration is then deployed in a *production phase* where the algorithm is used to solve previously unseen instances.

A crucial aspect of this *algorithm configuration problem* is that it is a problem of generalization, as it occurs in other fields such as machine learning. Based on a given set of training instances, the goal is to find high-perfoming algorithm configurations that perform well on (a potentially infinite set of) unseen instances that are not available when deciding on the algorithm's parameters. Hence, one assumption that is tacitly made is that the set of training instances is representative for the instances the algorithm faces once it is employed in the production phase. The notions of best performance, generalization, etc. are made explicit in the formal definition of the algorithm configuration problem.

2.1 The algorithm configuration problem

The problem of configuring a parameterized algorithm can be formally defined as a 7 tuple $\langle \Theta, I, P_I, P_C, t, \mathcal{C}, T \rangle$, where

- Θ is the possibly infinite set of candidate configurations.
- I is the possibly infinite set of instances.

- P_I is a probability measure over the set I.
- $t : I \to$ is a function associating to every instance the computation time that is allocated to it.
- $c(\theta, i, t(i))$ is a random variable representing the cost measure of a configuration $\theta \in \Theta$ on instance $i \in I$ when run for computation time $t(i)$.[1]
- $C \subset$ is the range of c, that is, the possible values for the cost measure of the configuration $\theta \in \Theta$ on an instance $i \in I$.
- P_C is a probability measure over the set C: With the notation $P_C(c|\theta, i)$, we indicate the probability that c is the cost of running configuration θ on instance i.
- $\mathcal{C}(\theta) = \mathcal{C}(\theta|\Theta, I, P_I, P_C, t)$ is the criterion that needs to be optimized with respect to θ. In the most general case it measures in some sense the desirability of θ.
- T is the total amount of time available for experimenting with the given candidate configurations on the available instances before delivering the selected configuration.[2]

On the basis of these concepts, solving the problem of configuring a parameterized algorithm is to find the configuration $\bar{\theta}$ such that:

$$\bar{\theta} = \arg\min_{\theta \in \Theta} \mathcal{C}(\theta). \tag{2.1}$$

Throughout the whole chapter, we consider for $\mathcal{C}$ the expected value of the cost measure c:

$$\mathcal{C}(\theta) = E_{I,C}[c] = \int c \, \mathrm{d}P_C(c|\theta, i) \, \mathrm{d}P_I(i), \tag{2.2}$$

where the expectation is considered with respect to both P_I and P_C, and the integration is taken in the Lebesgue sense [11]. However, other options for defining the cost measure to be minimized such as the median cost or a percentile of the cost distribution are easily conceivable.

[1] To make the notation lighter, in the following we often will not mention the dependence of the cost measure on $t(i)$. We use the term cost to refer, without loss of generality, to the minimization of some performance measure such as the objective function value in a minimization problem or the computation time taken for a decision problem instance.

[2] In the following, we refer to T also as *computational budget*; often it will be measured as the number of algorithm runs instead of a total amount of computation time.

The measures P_I and P_C are usually not explicitly available and the analytical solution of the integrals in Eq. 2.2 is not possible. In order to overcome this limitation, the expected cost can be estimated in a Monte Carlo fashion on the basis of running the particular algorithm configuration on a training set of instances.

The cost measure c in Eq. 2.2 can be defined in various ways. For example, the cost of a configuration θ on an instance i can be measured by the objective function value of the best solution found in a given computation time $t(i)$. In such a case, the task is to tune algorithms for an optimization problem and the goal is to optimize the solution quality reached within a given computation time. In the case of decision problems, the goal is rather to choose parameter settings such that the computation time to arrive at a decision is minimized. In this case, the cost measure would be the computation time taken by an algorithm configuration to decide an instance i. Since arriving at a decision may take infeasibly long computation times, the role played by the function t is to give a maximum computation time budget for the execution of the algorithm configuration. If after a cutoff time of $t(i)$ the algorithm has not finished, the cost measure may use additional penalties [33]. Finally, let us remark that the definition of the algorithm configuration problem applies not only to the configuration of stochastic algorithms, but it extends also to deterministic, parameterized algorithm: in this case, $c(\theta, i, t(i))$ is strictly speaking not anymore a random variable but a deterministic function; the stochasticity is then due to the instance distribution P_I.

One basic question concerns how many times a configuration should be evaluated on each of the available problem instances for estimating the expected cost. Assuming that the performance of a stochastic algorithm is evaluated by a total of N runs, it has been proved by Birattari [13, 15], that sampling N instances with one run on each instance results in the lowest variance of the estimator. Hence, it is always preferable to have a large set of training instances available. If, however, only few training instances are provided, one needs to go back to evaluating algorithm configurations on the instances more than once.

2.2 Types of parameters

As said in the introduction, algorithms can have different types of parameters. There we have distinguished between *categorical* and *numerical* parameters. Categorical parameters typically refer to different procedures or discrete choices that can be taken by an algorithm (or, more in general, an

algorithm framework such as a metaheuristic). In SLS algorithms examples are the type of perturbations and the particular local search algorithm used in iterated local search (ILS) or the type of neighborhood structure to be used in iterative improvement algorithms. Sometimes it is possible to order the categories of these categorical parameter according to some surrogate measure. For example, neighborhoods may be ordered according to their size or crossover operators in genetic algorithms according to the disruptedness they introduce. Hence, sometimes categorical parameters can be converted into ordinal ones. (We are, however, not aware of configuration methods that exploited this possibility so far.) Categorical parameters that may be ordered based on secondary criteria, we call `pseudo-ordinal` parameters.[3]

Besides categorical parameters, numerical parameters are common in many algorithms. `Continuous` numerical parameters take as values some subset of the real numbers. Examples of these are the pheromone evaporation rate in ACO, or the cooling rate in simulated annealing. Often, numerical parameters take integer values; an example is the strength of a perturbation that is measured by the number of solution components that change. If such parameters have a relatively large domain, they may be treated in the configuration task as continuous parameters, which are then rounded to the next integer. In the following we call such integer parameters `quasi-continuous` parameters.

Furthermore, it is often the case that some parameter is only in effect when another parameter, usually a categorical one, takes certain values. This is the case of a `conditional` parameter. An example can be given in ILS, where as one option a tabu search may be used as the local search; in this case, the tabu list length parameter is a conditional parameter that depends on whether a categorical parameter "type of local search" indicates that tabu search is used.[4] The *F-Race* based configuration algorithms described in this chapter are able to handle all aforementioned types of parameters, including conditional parameters.

[3]Note that, strictly speaking, binary parameters are also ordinal ones, although they are usually handled without considering an ordering.

[4]It is worth noticing that sometimes it may make sense to replace a numerical parameter by a categorical parameter plus a conditional parameter, if changing the numerical parameter may lead to drastic changes in design choices of an algorithm. Consider as an example the probability of applying a crossover operator. This parameter may take a value of zero, which indicates actually that no crossover is applied. In such cases it may be useful to introduce a binary parameter, which indicates whether crossover is used or not, together with a conditional parameter on the crossover probability, which is only used if the binary parameter indicates that crossover is used.

Chapter 3

F-Race

The conceptually simplest approach for estimating the expected cost of an algorithm configuration θ, as defined by Eq. 2.2, is to run the algorithm using a sufficiently large number of instances. This estimation can be repeated for a number of candidate configurations and once the overall computational budget allocated for the selection process is consumed, the candidate configuration with the lowest estimate is chosen as the best performing configuration. This is an example of what can be characterized as the *brute-force approach* to algorithm configuration.

There are two main problems associated with this brute-force approach. The first is that one needs to determine *a priori* how often a candidate configuration is evaluated. The second is that also poor performing candidate configurations are evaluated with the same amount of computational resources as the good ones.

3.1 The racing approach

As one possibility to avoid the disadvantages of the brute-force approach we have used a racing approach. The racing approach originated from the machine learning community [38], where it was first proposed for solving the model selection problem [19]. We adapted this approach to make it suitable for the algorithm configuration task. The racing approach performs the evaluation of a finite set of candidate configurations using a systematic way to allocate the computational resources among them. The racing algorithm evaluates a given finite set of candidate configurations `step` by `step`. At each `step`, all the remaining candidate configurations are evaluated in paral-

lel,[1] and the poor candidate configurations are discarded as soon as sufficient statistical evidence is gathered against them. The elimination of the poor candidates allows to focus the computations on the most promising ones to obtain lower variance estimates for these. In this way, the racing approach overcomes the two major drawbacks of the brute-force approach. First, it does not require a fixed number of steps for each candidate configuration but it determines it adaptively based on statistical evidence. Second, poor performing candidates will not be evaluated as soon as enough evidence is gathered against them. A graphical illustration of the *racing* algorithm and the *brute-force* approach is shown in Fig. 3.1.

To describe the racing approach formally, suppose a sequence of training instances i_k, with $k = 1, 2, \ldots$, is randomly generated from the target class of instances I following the probability model P_I. Denote by c_k^θ the cost of a single run of a candidate configuration θ on instance i_k. The evaluation of the candidate configurations is performed incrementally such that at the k-th `step`, the array of observations for evaluating θ,

$$\mathsf{c}^k(\theta) = \left(c_1^\theta, c_2^\theta, \ldots, c_k^\theta\right),$$

is obtained by appending c_k^θ to the end of the array $\mathsf{c}^{k-1}(\theta)$. A *racing* algorithm then generates a sequence of nested sets of candidate configurations

$$\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \ldots,$$

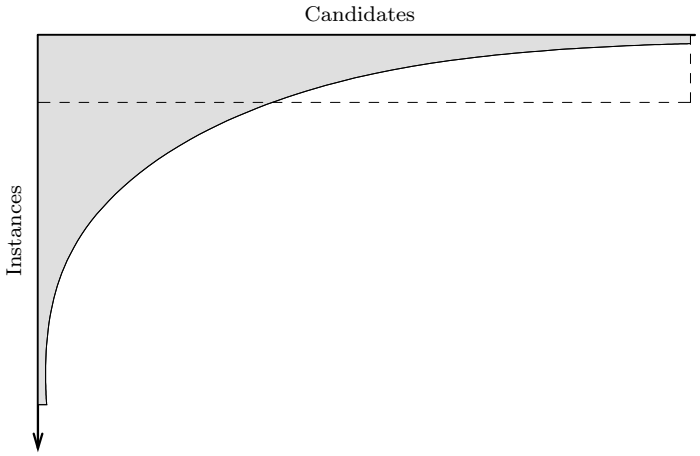
where Θ_k is the set of the surviving candidate configurations after step k. The sets of surviving candidate configurations start from a finite set $\Theta_0 \subseteq \Theta$, which is typically obtained by sampling $|\Theta_0|$ candidate configurations from Θ. How the initial set of candidate configurations can be generated is the topic of Section 4. The step from a set Θ_{k-1} to Θ_k is obtained by possibly discarding some configurations that appear to be suboptimal on the basis of information that becomes available at step k.

At step k, when the set of the surviving candidates is Θ_{k-1}, a new instance i_k is considered. Each candidate $\theta \in \Theta_{k-1}$ is tested on i_k and each observed cost c_k^θ is appended to the respective array $\mathsf{c}^{k-1}(\theta)$ to form the arrays $\mathsf{c}^k(\theta)$ for each $\theta \in \Theta_{k-1}$. Step k terminates defining set Θ_k by dropping from Θ_{k-1} the candidate configurations that appear to be suboptimal based on some statistical test that compares the arrays $\mathsf{c}^k(\theta)$ for all $\theta \in \Theta_{k-1}$.

The above described procedure is iterated and stops either when all candidate configurations but one are discarded, a given maximum number of

[1] A round of function evaluations of surviving candidate configurations on a certain instance is called an `evaluation step`, or, simply, a `step`. By `function evaluation`, we refer to one run of the candidate configuration on one instance.

Figure 3.1: Graphical representation of the allocation of configuration evaluations by the *racing* approach and the *brute-force* approach. In the *racing* approach, as soon as sufficient evidence is gathered that a candidate is sub-optimal, such candidate is discarded from further evaluation. As the evaluation proceeds, the *racing* approach focuses thus more and more on the most promising candidates. On the other hand, the *brute-force* approach tests all given candidates on the same number of instances. The shadowed figure represents the computation performed by the *racing* approach, while the dashed rectangle the one of the *brute-force* approach. The two figures cover the same surface, that is, the two approaches are allowed to perform the same total number of experiments.



instances have been sampled, or when the predefined computational budget B has been exhausted.[2]

3.2 The peculiarity of *F-Race*

F-Race is a racing algorithm based on the non-parametric Friedman's two-way analysis of variance by ranks [25], for short, Friedman test. This algorithm was first proposed by Birattari et al. [17] and studied in detail in Birattari's PhD thesis [14].

[2]The computational budget may be measured as a total available computation time T (see definition of the configuration problem on page 5). It is, however, often more convenient to define the maximum number of function evaluations, if each function evaluation is limited to a same amount of computation time.

To describe *F-Race*, assume it has reached step k, and $m = |\Theta_{k-1}|$ candidate configurations are still in the race. The Friedman test assumes that the observed costs are k mutually independent m-variate random variables

$$\begin{array}{rcccc} b_1 & = & \left(c_1^{\theta_{v_1}}, & c_1^{\theta_{v_2}}, & \ldots, \quad c_1^{\theta_{v_m}}\right) \\ b_2 & = & \left(c_2^{\theta_{v_1}}, & c_2^{\theta_{v_2}}, & \ldots, \quad c_2^{\theta_{v_m}}\right) \\ \vdots & & \vdots & \vdots & \ddots \quad \vdots \\ b_k & = & \left(c_k^{\theta_{v_1}}, & c_k^{\theta_{v_2}}, & \ldots, \quad c_k^{\theta_{v_m}}\right) \end{array}$$

called blocks, where each block b_l corresponds to the computational results obtained on instance i_l by each surviving configuration at step k.

Within each block, the costs c_l^θ are ranked in non-decreasing order; average ranks are used in case of ties. For each configuration $\theta_{v_j} \in \Theta_{k-1}$, R_{lj} is the rank of θ_{v_j} in block b_l, and $R_j = \sum_{l=1}^{k} R_{lj}$ is the sum of ranks for configuration θ_{v_j}, over all instances i_l, with $1 \leq l \leq k$. The test statistic used by the Friedman test is the following [25]:

$$T = \frac{(m-1)\sum_{j=1}^{m}\left(R_j - \frac{k(m+1)}{2}\right)^2}{\sum_{l=1}^{k}\sum_{j=1}^{m} R_{lj}^2 - \frac{km(m+1)^2}{4}}.$$

Under the null hypothesis that all candidates are equivalent, T is approximately χ^2 distributed with $m-1$ degrees of freedom [44]. If the observed value of T is larger than the $1-\alpha$ quantile of this distribution, the null hypothesis is rejected. This indicates that at least one candidate configuration gives better performance than at least one of the others.

If the null hypothesis is rejected in this *family-wise* test, it is justified to do pairwise comparisons between individual candidates. There are various ways of conducting these Friedman *post hoc tests*. For *F-Race*, we have chosen one particular one that is presented in the book of Conover [25]: candidates θ_j and θ_h are considered to be statistically significantly different if

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k\left(1-\frac{T}{k(m-1)}\right)\left(\sum_{l=1}^{k}\sum_{j=1}^{m} R_{lj}^2 - \frac{km(m+1)^2}{4}\right)}{(k-1)(m-1)}}} > t_{1-\alpha/2},$$

where $t_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the Student's t distribution.

If *F-Race* does not reject at `step` k the null hypothesis of the family-wise comparison, all candidate configurations in Θ_{k-1} pass to Θ_k; if the null hypothesis is rejected, pairwise comparisons are performed between the best candidate configuration and each other one. The best candidate configuration is selected as the one that has the lowest expected rank. All candidate configurations that result significantly worse than the best one are discarded and will not appear in Θ_k.

When only two candidates remain in the race, the Friedman test reduces to the *binomial sign test for two dependent samples* [52]. However, in the *F-Race* algorithm, the *Wilcoxon matched-pairs signed-ranks test* [25] is adopted, for the reason that the Wilcoxon test is more powerful and data-efficient than the binomial sign test in such a case [53].

In *F-Race*, the test statistic is based on the ranking of the candidates. Ranking plays an important two-fold role. The first one is due to the non-parametric nature of a test based on ranking. A second role played by ranking in *F-Race* is to implement in a natural way a blocking design [26, 40]. By focusing only on the ranking of the different configurations within each instance, this blocking design becomes an effective way for normalizing the costs observed on different instances.

Chapter 4

The sampling strategy for *F-Race*

In the previous section, the question, how the set of candidate configurations Θ_0 is defined, was left open. This is the question we address in this section; in fact, it should be clear that this question is rather independent of the definition of *F-Race*: any reasonable sampling method may be considered.

4.1 Full factorial design

When *F-Race* was first proposed [17], the candidate configurations were collected by a full factorial design (`FFD`) on the parameter space. The reason of adopting a full factorial design at that time was that it made more convenient the focus on the evaluation of *F-Race* and its comparison to other ways of defining races.

A full factorial design can be done by determining for each parameter a number of levels either manually, randomly or in some other way. Then, each possible combination of these levels represents a unique configuration, and Θ_0 comprises all possible combinations. One main drawback of a full factorial design is that it requires expertise to select the levels of each parameter. Maybe more importantly, the set of candidate configurations grows exponentially with the number of parameters. Suppose that d is the dimension of the parameter space and that each dimension has l levels; then the total number of candidate configurations would be l^d. It therefore quickly becomes impractical and computationally prohibitive to test all possible combinations even for a reasonable number of levels at each dimension. We denote the version of *F-Race* using a full factorial design by *F-Race(FFD)*.

4.2 Random sampling design

The drawbacks of the full factorial design were described also by Balaprakash et al. [7]. They showed that *F-Race* with initial candidates generated by a random sampling design significantly outperforms the full factorial design for a number of applications. In the random sampling design, the initial elements are sampled according to some probability model P_X defined over the parameter space X.[1] If *a priori* information is available, such as the effects of certain parameters or their interactions, the probability model P_X can be defined accordingly. However, this is rarely the case, and the default way of defining the probability model P_X is to assume a uniform distribution over X. We denote the random sampling version of *F-Race* based on uniform distributions *F-Race(RSD)*.

Two main advantages of the random sampling design are that for numerical parameters, no *a priori* definition of the levels needs to be done and that an arbitrary number of candidate configurations can be sampled while still covering the parameter space, on average, uniformly.

4.3 Iterated *F-Race*

As a next step, Balaprakash et al. [7] proposed the iterative application of *F-Race*, where at each iteration a number of surviving candidate configurations of the previous iteration bias the sampling of new candidate configurations. It is hoped in this way to focus the sampling of candidate configurations around the most promising ones. In this sense, iterated *F-Race* follows directly the framework of model-based search [61], which is usually implemented in three steps. First, construct a candidate solution based on some probability model; second, evaluate all candidates; third, update the probability model biasing the next sampling towards the better candidate solutions. These three steps are iterated, until some termination criterion is satisfied.

Iterated *F-Race* proceeds in a number of `iterations`. In each iteration, first a set of candidate configurations is sampled; this is followed by one run of *F-Race* applied to the sampled candidate configurations. An outline of the general framework of iterated *F-Race* is given in Alg. 4.3.

There are many possible ways how iterated *F-Race* can be implemented. In fact, one possibility would be to use some algorithms for black-box mixed

[1] Note that the space of possible parameter value combinations X is different from the one-dimensional vector of candidate algorithm configurations Θ, and there exists a one-to-one mapping from X to Θ.

Algorithm 4.1 Iterated *F-Race*

Require: parameter space X, a noisy objective function black-box f.
  initialize probability model P_X for sampling from X
  set iteration counter $l = 1$
  repeat
    sample the initial set of configurations Θ_0^l based on P_X
    evaluate set Θ_0^l by f using *F-Race*
    collect elite configurations from *F-Race* to update P_X
    $l = l + 1$
  until termination criterion is met
  identify the best parameter configuration x^*
  return x^*

discrete-continuous optimization problems. However, a difficulty here may be that for *F-Race* to be effective, the number of candidate configurations should be reasonably large, while due to the necessarily strongly limited number of `function evaluations`, few iterations should be run. Therefore, in [7] a different approach was followed and an ad-hoc method was proposed for biasing the sampling. Unfortunately, the ad-hoc iterated *F-Race* was there only defined and tested on numerical parameters. Nevertheless, it is relatively straightforward to generalize the ideas presented there to categorical parameters. In what follows, we first give a general discussion of the issues that arise in the definition of an iterated *F-Race* algorithm and then we present one particular implementation in Section 4.4. For the following discussion, we assume that the total computational budget B for the configuration process, which is measured by the number of function evaluations, is given *a priori*.

How many iterations? Iterated *F-Race* is an iterative process and therefore one needs to define the number of iterations. For a given computational budget, using few iterations will allow to sample at each iteration more candidate configurations and, hence, lead to more exploration at the cost of less possibilities of refining the model. In the extreme case of using only one iteration, this amounts to an execution of *F-Race(RSD)*. Intuitively, the number of iterations should depend on the number of parameters: if only few parameters are present, we expect, others things being equal, the problem to be less difficult to optimize and, hence, less iterations to be required.

Which computational budget at each iteration? Another issue concerns the distribution of the computational budget B among the iterations.

The simplest idea is to divide the computational budget equally among all iterations. However, other possibilities are certainly reasonable; for example, one may decrease the number of function evaluations available with an increase of the iteration counter to increase exploration in the first iterations.

How many candidate configurations at each iteration? For *F-Race*, the number of candidate configurations to be sampled needs to be defined. A good idea is to make the number of candidate configurations dependent on the status of the race, in other words, the iteration counter. Typically, in the first iteration(s), the sampled candidate configurations are very different from each other, resulting in large performance differences. As a side effect, poor candidate configurations usually can be quickly eliminated. In later iterations, the sampled candidate configurations become more similar and it becomes more difficult to determine the winner, that is, more instances are needed to detect significant differences among the configurations. Hence, for a same budget of function evaluations for one application of *F-Race*, in early iterations more configurations can be sampled, while in later iterations less candidate configurations should be generated to identify with a low variance a winning configuration.

When to terminate *F-Race* at each iteration? At each iteration l, *F-Race* terminates if one of the following two conditions is satisfied: (i) if the computational budget for the l-th iteration, B_l, is spent; (ii) when a minimum number of candidate configurations, denoted by N_{min}, remains. Another question concerns the value of N_{min}. *F-Race* terminates by default if a unique survivor is identified. However, to maintain sufficient exploration of the parameter space, in iterated *F-Race* it may be better to keep a number of survivors at each iteration and to sample around these survivors the candidate configurations for the next iteration. Additionally, for setting N_{min}, it may be a good idea to take into account the number of dimensions in the parameter space X: the larger the parameter space, the more survivors should remain to ensure sufficient exploration.

How should the candidate configurations be generated? As said, all candidate configurations are randomly sampled in the parameter space according to some probability distribution. For continuous and quasi-continuous parameters, continuous probability distributions are appropriate; for categorical and ordinal parameters, however, discrete probability distributions will be more useful. A first question related to

the probability distributions is of which type they should be. For example, in the first paper on iterated *F-Race* [7], normal distributions were chosen as models, but this choice need not be optimal. Another question related to the probability distributions is how they should be updated and, especially, how strong the bias towards the surviving configurations of the current iteration should be. Again, here the trade-off between exploration and exploitation needs to be taken into account.

4.4 An example iterated *F-Race* algorithm

Here we describe one example implementation of iterated *F-Race*, to which we refer as *I/F-Race* in the following. This example implementation is based on the previous one published by Balaprakash et al. [7]. However, it differs in some parameter choices and extends the earlier version by defining a way to handle categorical parameters. Note that the proposed parameter settings are chosen in an ad-hoc version; tuning the parameter settings of *I/F-Race* was beyond the scope of this chapter.

Number of iterations. We denote by L the *number of iterations* of *I/F-Race*, and increase L with d, the number of parameters, using a setting of $L = 2 + round(\log_2 d)$.

Computational budget at each iteration. The *computational budget* is distributed as equally as possible across the iterations. B_l, the computational budget in iteration l, where $l = 1, ..., L$, is set to $B_l = (B - B_{used})/(L - l + 1)$; B_{used} denotes the total computational budget used until iteration $l - 1$.

The number of candidate configurations. We introduce a parameter called candidate-evaluation trade-off factor μ_l, and set the number of candidate configurations sampled at iteration l to be $N_l = \lfloor B_l/\mu_l \rfloor$. We let μ_l increase with the number of iterations, using a setting of $\mu_l = 5 + l$. This allows more evaluation steps to identify the winners when the configurations are deemed to become more similar.

Termination of *F-Race* at each iteration. In addition to the usual termination criteria of *F-Race*, we stop it if at most $N_{min} = 2 + round(\log_2 d)$ candidate configurations remain.

Generation of candidate configurations. In the first iteration, all candidate configurations are sampled uniformly at random. Once *F-Race*

terminates, the best N_s candidate configurations are selected for the update of the probability model. We use $N_s = \min(N_{survive}, N_{min})$, where $N_{survive}$ denotes the number of candidates that survive the race. These N_s elite configurations are then weighted according to their ranks, where the weight of an elite configuration with rank r_z ($z = 1, \ldots, N_s$) is given by:

$$w_z = \frac{N_s - r_z + 1}{N_s \cdot (N_s + 1)/2}. \tag{4.1}$$

In other words, the weight of an elite configuration is inversely proportional to its rank. Since the instances for configuration are sampled randomly from the training set, the N_s elite configurations of the lth iteration will be re-evaluated in the $(l+1)$st iteration, together with the $N_{l+1} - N_s$ candidate configurations to be sampled anew. (Alternatively, it is possible to evaluate the configurations on fixed instances, so that the results of the elite configurations from the last iteration could be reused.) The $N_{l+1} - N_s$ new candidate configurations are iteratively sampled around one of the elite configurations. To do so, for sampling each new candidate configuration, first one elite solution E^z ($z \in \{1, \ldots, N_s\}$) is chosen with a probability proportional to its weight w_z and next a value is sampled for each parameter. The sampling distribution of each parameter depends on whether it is a numerical one (the set of such parameters is denoted by X^{num}) or a categorical one (the set of such parameters is denoted by X^{cat}). We have that the parameter space $X = X^{num} \cup X^{cat}$.

First suppose that X_i is a numerical parameter, i.e. $X_i \in X^{num}$, with boundary $X_i \in [\underline{X_i}, \overline{X_i}]$. Denote $v_i = \overline{X_i} - \underline{X_i}$ the range of the parameter X_i. The sampling distribution of X_i follows a normal distribution $N(x_i^z, \sigma_i^l)$, with x_i^z being the mean and σ_i^l being the standard deviation of X_i in the lth iteration. The standard deviation is reduced in a geometric fashion from iteration to iteration using a setting of

$$\sigma_i^{l+1} = v_i \cdot \left(\frac{1}{N_{l+1}}\right)^{\frac{l}{d}} \qquad \text{for } l = 1, \ldots, L-1. \tag{4.2}$$

In other words, the standard deviation for the normal distribution is reduced by a factor of $\left(\frac{1}{N_{l+1}}\right)^{\frac{1}{d}}$ as the iteration counter increments. Hence, the more parameters, the smaller the update factor becomes, resulting in a stronger bias of the elite configuration on the sampling.

Furthermore, the larger the number of candidate configurations to be sampled, the stronger the bias of the sampling distribution.

Now, suppose that $X_i \in X^{cat}$ with n_i levels $F_i = f_1, \ldots, f_{n_i}$. Then we use a discrete probability distribution $P_l(F_i)$ with iteration $l = 1, \ldots, L$, and initialize P_1 to be uniformly distributed over F_i. Suppose further that after the l-th iteration ($l > 1$), the ith parameter of the selected elite configuration E^z takes level f_i^z. Then, the discrete distribution of parameter X_i is updated as:

$$P_{l+1}(f_j) = P_l(f_j) \cdot (1 - \frac{l}{L}) + I_{j=f_i^z} \cdot \frac{l}{L} \qquad \text{for } l = 1, \ldots, L-1 \text{ and } j = 1, \ldots, n_i \tag{4.3}$$

where I is an indicator function; the bias of the elite configuration on the sampling distribution is getting stronger as the iteration counter increments.

The conditional parameters are sampled only when they are activated by their associated upper-level categorical parameter, and their sampling model is updated only when they appear in elite configurations.

4.5 Case studies: *I/F-Race*

In this section, we experimentally evaluate the presented variant of *I/F-Race* and we compare it in three case studies to *F-Race(RSD)* and *F-Race(FFD)*.

All three case studies concern the configuration of ant colony optimization (ACO) algorithms applied to the traveling salesman problem (TSP). They are ordered according to the number of parameters to be tuned. In particular, they involve configuring *MAX–MIN Ant System* ($\mathcal{MM}$AS), a particularly successful ACO algorithm [56], using four categorical parameters and configuring $\mathcal{MM}$AS using seven categorical parameters. Both case studies use the $\mathcal{MM}$AS implementation available in the ACOTSP software package.[2] The ACOTSP package implements several ACO algorithms for the TSP. The third case study uses the ACOTSP package as a black-box software and involves setting 12 mixed parameters. Among others, one of these parameters is the choice of which ACO algorithm should be used.

In all experiments we used Euclidean TSP instances with 750 nodes, where the nodes are uniformly distributed in a square of side length 10 000.

[2]The ACOTSP package is available at `http://www.aco-metaheuristic.org/aco-code/`.)

We generated 1 000 instances for training and 300 for evaluating the winning configurations using the DIMACS instance generator [34]. The experiments were carried out on cluster computing nodes, each equipped with two quad-core XEON E5410 CPUs running at 2.33 GHz with 2×6 MB second level cache and 8 GB RAM. The cluster was running under Cluster Rocks Linux version 4.2.1/CentOS 4. The programme was compiled with gcc-3.4.6-3, and only one CPU core was used for each run due to the sequential implementation of the ACOTSP software.

For each case study we have run a total of six experiments, which result by all six combinations of two different computation time limits allocated for each `function evaluation` to the ACOTSP software (five and twenty CPU seconds) and three values for the computational budget. The different levels of the computational budget have been chosen to examine the dependence of the possible advantage of *I/F-Race* as a function of the corresponding computational budget.

In each of the six experiments, 10 `trials` were run. Each `trial` is the execution of the `configuration process` (in our case, either *F-Race(FFD)*, *F-Race(RSD)*, or *I/F-Race*) together with a subsequent `testing procedure`. In the testing procedure, the final parameter setting returned by configuration process is evaluated on 300 test instances.

4.5.1 Case Study 1, $\mathcal{MM}$AS under four parameters

In this case study, we tune four parameters of $\mathcal{MM}$AS: the relative influence of pheromone trails α; the relative influence of heuristic information β; the pheromone evaporation rate ρ; and the number of ants, m.

In this first and the second case study, we discretize these numerical parameters and treat them as categorical ones. Each parameter is discretized by regular grids, resulting in a relatively large number of levels. Their ranges and number of levels as listed in Table 4.1.[3] The motivation for discretizing numerical parameters is to test whether *I/F-Race* is able to improve over *F-Race(RSD)* and *F-Race(FFD)* for categorical parameters; previously, it was already shown that *I/F-Race* gives advantages for numerical parameters [7].

The three levels of the computational budget chosen are $6 \cdot 3^4 = 486$, $6 \cdot 4^4 = 1\,536$ and $6 \cdot 5^4 = 3\,750$. In this way the candidate generation of *F-Race(FFD)* can be done by selecting the same number of levels for each parameter, in our case three, four, and five. Without *a priori* knowledge, the level of each parameter is selected randomly in *F-Race(FFD)*.

[3] For the other parameters, we use default values and we opted for an ACO version that does not use local search.

Table 4.1: Given are the parameters, the original range considered before discretization and the number of levels considered after discretization for the first case study. The number of candidate parameter settings is 12 100.

parameter	range	# levels
α	$[0.01, 5.00]$	11
β	$[0.01, 10.00]$	11
ρ	$[0.00, 1.00]$	10
m	$[5, 100]$	10

The experimental results are given in Table 4.2, and the boxplots for comparison are shown in Figure 4.5.1. The table shows the average percentage deviation of each algorithm from the reference cost, which for each instance is defined by the average cost across all candidate algorithms on that instance. The results of the algorithms tuned by *F-Race(FFD)*, *F-Race(RSD)*, and *I/F-Race*, are compared using the non-parametric pairwise Wilcoxon test with Holm adjustment, using blocking on the instances; the significance level chosen is 0.05. Results in boldface indicate that the corresponding configurations are statistically better than the ones of the two competitors.

In all experiments, *I/F-Race* and *F-Race(RSD)* significantly outperform *F-Race(FFD)*. Overall, *I/F-Race* has a slight advantage over *F-Race(RSD)*: in three of six experiments *I/F-Race* returns configurations that are significantly better than those found by *F-Race(RSD)*, while the opposite is true on only one experiment. The trend appears to be that with larger total budget, the advantage of *I/F-Race* over *F-Race(RSD)* increases. The reason for the relatively good performance of *F-Race(RSD)* could be due to the fact that the parameter space is rather small (12100 candidate configurations) and that the number of levels (10 or 11) for each parameter is large.

4.5.2 Case study 2, $\mathcal{MM}$AS under seven parameters

In this case study we have chosen seven parameters. These are the same as in the first case study plus three additional parameters: γ, a parameter that controls the gap between the minimum and maximum pheromone trail value in $\mathcal{MM}$AS, $\gamma = \tau_{max}/(\tau_{min} \cdot instance\_size)$; nn, the number of nearest neighbors used in the solution construction phase; and q_0, the probability of selecting the best neighbor deterministically in the pseudo-random proportional action choice rule; for a detailed definition see [29].

The parameters are discretized using the ranges and number of levels

Figure 4.1: Results on different *F-Race* algorithms tuning on MMASTSP-cat4. The two plots of the first row show the results of computational budget 486, and second row shows the results of budget 1536, and the last row shows the results of budget 3750. Each experiment runs 10 trials, the fine-tuned parameter of each trial will be evaluated on 300 unseen testing instances. The vertical coordinate of each box-plot shows the cost that is normalized for each testing instance instance, and the reference cost for the normalization is taken by the mean cost across all listed algorithms.

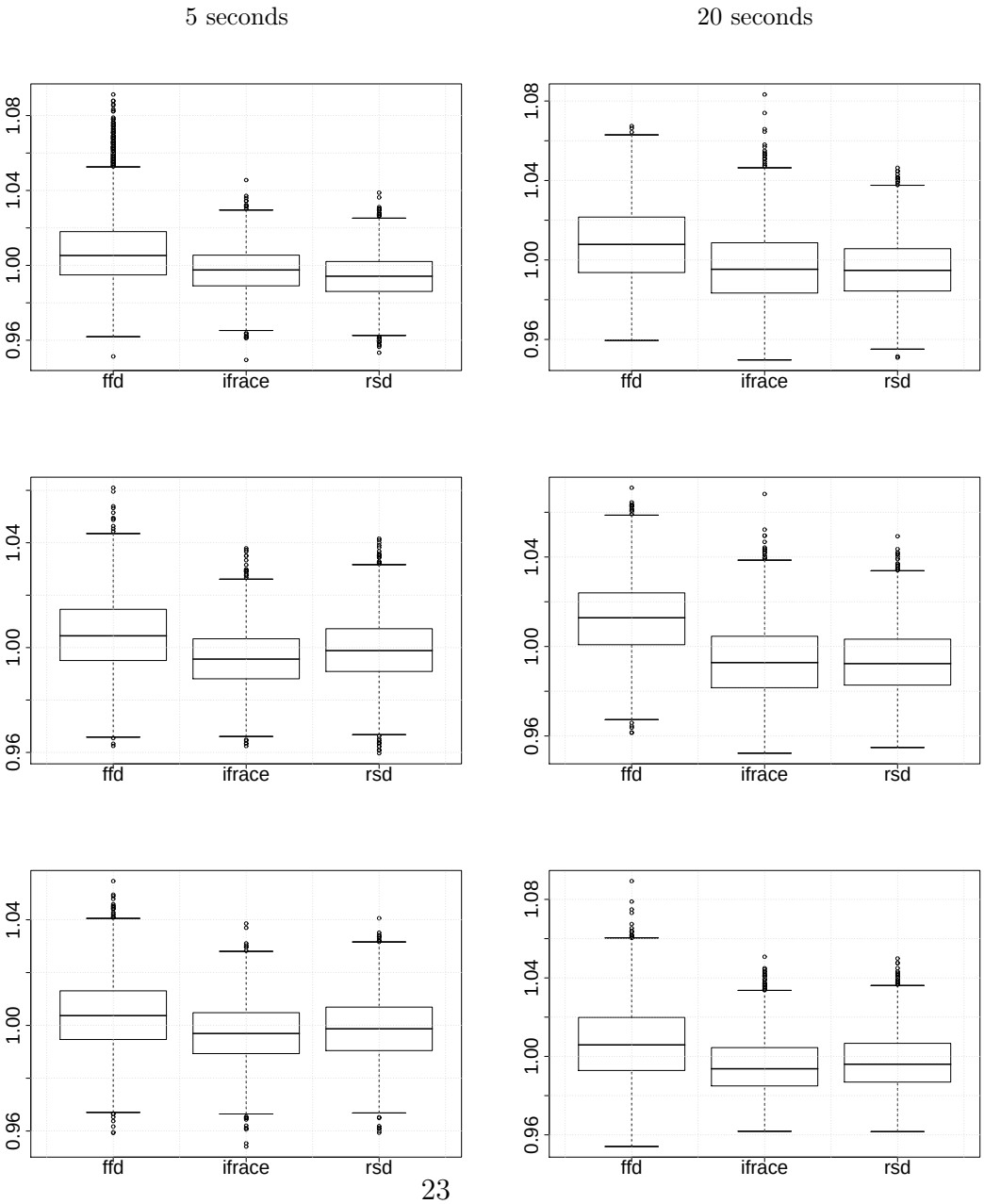


Table 4.2: Computational results for configuring $\mathcal{MM}$AS for the TSP with 4 discretized parameters for a computation time bound of 5 and 20 seconds, respectively. The column entries with the label `per.dev` shows the mean percentage deviation of each algorithm from the reference cost. $+x$ ($-x$) means that the solution cost of the algorithm is x% more (less) than the reference cost. The column with the label `max.bud` gives the maximum number of evaluations given to each algorithm.

	5 seconds	20 seconds	
algo	per.dev	per.dev	max.bud
F-Race(FFD)	+0.85	+0.79	486
F-Race(RSD)	**−0.58**	−0.44	486
I/F-Race	−0.26	−0.34	486
F-Race(FFD)	+0.51	+1.27	1 536
F-Race(RSD)	−0.08	−0.66	1 536
I/F-Race	**−0.42**	−0.61	1 536
F-Race(FFD)	+0.40	+0.71	3 750
F-Race(RSD)	−0.12	−0.27	3 750
I/F-Race	**−0.28**	**−0.45**	3 750

given in Table 4.3. Note that in comparison to the previous experiment, the parameter space is more than one order of magnitude larger (259 200 $\gg$ 12 100). Besides, there is smaller number of levels for each parameter, usually between four to nine. We use the same experimental setup as in the previous section, except that for the computational budget, we choose $6 \cdot 2^7 = 768$ such that each parameter in *F-Race(FFD)* has two levels, $6 \cdot 2^5 \cdot 3^2 = 1\,728$, such that in *F-Race(FFD)*, five parameters will have two levels and the other two three levels, and $6 \cdot 2^3 \cdot 3^4 = 3\,888$, such that in *F-Race(FFD)*, three parameters will have two levels, and the other four parameters have three levels.

The experimental results are listed in Table 4.4 and the comparison boxplots are shown in Figure 4.5.2 and the results are analyzed in a way analogous to case study 1. The results clearly show that *I/F-Race* significantly outperforms *F-Race(FFD)* and *F-Race(RSD)* in each experiment. As expected, also *F-Race(RSD)* outperforms *F-Race(FFD)* significantly.

Figure 4.2: Results on different *F-Race* algorithms tuning on MMASTSP-cat7. The two plots of the first row show the results of computational budget 768, and second row shows the results of budget . Each experiment runs 10 trials, the fine-tuned parameter of each trial will be evaluated on 300 unseen testing instances. The vertical coordinate of each box-plot shows the cost that is normalized for each testing instance instance, and the reference cost for the normalization is taken by the mean cost across all listed algorithms.

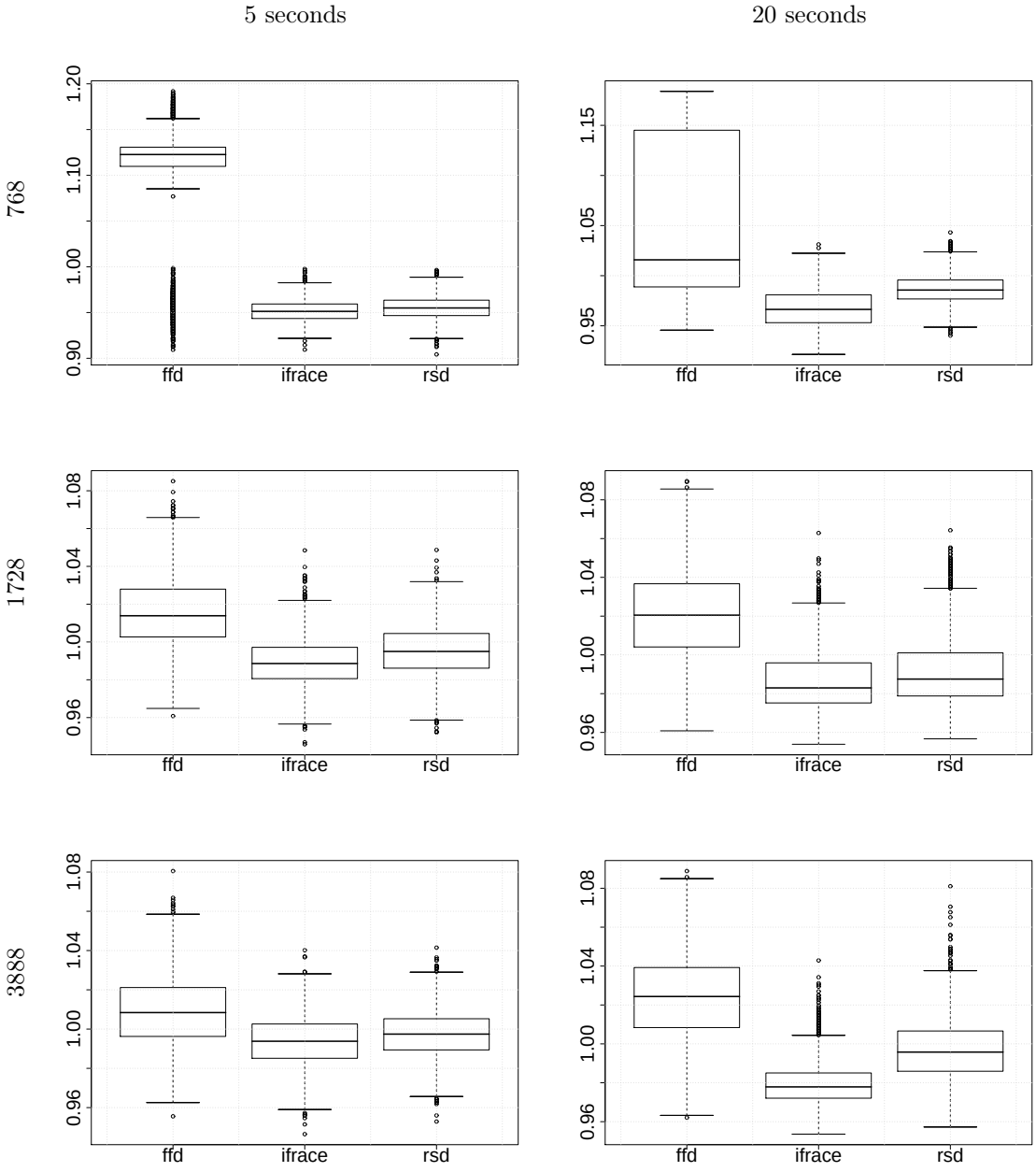


Table 4.3: Given are the parameters, the original range considered before discretization and the number of levels considered after discretization for the first case study. The number of candidate parameter settings is 259 200.

parameter	range	#levels
α	$[0.01, 5.00]$	5
β	$[0.01, 10.00]$	6
ρ	$[0.00, 1.00]$	8
γ	$[0.01, 5.00]$	6
m	$[5, 100]$	5
nn	$[5, 50]$	4
q_0	$[0.0, 1.0]$	9

4.5.3 Case study 3, ACOTSP under twelve parameters

In a final experiment, 12 parameters of the ACOTSP software are examined. This configuration task is the most complex and it requires the setting of categorical as well as numerical parameters.

Among these parameters, firstly two categorical parameters have to be determined, (i) the choice of the ACO algorithm, among the five variants $\mathcal{MM}$AS, ant colony system (ACS), rank-based ant system (RAS), elitist ant system (EAS), ant system (AS); (ii) the local search type l, including four levels: no local search, 2-opt, 2.5-opt, and 3-opt. All the ACO construction algorithms share the three continuous parameter α, β, and ρ, and two quasi-continuous parameters m and nn, which have been introduced before. Moreover, five conditional parameters are considered: (i) the continuous parameter q_0 (introduced in Sec. 4.5.2) is only in effect when ACS is deployed; (ii) the quasi-continuous $rasrank$, is only in effect when RAS is chosen; (iii) the quasi-continuous $eants$ is only in effect when EAS is applied; (iv) the quasi-continuous parameter $nnls$ is only in effect when local search is used, namely either 2-opt, 2.5-opt or 3-opt; (v) the categorical parameter dlb is only in effect when local search is used. Here, only *F-Race(RSD)* and *I/F-Race* are tested because *F-Race(FFD)* has so far already been outperformed by the other two variants, and, due to the large number of parameters, running *F-Race(FFD)* becomes infeasible. As computational budgets we adopted 1 500, 3 000 and 6 000 function evaluations. The experimental results are given in Table 4.5, and the comparison boxplots are shown in Figure 4.5.3. The two algorithms *F-Race(RSD)* and *I/F-Race* are compared using the non-parametric pairwise Wilcoxon test using a 0.05 significance level. The statistical com-

Table 4.4: Computational results for configuring $\mathcal{MM}$AS for TSP with seven categorical parameters in 5 and 20 CPU seconds. For an explanation of the table entries see the caption of Table 4.2.

	5 seconds	20 seconds	
algo	per.dev	per.dev	max.bud
F-Race(FFD)	+9.33	+4.61	768
F-Race(RSD)	−4.49	−1.35	768
I/F-Race	**−4.84**	**−3.25**	768
F-Race(FFD)	+1.58	+2.11	1 728
F-Race(RSD)	−0.49	−0.78	1 728
I/F-Race	**−1.10**	**−1.33**	1 728
F-Race(FFD)	+0.90	+2.38	3 888
F-Race(RSD)	−0.27	−0.33	3 888
I/F-Race	**−0.63**	**−2.05**	3 888

parisons show that *I/F-Race* is again dominating. It is significantly better performing in five out of six experiments; only in one case no statistically significant difference can be identified. However, the quality differences in this set of experiments are quite small, usually below 0.1% in the five CPU seconds case, while in the 20 CPU seconds case the difference is below 0.01%. This shows that the solution quality is not very sensitive to the parameter settings. This is usually the case when a strong local search such as 3-opt is used.

4.6 Combine Mesh Adaptive Direct Search (MADS) and *F-Race*

The MADS class of algorithms is designed for global optimization problems which can be in general stated as

$$\min_{p \in \Omega} f(p) \tag{4.4}$$

for a function $f : \Omega \in \mathbb{R}^d \to \mathbb{R} \cup +\infty$, and $d = |\Omega|$ denotes the dimension of the variable space. The algorithm does not require derivative information or continuity of f. In fact, the evaluation function f can be treated as a black-box, which makes it convenient for the algorithm configuration problem. MADS class of algorithms are reported to be robust for nonsmooth

Figure 4.3: Results on different *F-Race* algorithms tuning on ACOTSP-full. The two plots of the first row show the results of computational budget 1500, and second row shows the results of budget 3000, and the last row shows the results of budget 6000. Each experiment runs 10 trials, the fine-tuned parameter of each trial will be evaluated on 300 unseen testing instances. The vertical coordinate of each box-plot shows the cost that is normalized for each testing instance instance, and the reference cost for the normalization is taken by the mean cost across all listed algorithms.

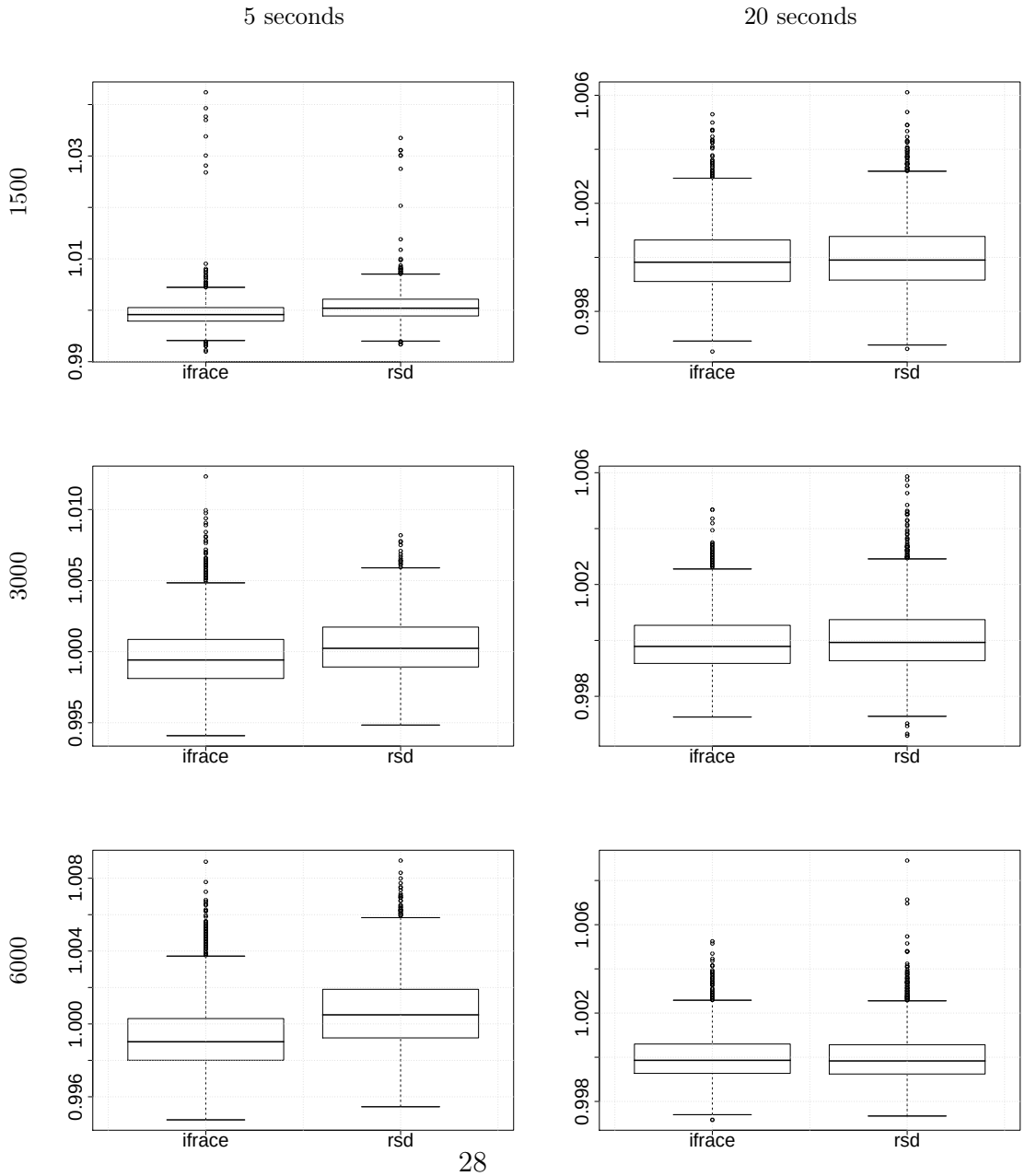


Table 4.5: Computational results for configuring $\mathcal{MM}$AS for TSP with 12 parameters in 5 and 20 CPU seconds. For an explanation of the table entries see the caption of Table 4.2.

	5 seconds	20 seconds	
algo	per.dev	per.dev	max.bud
F-Race(RSD)	+0.06	+0.005	1 500
I/F-Race	**−0.06**	**−0.005**	1 500
F-Race(RSD)	+0.04	+0.009	3 000
I/F-Race	**−0.04**	**−0.009**	3 000
F-Race(RSD)	+0.07	−0.001	6 000
I/F-Race	**−0.07**	+0.001	6 000

optimization problems [3], and its convergence properties given only bound constraints is shown in [4], and given more general constraints is shown in [3]. It is an extention to the generalized pattern search [2] by allowing more general exploration of the variable space.

4.6.1 A general overview of MADS

As described in algorithm 4.2, MADS is an iterative algorithm, where at each iteration a population of trial points are generated. All these trial points must lie on the *current mesh*, whose coarseness is controlled by a *mesh size parameter* $\Delta_k \in \mathbb{R}_+$. The set of mesh points at iteration k is defined as:

$$M_k = \bigcup_{p \in S_k} \{p + \Delta_k z : z \in \mathbb{Z}^d\} \tag{4.5}$$

where S_k denotes the set of points that have been evaluated in the previous iterations. Each iteration of MADS essentially consists of two steps, the search step and the poll step. In the search step, a number of trial points are randomly sampled on the current mesh. In the poll step, a number of trial points are generated around the incumbent point. These trial points of poll forms the set called *frame*, which is defined as:

$$F_k = \{p_k \pm \Delta_k b : b \in B_k\} \tag{4.6}$$

where $B_k = \{b^1, b^2, \ldots, b^d\}$ is a basis in $\mathbb{Z}^d$. At iteration k, the mesh size parameter is updated as follows:

$$\Delta_{k+1} = \begin{cases} \Delta_k/4 & \text{if no improved mesh point is found;} \\ 4\Delta_k & \text{if an improved mesh point is found, and } \Delta_k \leq \frac{1}{4}; \\ \Delta_k & \text{otherwise.} \end{cases} \quad (4.7)$$

Algorithm 4.2 The mesh adaptive direct search

Step 0: Initialization Given p_0 to be a feasible initial point, $\Delta_0 > 0$, set $k = 0$ and go to Step 1.

Step 1: `search` **step** Let $L_k \subset M_k$ be the finite set of mesh points, and let $q^* = \arg\min_{q \in L_k} f(q)$ be the iteration best point from `search`. If $f(q^*) < f(p_k)$, declare k successful and set $p_{k+1} = q^*$ and go to Step 3; otherwise go to Step 2.

Step 2: `poll` **step** Generate the frame F_k as in eq. 4.6, and let $q^* = \arg\min_{q \in F_k} f(q)$ be the iteration best point from `poll`. If $f(q^*) < f(p_k)$, declare k successful and set $p_{k+1} = q^*$; otherwise declare k unsuccessful. Go to Step 3.

Step 3: Parameter update If iteration k is unsuccessful, set $p_{k+1} = p_k$, otherwise set p_{k+1} as the improved mesh point. Update mesh size parameter Δ_k according to eq. 4.7, increment $k \leftarrow k + 1$ and go to Step 1.

4.6.2 The hybrid of *F-Race* and `MADS`

In order to handle the noisy black-box objective function, `MADS` should be adapted by allowing more than one evaluations on the trial points so as to reduce the evaluation variance. A brute-force approach is to perform a fixed number of evaluations. However, it is not a priori known what is the appropriate evaluation step number, besides, the same amount of computational resources have to be allocated to the good performing candidates as well as the bad performing ones. To this end, a hybrid of *F-Race* and `MADS` (or `MADS/F-Race`), is proposed to cope with this difficulty. The hybrid can be done in a rather straightforward way. At each iteration of `MADS`, be it the `search` or `poll` step, a population of candidate configurations L_k sampled from `MADS` will be evaluated by *F-Race*, together with the incumbent point p_k. The objective of *F-Race* in this task is to identify the best point out of $L_k \cup \{p_k\}$. If p_k is identified by *F-Race* the best, then declare the iteration unsuccessful; if any candidate in L_k is the winner, the iteration is successful, and the `MADS` parameters are updated accordingly.

Table 4.6: Parameters used in `MADS`. d denotes the dimension of the parameter space.

parameter	value
Initial search type	latin hypercube search
#Points in initial search	d^2
Iterative search type	random search
#Points in iterative search	$2d$
#Points in `poll` step	$2d$
Speculative search	yes
Initial mesh size parameter	1.0

4.6.3 Implementation details of `MADS`

The specific parameter values chosen in the `MADS` implementation are listed in Table 4.6. For the setting of the `MADS` algorithm, we follow [4].

For the parameter settings of *F-Race* in `MADS/F-Race`, we follow the example of *I/F-Race* in Section 4.4, except that the candidate-evaluation trade-off factor $\mu = 10$ to be a constant for each iterations. Let N_l be the number of trial points generated from `MADS` at the l-th iteration, the budget of the iteration l is set to $B_l = N_l \cdot \mu$

4.7 Case Study: `MADS/F-Race` and `MADS(fixed)`

The experiments are conducted on six problem classes:

- MMASTSP, the $\mathcal{MAX}$–$\mathcal{MIN}$ *Ant System* ($\mathcal{MM}$AS) for traveling salesman problem (TSP) with 6 parameters;
- MMASTSP_ls, the $\mathcal{MM}$ASwith 2-opt local search for TSP, the same parameters as MMASTSP;
- ACSTSP, ant colony system (ACS) for TSP with 6 parameters;
- ACSTSP_ls, ACS with 2-opt local search fo TSP, the same parameters as ACSTSP, and the names and ranges of the parameters of these four ACOTSP problem classes are listed in Table 4.7;
- ILSQAP, iterated local search (ILS) for qradratic assignment problem (QAP) with 8 parameters (the names and ranges of the parameters of ILSQAP are listed in Table 4.8);

Table 4.7: Range of each parameter considered for tuning MMASTSP (including MMASTSP_ls) and ACSTSP (including ACSTSP_ls).

MMASTSP		ACSTSP	
parameter	range	parameter	range
α	$[0.0, 5.0]$	α	$[0.0, 5.0]$
β	$[0.0, 10.0]$	β	$[0.0, 10.0]$
ρ	$[0.0, 1.00]$	ρ	$[0.0, 1.00]$
γ	$[0.01, 5.00]$	q_0	$[0.0, 1.0]$
m	$[1, 1200]$	m	$[1, 1200]$
nn	$[5, 100]$	nn	$[5, 100]$

Table 4.8: Range of each parameter considered for tuning ILSQAP.

parameter	range	parameter	range	parameter	range	parameter	range
tc	$[0, 5]$	ia	$[3, 99]$	iu	$[0, 2]$	iy	$[0, 1]$
ti	$[0, 20]$	it	$[0, 10]$	ix	$[1, 100]$	iz	$[0, 1]$

- SAQAP, simulated annealing (SA) for QAP with 3 parameters (the names and ranges of the parameters of SAQAP are listed in Table 4.9).

Each class contains 4 different computation time (1, 2, 5, 10 seconds) and 3 different numbers of function evaluation budgets (1000, 2000 and 4000), which makes it in total 12 *domains* per problem class.

4.7.1 Compare `MADS/F-Race` to `MADS(fixed)` with random evaluation number

In order to apply `MADS(fixed)`, six levels of the evaluation number are preassigned: 1, 5, 10, 20, 30, 40. Given no a priori information what the optimal

Table 4.9: Range of each parameter considered for tuning SAQAP.

parameter	range	parameter	range	parameter	range
sb	$[0.0, 10.0]$	sc	$[0.0, 1.0]$	sd	$[1, 100000]$

setting of `MADS(fixed)` is, we select one setting from the six randomly for each problem domain, and compare it with `MADS/F-Race` across all domains, with blocking on each instance. The comparison is done using the Wilcoxon signed rank test with continuity correction; the α-level chosen is 0.05. The experimental results show that `MADS/F-Race` statistically significantly performs better than `MADS(fixed)` with randomly selected number of evaluations on each individual domain. The average improvement of `MADS/F-Race` over `MADS(fixed)` is around 0.25%. The QQplot is shown in Figure 4.4.

4.7.2 The leave-one-out cross-validation

In the sequel, we fine-tune the parameter of `MADS(fixed)`, the number of evaluations, through a so-called leave-one-out cross-validation.

Cross-validation is a technique that is commonly used in the field of machine learning and statistics, to assess how the results of statistical analysis are generalized to an independent data set. To do so, on each cross-validation iteration, the available data set is usually partitioned into two sets, the training set based on which the quality of the candidate settings are observed and analyzed, and the testing set on which the previously assessed results are tested. Then the process is so iterated, using different partitions in each iteration, that the variability of the validation is reduced.

There are various ways how the cross-validation can be performed. Specifically in our case, the common leave-one-out cross-validation is applied. As the name suggest, in each iteration of the leave-one-out cross-validation, one domain is picked for testing, and the rest of the data set serve as the training set. The best candidate chosen according to the training set will be performed on the testing domain, and its results are collected into a validation set. The process repeats until each domain has collected its validation data. More formal description of how the leave-one-out cross-validation is applied in our case is given in Algorithm Box 4.3.

The goal of this leave-one-out cross-validation in our experiment is to compare the validation set of `MADS(fixed)` to the `MADS/F-Race`. Note that the data collected in the validation set are first trained, while `MADS/F-Race` is not. The unfairness of the comparison will show the robustness and advantage of `MADS/F-Race` over the naive `MADS(fixed)`.

4.7.3 Compare `MADS/F-Race` to `MADS(fixed)` with tuned evaluation number

The comparison results show MADS/F-race significantly (by wilcoxon test) outperforms MADS-fixed with tuned evaluation numbers, and the difference

Algorithm 4.3 The leave-one-out cross-validation

Given a set of domains D, a set of candidate algorithms A^c, a target algorithm a^t, and the set of cost function $c(a, d)$ referring to the expected cost of algorithm $a \in A^c \cup \{a^t\}$ on domain $d \in D$, $C(a, D)$ referring to the expected cost of algorithm a on a domain set D.
Set $V^t \leftarrow \bigcup_{d \in D} c_d^{a^t}$ stores the results of target algorithm.
Set $V^c \leftarrow \emptyset$ stores the results of tuned candidate algorithm
for $d \in D$ **do**
  Select $\bar{a} = \arg\min_{a \in A^c} C(a, D \setminus \{d\})$
  Set $V^c \leftarrow V^c \cup \{c(\bar{a}, d)\}$
end for
Compare the tuned candidate vector V^c and the target vector V^t

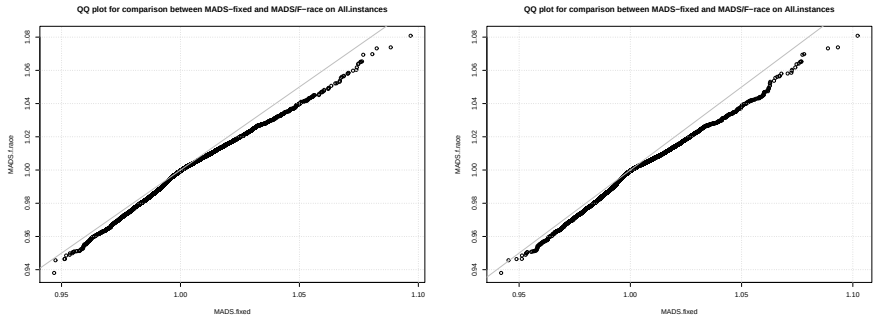


Figure 4.4: The comparison of normalized costs between random (left) or tuned (right) MADS-fixed and MADS/F-race across all domains.

of the mean performance between `MADS(fixed)` and `MADS/F-Race` is around 0.17%. The QQplot is shown in Figure 4.4.

However, in each individual problem class, the comparison results of `MADS/F-Race` and tuned `MADS(fixed)` are less unanimous. As shown in Figure 4.5, `MADS/F-Race` obtains significantly better results than tuned `MADS(fixed)` in MMASTSP, ACSTSP and SAQAP, while in the rest of the problem classes MMASTSP_ls ACSTSP_ls and ILSQAP, `MADS/F-Race` has significantly worse results than tuned `MADS(fixed)`. Table 4.10 also shows the percentage deviation between `MADS/F-Race` and tuned `MADS(fixed)`.

4.7.4 Variation Coefficient

The fact that the advantage of `MADS/F-Race` is more clear in some problem classes than the others urges us to find out what factors of the problem classes affects the performance of the tuning algorithms. One of the factors we have investigated is the *variation coefficient*. The variation coefficient

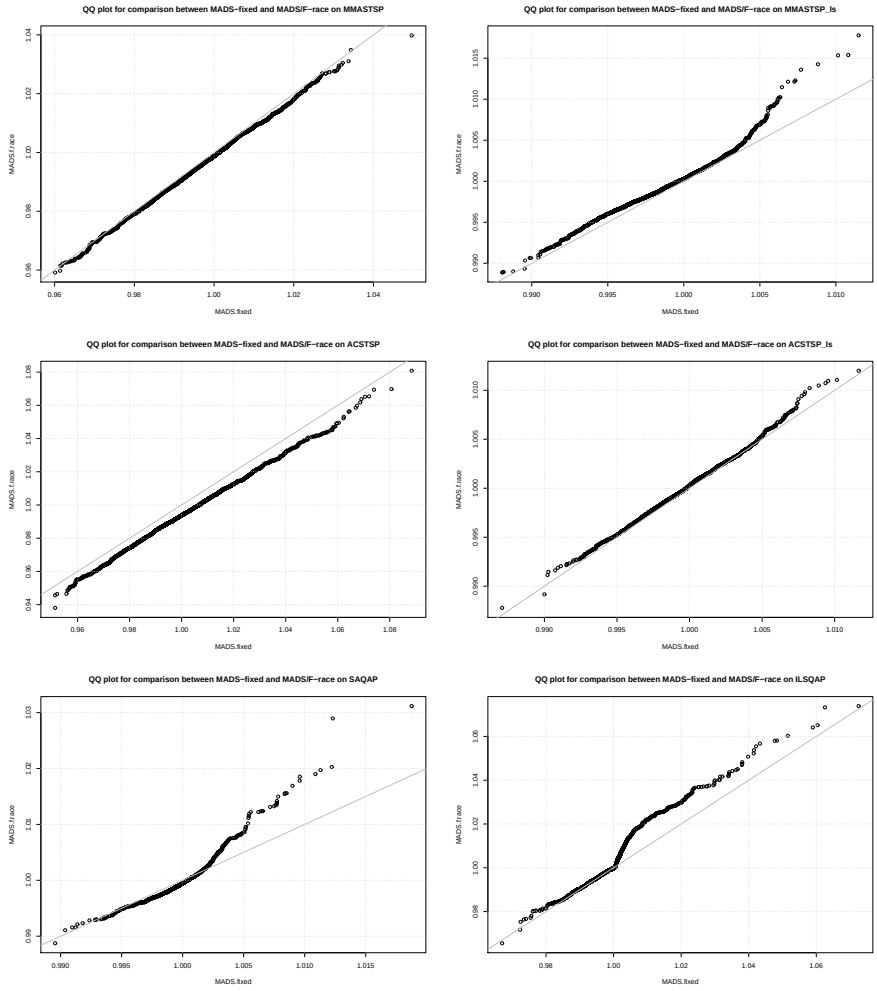


Figure 4.5: The comparison of normalized costs for tuned MADS-fixed and MADS/F-race on MMASTSP.

measures the variance of the search landscape. The variation coefficient of each problem class is obtained as follows, firstly 25 randomly generated parameter settings are applied on 25 generated instances, then for each of the instances we compute the ratio of the standard deviation over the mean cost of the 25 parameter settings. In such a way the variation coefficient is scale invariant. High value of the variation coefficient infers steep slopes in the search landscape, while low value of variation coefficient usually infers a flat landscape, which is usually the case for the robust algorithms when a strong local search is adopted. The box-plot of the variation coefficient of the six problem classes is shown in Figure 4.6.

The problem classes MMASTSP, ACSTSP and SAQAP, on which `MADS/F-Race`

Table 4.10: Comparison results of MADS/F-Race and MADS(fixed) with tuned evaluation numbers for tuning six different problem classes. The column entries with the label per.dev shows the mean percentage deviation of cost obtained by MADS/F-Race comparing with tuned MADS(fixed). $+x$ ($-x$) means that the solution cost of MADS/F-Race is $x\%$ more (less) than tuned MADS(fixed), i.e. MADS/F-Race performs $x\%$ worse (better) than tuned MADS(fixed).

problem	per.dev	problem	per.dev
MMASTSP	-0.12	MMASTSP_ls	$+0.055$
ACSTSP	-0.64	ACSTSP_ls	$+0.027$
SAQAP	-0.030	ILSQAP	$+0.13$

obtains significant better results than tuned MADS(fixed), have relatively higher variation coefficient values than the other three problem classes. The positive correlation between the problem variation and the performance of MADS/F-Race is an interesting topic to investigate in the future.

4.8 BOBYQA for algorithm configuration problem

In the sequel, we study another algorithm for derivative-free continuous optimization and apply it to the algorithm configuration problem.

4.8.1 BOBYQA

BOBYQA [48], acronym for Bound Optimization BY Quadratic Approximation, is a mathematical software for solving continuous optimization problems. It is an iterative algorithm for finding a minimum of a function $f(x), x \in \mathbb{R}^d$, subject to bound constraints $\underline{x} \leq x \leq \bar{x}$ on the variables. Similar to MADS algorithm, BOBYQA requires no derivative information of f, thus f can be specified by any black-box evaluation function. At each iteration, BOBYQA employs a quadratic model to interpolate m automatically chosen points that have been evaluated in the previous iterations. The value m is a constant and is typically suggested [48] to take the value as $m = 2d + 1$, where d denotes the dimension of the search space. One trial point will be generated by the guess from the quadratic approximation and under the framework of a trust-region procedure. BOBYQA is an extention to the soft-

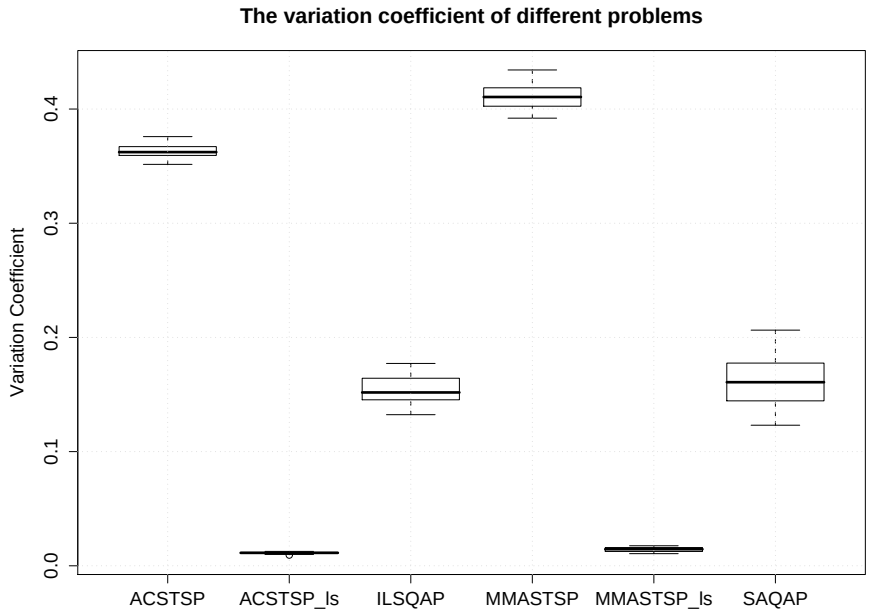


Figure 4.6: The variance coeffient of each problem used.

ware NEWUOA [47] by imposing an addtional bound constraints. NEWUOA is considered to be a state-of-the-art continuous optimization technique [41]

One aspect of the algorithm configuration problem that makes the adoption of BOBYQA difficult is its stochastic nature. To the best of our knowledge, BOBYQA has not been applied to optimization of noisy functions. To cope with this difficulty, we have to increase the number of evaluation steps on each trial points in order to reduce the evaluation variance. Let n be the number of evaluation steps of each trial points within each BOBYQA execution.

Unlike `MADS`, where in each iteration only the winner of a set of trial points will influence the search behavior, in BOBYQA each trial point will have the possibility to be chosen to build the quadratic predictive model and thus influence the search behavior. Besides, each trial point has to be evaluated with the same number of fixed instances, so there is no direct way to hybridize *F-Race* with BOBYQA.

4.9 Case studies: BOBYQA and iterated *F-Race*

The problem classes used in this case study are:

Table 4.11: Parameters used in BOBYQA. d denotes the dimension of the parameter space.

parameter	value
Initial trust region radius	0.5
Final trust region radius	10^{-15}
# interpolation points	$m = 2d + 1$

- MMASTSP-2, *MAX–MIN Ant System* for TSP with 2 parameters, namely β with range $[0.0, 10.0]$, and ρ with range $[0.0, 1.0]$, the rest of the parameters take the default values;
- MMASTSP-4, *MAX–MIN Ant System* for TSP with 4 parameters, besides aforementioned β and ρ, also α with range $[0.0, 5.0]$ and m with range $[1, 1200]$;
- MMASTSP-6, the *MAX–MIN Ant System* for TSP with 6 parameters as listed in Table 4.7;

The detailed settings of BOBYQA are listed in Table 4.11

For the number of evaluation steps for BOBYQA, we have tested four different levels: 1, 5, 10 and 20. As the tuning budget, three levels are chosen: 1000, 2000 and 4000. As comparison to BOBYQA, we choose 3 different tuning algorithms: *F-Race(RSD)*, *I/F-Race*, and `MADS/F-Race`. The advantage of `MADS/F-Race` over `MADS(fixed)` has been shown in Section 4.7, therefore we remove them from the comparison.

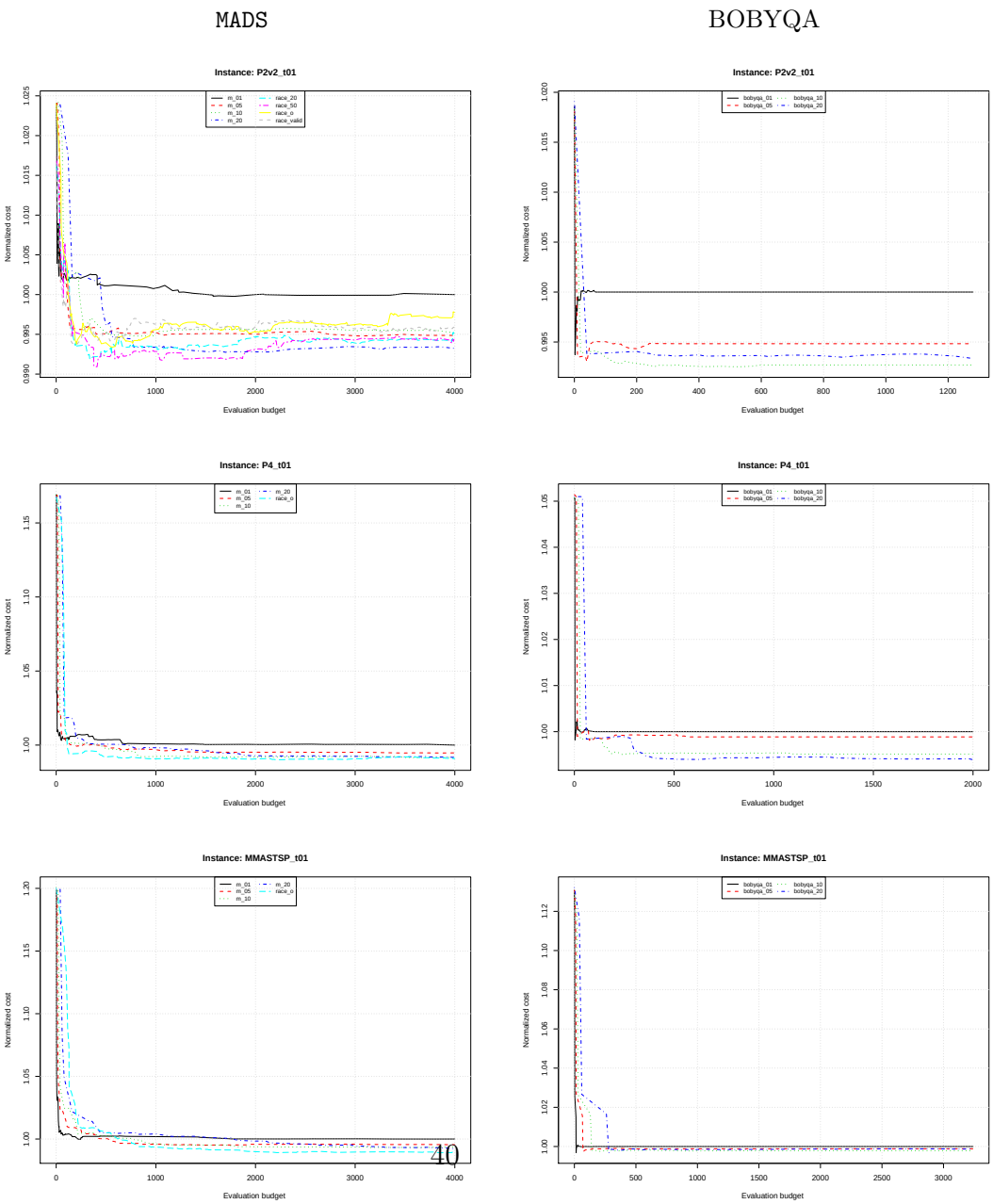
As shown in Table 4.12, `MADS/F-Race` and *I/F-Race* are usually the best-performing tuning algorithms for the problem classes chosen. BOBYQA algorithms usually perform poorly, in most of the cases it performs worse than *F-Race(RSD)*. Within the BOBYQA class of algorithms, it is observed that the performance of the algorithm grows with increased evaluation numbers. This is not only true for BOBYQA, it holds also for `MADS` class of algorithms. This is more clearly shown in the runtime development plots in Figure 4.9.

It can be shown from Figure 4.9 that, firstly the both classes of algorithms, especially the BOBYQA class of algorithms, reach a very good-quality solution with very few evaluation budget and then usually stagnate, which motivates us to restart the search everytime when the solution stagnates. This will be the topic of section 4.10. The second feature we observe is that, in some of the plots, the best found solution is lost during the tuning process, especially when the evaluation number is low (e.g. 1 or 5). Increasing the

Table 4.12: The comparison results of four BOBYQA algorithms (b-01, b-05, b-10 and b-20) with *I/F-Race*, `MADS/F-Race`, *F-Race(RSD)* for tuning three different problem classes with three levels of tuning budget each class. The entries show the mean percentage deviations of each algorithm from the reference cost. $+x$ ($-x$) means that the solution cost of the algorithm is $x\%$ more (less) than the reference cost. The statistically (by pairwise Wilcoxon test with α level 0.05) significant winner(s) in each problem domain are marked with bold face.

domains	*I/F-Race*	`MADS/F-Race`	*F-Race(RSD)*	b-01	b-05	b-10	b-20
MMASTSP-2							
1000	0.0457	**−0.6531**	−0.0136	0.3370	0.1029	−0.3877	−0.4197
2000	−0.0788	−0.2927	−0.0108	0.1238	−0.1620	−0.1875	**−0.3951**
4000	−0.1422	**−0.5716**	−0.0151	0.3833	−0.1598	−0.3381	−0.3026
MMASTSP-4							
1000	**−0.2068**	**−0.2081**	−0.0072	0.9422	0.6167	0.2859	0.1213
2000	**−0.3914**	−0.2964	−0.0291	1.0339	0.6590	0.2350	0.2088
4000	−0.3599	**−0.4789**	0.0015	0.7977	0.6677	0.3910	0.2787
MMASTSP-6							
1000	−0.7522	**−0.7935**	−0.0471	0.1654	−0.1099	−0.3526	−0.3259
2000	**−0.8094**	**−0.8827**	−0.0129	0.0317	−0.0729	0.1642	0.0801
4000	**−0.8389**	−0.8174	−0.0035	0.3743	0.2649	0.1596	0.2220

Figure 4.7: The runtime development plot of the `MADS` class of algorithms and BOBYQA class of algorithms on three problem classes with tuning budget 4000. The two plots of the first row show the results of problem class MMASTSP-2, and second row shows the results of MMASTSP-4, and the last row shows the results of MMASTSP-6. Each improving point in the tuning phase is evaluated on 300 unseen testing instances. The vertical coordinate of each box-plot shows the cost that is normalized for each testing instance instance, and the reference cost for the normalization is taken by the mean cost across all listed algorithms.



evaluation number gradually solves the problem. But note that the trade-off exists, such that high evaluation number will limit the exploration of the search space. Therefore a more aggressive incumbent point defense should be considered, such as in [32].

4.10 Multiple restart of BOBYQA and MADS classes of algorithms

Since BOBYQA and MADS classes of algorithms are developed based on the trust-region methods, which is by definition a local optimizer, the search may converge to some local optimum in the case that the objective function is multimodal. For this reason we applied BOBYQA and MADS with multiple restarts, in order to reduce the probability of falling into a low-quality local optima. In our setting, when the trust region radius reaches a certain threshold, the program is restarted from a new randomly generated initial solution.

The best solution of each BOBYQA or MADS execution is stored into an archive set, and *F-Race* is applied at the end to select the best parameter setting out of the archive set in the phase that we call the post selection phase. We name the multiple restart BOBYQA with the help of *F-Race* in the post selection phase to be BOBYQA/F-Race .

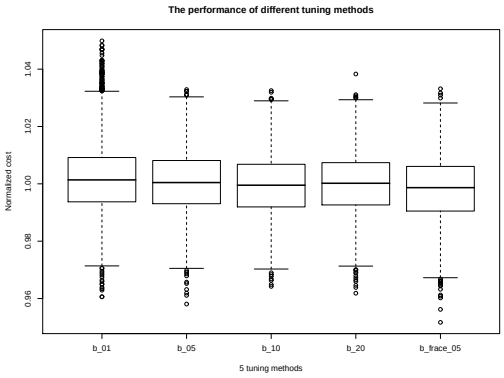
For both BOBYQA and MADS, we set the final trust region radius to be 10^{-3}, and the initial trust region radius for BOBYQA is set to 0.1. For MADS we have chosen to extend the MADS/F-Race with multiple restart. And for BOBYQA, we first set the evaluation number to be $n = 5$. In the post selection phase, *F-Race* is applied with the following settings: i) the first test after the 10th evaluation step; ii) the candidate-evaluation trade-off factor μ is set to be a constant 20; iii) The previous evaluations are stored and need not to be re-evaluated.

The experiment is performed on the MMASTSP problem class with 6 parameters, and the results in Figure 4.10 show a substantial and statistically significant advantage of the multiple restart version over the original algorithms. Note that especially for MADS/F-Race, the original MADS/F-Race starts its search from a relatively good initial solution, while the multiple restart MADS/F-Race is not provided with a priori initial solutions, which also shows the robustness of the multiple restart mechanism.

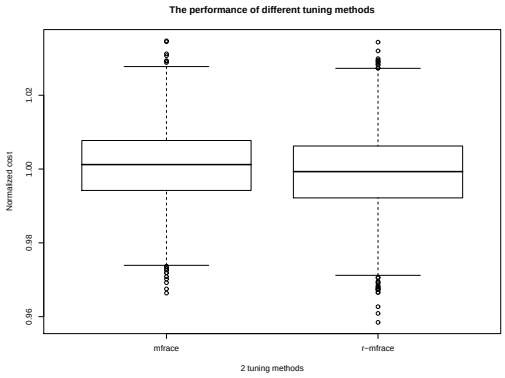
However, More experiments on various problem classes are still needed to confirm the potential of the multiple restart BOBYQA and MADS/F-Race.

Figure 4.8: The boxplots to compare the multiple restart version of BOBYQA and `MADS/F-Race` with their original version on tuning MMASTSP. The left plot shows the comparison of multiple restart BOBYQA with the BOBYQA with fixed evaluation number. The right plot shows the comparison of multiple restart `MADS/F-Race` with the original `MADS/F-Race`. Each experiment runs 10 trials, the fine-tuned parameter of each trial will be evaluated on 300 unseen testing instances. The vertical coordinate of each box-plot shows the cost that is normalized for each testing instance instance, and the reference cost for the normalization is taken by the mean cost across all listed algorithms.

BOBYQA



`MADS`



Chapter 5

A review of *F-Race* applications

F-Race has received significant attention as it is witnessed by the 99 citations to the first article on *F-Race* [17] in the google scholar database as of June 2009. In what follows, we give an overview of researches that applied *F-Race* in various contexts.

Fine-tuning algorithms. The by far most common use of *F-Race* is to use it as a method to fine-tune an existing or a recently developed algorithm. Often, the tuning through *F-Race* is also done before comparing the performance of various algorithms. In fact, this latter usage is important to make reasonably sure that performance differences between algorithms are not simply due to an uneven tuning.

A significant fraction of the usages of *F-Race* is due to researchers either involved in the development of the *F-Race* method or by their collaborators. In fact, *F-Race* has been developed in the research for the Metaheuristics Network, an EU-funded research and training network on the study of metaheuristics. Various applications there have been for configuring different metaheuristics for the university-course timetabling problem [23, 36, 50] and also for various other problems [21, 24, 27, 49, 51].

Soon after these initial applications, *F-Race* has also been adopted by a number of other researchers. Most applications focus on configuring SLS methods for combinatorial optimization problems [12, 5, 28, 30, 35, 45, 46]. However, also other applications have been considered including the tuning of algorithms for training neural networks [18, 55], or the tuning of parameters of a control system for simple robots [42, 43].

Industrial applications. Few researches have evaluated *F-Race* in pilot studies for industrial applications. The first has been a feasibility study, where *F-Race* was used to configure a commercial solver for vehicle routing and scheduling problems, which has been developed by the software com-

pany SAP. In this research, six configuration tasks have been considered that ranged from the study of specific parameters, which determined the frequency of the application of some important operators of the program, to the configuration of the SLS method that was used in the software package. *F-Race* was compared to a strategy that after each fixed number of instances discarded a fixed percentage of the worst candidate configurations, showing, as expected, advantages for *F-Race* when the performance differences between configurations were stronger. Some results of this study have been published in [10]; more details are available in a master thesis [9].

Yuan et al. [60] have adopted *F-Race* to configure several algorithms for a highly constrained train scheduling problem arising at Deutsche Bahn AG. A comparison of various tuned algorithms identified an iterated greedy algorithm as the most promising one.

Algorithm development. *F-Race* has occasionally also been used to explicitly support the algorithm development process. A first example is described by Chiarandini et al. [22] who used *F-Race* to design a hybrid metaheuristic for the university-course timetabling problem. In their work they have adopted *F-Race* in a semi-automatic way. They observed the algorithm candidates that were maintained in a race and based on this information they generated new algorithm candidates that were then manually added to the ongoing race. In fact, one of these newly injected candidate algorithms was finally the best performing algorithm in an international timetabling competition (see also `http://www.idsia.ch/Files/ttcomp2002`).

The PhD work of Matthijs den Besten [27] provides an empirical investigation into the application of ILS to solve a range of deterministic scheduling problems with tardiness penalties. Racing, in general, and *F-Race*, in particular, is a very important ingredient throughout the algorithm development and calibration. The ILS algorithms are built in a modular way and *F-Race* is applied to assess each combination of modular components of the algorithm.

Comparison of *F-Race* to other methods There have been some comparisons of *F-Race* with other racing algorithms. Some preliminary results comparing *F-Race* and t-test based racing techniques are presented by Birattari [14, 15], showing that *F-Race* typically performs best.

Yuan and Gallagher [57] discuss the use of *F-Race* for the empirical evaluation of evolutionary algorithms. They also use an algorithm called *A-Race*, where the family-wise test is based on the *analysis of variance* (ANOVA) method. From the experiments they conduct, they conclude that their version of an *F-Race* obtains better results than *A-Race*.

Caelen and Bontempi in their work [20] compare five techniques from various communities on a model selection task. The techniques compared

are (i) a two-stage selection technique proposed in the stochastic simulation community, (ii) a stochastic dynamic programming approach conceived to address the multi-armed bandit problem, (iii) a racing method, (iv) a greedy approach, (v) a round-search technique. *F-Race* is mentioned and applied for comparison purposes. The comparison results shows that the bandit strategy yields the most promising performance when the sample size is small, but *F-Race* outperforms other techniques when the sample size is sufficiently large.

Extensions and Hybrids of *F-Race*. The *F-Race* algorithm has been adopted as a module integrated into an ACO algorithm framework for tackling combinatorial optimization problems under uncertainty [16]. The resulting algorithm is called *ACO/F-Race* and it uses *F-Race* to determine the best of a set of candidate solutions generated by the ACO algorithm. In later work by Balaprakash et al. [6] on the application of estimation-based ACO algorithms to the probabilistic traveling salesman problem the Friedman test is replaced by an ANOVA.

Yuan and Gallagher [58, 59] propose an approach to tune evolutionary algorithms by hybridizing Meta-EA and *F-Race*. Meta-EA is an approach that uses various genetic operators to tune the parameters of EAs. It is reported that one major difficulty in Meta-EA is that it cannot handle effectively categorical parameters. These categorical parameters are usually handled in Meta-EA by pure random search. The proposed hybridization uses Meta-EA to evolve part of the numerical parameters and leave the categorical parameters for *F-Race*. Experiment show that Meta-EA plus *F-Race* required only around 12% of the computational effort taken by Meta-EA plus random search.

Table 5.1 lists the number and types of parameters in the literature that has applied *F-Race* on. It is shown that *F-Race* (or iterated *F-Race*) has been applied to tuning maximum 11 parameters in practice, however, that does not impose a limit for *F-Race* or iterated *F-Race*. It is also shown that most of the *F-Race* applications up to now obtain the initial set of parameter settings for *F-Race* by a full factorial design. Usually not only the categorical parameters are treated by full factorial design, but also the numerical parameters are first discretized and then extracted by full factorial design. The sampling design methods for *F-Race* discussed in this article, more specifically the iterated *F-Race*, will be substantially useful for the conventional *F-Race* users.

Table 5.1: The status of the parameter space that *F-Race* has been applied to but not limited to.

literature	algorithm	# param	# num	# cat	# cond	# conf
[23, 50]	SA	5	4	1	1	78
	ILS	5	2	3	3	152
[24, 21]	TS	1	1	0	0	10
	HEA	2	2	0	0	4
	GLS	1	1	0	0	4
	MC	1	1	0	0	4
[36]	na					
[51]	ils	6	5	1	4	78
	ma	na	na	na	na	144
[27]	ils	4	2	2	0	81
[49]	na					
[30]	ts	2	2	0	0	-
[28]	ts 2	2	0	0	-	
[12]	na					
[45]	rnn	3	3	0	0	2744
[46]	oGA	2	2	0	0	44
	SGA	3	3	0	0	220
[18, 54, 55]	ACOR	2	2	0	0	-
	ACOR-BP	3	3	0	0	-
	ACOR-LM	3	3	0	0	-
	BP	1	1	0	0	-
	LM	1	1	0	0	-
[43, 42]	ctrl	3	3	0	0	27
[10, 9]	na					
[22]	hyb	6	4	2	0	1185
[57]	ea	2	2	0	0	60
[20]	ms	1	0	1	0	300 - 1200
[58, 59]	ea	6	3	3	3	66
[5]	eeais	4	4	0	0	-
[6]	mmas,bwas	4	4	0	0	-
	acs,ras	5	5	0	0	-
[60]	gr	2	2	0	0	500
	ig	5	5	0	0	500
	ia	11	11	0	0	500
[58, 59]	ea	6	3	3	3	66

Chapter 6

Summary and Outlook

In this article, we have presented the algorithm configuration problem that *F-Race* tackles and have given a detailed review of the method. *F-Race* essentially is selection method of the best algorithm configuration under stochastic evaluations. As such, it is a method that is independent of the way the candidate configurations are sampled. In a next step, we have introduced the family of iterated *F-Race* algorithms, where the sampling of new candidate configurations is done through probability models that are iteratively refined. Three variants of iterated *F-Race* are presented, including an ad-hoc *I/F-Race*, `MADS/F-Race` and `BOBYQA/F-Race`.

There is a significant number of possible extensions and adaptations of the *F-Race* method. In fact, any mixed-integer (stochastic) optimization techniques could at least in principle provide the sampling method for an iterated *F-Race*. A part of our current research is actually devoted to this observation. We are currently studying the usage of *F-Race* on top of continuous optimization methods and first results show statistically significant advantages over strategies using a fixed sample size. Combinations of *F-Race* with other methods for parameter tuning such as SPO [8] and local search approaches [33] may also be useful. Finally, we believe that the ideas on which *F-Race* is based can be also fruitful for other tasks than algorithm tuning. In fact, we envision that especially applications to stochastic optimization problems may benefit very much, *ACO/F-Race* [16] being a first such successful example.

Bibliography

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

[2] C. Audet and JE Dennis. Analysis of generalized pattern searches. *SIAM Journal on Optimization*, 13:889–903, 2000.

[3] C. Audet and JE Dennis. Mesh adaptive direct search algorithms for constrained optimization. *SIAM Journal on optimization*, 17(1):188–217, 2007.

[4] C. Audet and D. Orban. Finding optimal algorithmic parameters using derivative-free optimization. *SIAM Journal on Optimization*, 17:642, 2006.

[5] P. Balaprakash, M. Birattari, T. Stützle, and M. Dorigo. Adaptive sample size and importance sampling in estimation-based local search for the probabilistic traveling salesman problem. *European Journal of Operational Research*, 199(1):98–110, 2009.

[6] P. Balaprakash, M. Birattari, T. Stützle, Z. Yuan, and M. Dorigo. Ant colony optimization and estimation-based local search for the probabilistic traveling salesman problem. *Swarm Intelligence*, 3(3):223–242, 2009.

[7] P. Balaprakash, M. Birattari, and T.Stützle. Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In T. Bartz-Beielstein et al., editors, *Hybrid Metaheuristics, 4th International Workshop, HM 2007*, volume 4771 of *Lecture Notes in Computer Science*, pages 108–122. Springer Verlag, Berlin, Germany, 2007.

[8] T. Bartz-Beielstein. *Experimental Research in Evolutionary Computation*. Springer Verlag, Berlin, Germany, 2006.

[9] S. Becker. Racing-Verfahren für Tourenplanungsprobleme. Diplomarbeit, Technische Universität Darmstadt, Darmstadt, Germany, 2004.

[10] S. Becker, J. Gottlieb, and T. Stützle. Applications of racing algorithms: An industrial perspective. In E.-G. Talbi et al., editors, *Artificial Evolution: 7th International Conference, Evolution Artificielle, EA 2005*, volume 3871 of *Lecture Notes in Computer Science*, pages 271–283, Lille, France, 2005. Springer Verlag, Berlin, Germany.

[11] P. Billingsley. *Probability and Measure*. John Wiley & Sons, New York, NY, USA, second edition, 1986.

[12] M. S. bin Hussin, T. Stützle, and M. Birattari. A study of stochastic local search algorithms for the quadratic assignment problems. In E. Ridge et al., editors, *Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, pages 11–15, Brussels, Belgium, September 2007.

[13] M. Birattari. On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Technical Report TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[14] M. Birattari. *The Problem of Tuning Metaheuristics as Seen from a Machine Learning Perspective*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2004.

[15] M. Birattari. *Tuning Metaheuristics: A Machine Learning Perspective*, volume 197 of *Studies in Computational Intelligence*. Springer Verlag, Berlin, Germany, 2009.

[16] M. Birattari, P. Balaprakash, and M. Dorigo. The ACO/F-Race algorithm for combinatorial optimization under uncertainty. In K. F. Doerner et al., editors, *Metaheuristics - Progress in Complex Systems Optimization*, Operations Research/Computer Science Interfaces Series, pages 189–203. Springer Verlag, Berlin, Germany, 2007.

[17] M Birattari, T. Stützle, L. Paquete, and K. Varrentrapp. A racing algorithm for configuring metaheuristics. In W. B. Langdon et al., editors, *GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference*, pages 11–18. Morgan Kaufmann Publishers, San Francisco, CA, 2002.

[18] C. Blum and K. Socha. Training feed-forward neural networks with ant colony optimization: An application to pattern classification. In N. Nedjah et al., editors, *Proceedings of Fifth International Conference on Hybrid Intelligent Systems (HIS'05)*, pages 233–238, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[19] K.P. Burnham and D.R. Anderson. *Model selection and multimodel inference: a practical information-theoretic approach*. Springer, 2002.

[20] O. Caelen and G. Bontempi. How to allocate a restricted budget of leave-one-out assessments for effective model selection in machine learning: a comparison of state-of-the-art techniques. In K. Verbeeck et al., editors, *Proceedings of the 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'05)*, pages 51–58, Brussels, Belgium, 2005.

[21] M. Chiarandini. *Stochastic local search methods for highly constrained combinatorial optimisation problems*. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany, 2005.

[22] M. Chiarandini, M. Birattari, K. Socha, and O. Rossi-Doria. An effective hybrid algorithm for university course timetabling. *Journal of Scheduling*, 9(5):403–432, 2006.

[23] M. Chiarandini and T. Stützle. Experimental evaluation of course timetabling algorithms. Technical Report AIDA-02-05, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany, 2002.

[24] M. Chiarandini and T. Stützle. Stochastic local search algorithms for graph set t-colouring and frequency assignment. *Constraints*, 12(3):371–403, 2007.

[25] W. J. Conover. *Practical Nonparametric Statistics*. John Wiley & Sons, New York, NY, USA, third edition, 1999.

[26] A. Dean and D. Voss. *Design and Analysis of Experiments*. Springer Verlag, New York, NY, USA, 1999.

[27] M. L. den Besten. *Simple Metaheuristics for Scheduling. An empirical investigation into the application of iterated local search to deterministic scheduling problems with tardiness penalities*. PhD thesis, FG Intellektik, FB Informatik, TU Darmstadt, 2004.

[28] L. Di Gaspero and A. Roli. Stochastic local search for large-scale instances of the haplotype inference problem by pure parsimony. *Journal of Algorithms*, 63(1-3):55–69, 2008.

[29] M. Dorigo and T. Stützle. *Ant colony optimization*. MIT Press, Cambridge, MA, 2004.

[30] L. Di Gaspero, G. di Tollo, A. Roli, and A. Schaerf. Hybrid local search for constrained financial portfolio selection problems. In *Integration of AI and OR Techniques in Constraint Programming for Combinatorial Optimization Problems*, volume 4510 of *Lecture Notes in Computer Science*, pages 44–58. Springer Verlag, Berlin, Germany, 2007.

[31] H. H. Hoos and T. Stützle. *Stochastic Local Search. Foundations and Applications*. Morgan Kaufmann Publishers, San Francisco, CA, USA, 2004.

[32] F. Hutter, H.H. Hoos, K. Leyton-Brown, and K.P. Murphy. An Experimental Investigation of Model-Based Parameter Optimisation: SPO and Beyond. In *Proc. of GECCO-09*, 2009.

[33] Frank Hutter, Holger H. Hoos, and Thomas Stützle. Automatic algorithm configuration based on local search. In R. C. Holte et al., editors, *Proceedings of the 22nd Conference on Artificial Intelligence (AAAI)*, pages 1152–1157. AAAI Press / The MIT Press, Menlo Park, CA, USA, 2007.

[34] D. S. Johnson, L. A. McGeoch, C. Rego, and F. Glover. 8th DIMACS implementation challenge. `http://www.research.att.com/~dsj/chtsp/` (webpage last visited in April 2009).

[35] R. Lenne, C. Solnon, T. Stützle, E. Tannier, and M. Birattari. Effective stochastic local search algorithms for the genomic median problem. In E. Ridge et al., editors, *Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms*, pages 1–5, Brussels, Belgium, September 2007.

[36] M. Manfrin. Metaeuristiche per la costruzione degli orari dei corsi universitari. Tesi di Laurea, Università degli Studi di Firenze, Firenze, Italy, 2003. In Italian.

[37] O. Maron. Hoeffding races: Model selection for MRI classification. Master's thesis, The Massachusetts Institute of Technology, Cambridge, MA, USA, 1994.

[38] O. Maron and A. W. Moore. Hoeffding races: Accelerating model selection search for classification and function approximation. In J. D. Cowan et al., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 59–66, San Francisco, CA, USA, 1994. Morgan Kaufmann Publishers.

[39] O. Maron and A. W. Moore. The racing algorithm: Model selection for lazy learners. *Artificial Intelligence Review*, 11(1–5):193–225, 1997.

[40] D. C. Montgomery. *Design and Analysis of Experiments*. John Wiley & Sons, New York, NY, USA, fifth edition, 2000.

[41] J.J. More and S.M. Wild. Benchmarking derivative-free optimization algorithms. *SIAM Journal on Optimization*, 20(1):172–191, 2009.

[42] S. Nouyan. *Teamwork in a Swarm of Robots – An Experiment in Search and Retrieval*. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium, 2008.

[43] S. Nouyan, A. Campo, and M. Dorigo. Path formation in a robot swarm. *Swarm Intelligence*, 2(1):1–23, 2008.

[44] A. Papoulis. *Probability, Random Variables, and Stochastic Processes*. McGraw-Hill, New York, NY, USA, third edition, 1991.

[45] P. Pellegrini. Application of two nearest neighbor approaches to a rich vehicle routing problem. Technical Report TR/IRIDIA/2005-15, IRIDIA, Université Libre de Bruxelles, Belgium, 2005.

[46] C. Philemotte and H. Bersini. The gestalt heuristic: learning the right level of abstraction to better search the optima. Technical Report TR/IRIDIA/2008-021, IRIDIA, Université Libre de Bruxelles, Belgium, 2008.

[47] M. J. D. Powell. *Large-Scale Nonlinear Optimization*, volume 83 of *Nonconvex Optimization and Its Applications*, chapter The NEWUOA software for unconstrained optimization, pages 255–297. Springer-Verlag, Berlin, Germany, 2006.

[48] M. J. D. Powell. The bobyqa algorithm for bound constrained optimization without derivatives. Technical Report NA2009/06, Cambridge NA Reports, 2009.

[49] M. Risler, M. Chiarandini, L. Paquete, T. Schiavinotto, and T. Stützle. An algorithm for the car sequencing problem of the ROADEF 2005 challenge. Technical Report AIDA–04–06, FG Intellektik, TU Darmstadt, Darmstadt, Germany, March 2004.

[50] O. Rossi-Doria, M. Sampels, M. Birattari, M. Chiarandini, M. Dorigo, L. M. Gambardella, J. Knowles, M. Manfrin, M. Mastrolilli, B. Paechter, L. Paquete, and T. Stützle. A comparison of the performance of different metaheuristics on the timetabling problem. In E. Burke et al., editors, *Practice and Theory of Automated Timetabling IV*, volume 2740 of *Lecture Notes in Computer Science*, pages 329–351. Springer Verlag, Berlin, Germany, 2003.

[51] T. Schiavinotto and T. Stützle. The linear ordering problem: Instances, search space analysis and algorithms. *Journal of Mathematical Modelling and Algorithms*, 3(4):367–402, 2004.

[52] D. Sheskin. *Handbook of Parametric and Nonparametric Statistical Procedures*. Chapman & Hall/CRC, Boca Raton, FL, USA, second edition, 2000.

[53] S. Siegel and N. J. Castellan, Jr. *Non Parametric Statistics for the Behavioral Sciences*. McGraw-Hill, New York, NY, USA, second edition, 1988.

[54] K. Socha and C. Blum. Ant colony optimization. In *Metaheuristic Procedures for Training Neutral Networks*, volume 36 of *Operations Research/Computer Science Interfaces*, pages 153–180. Springer US, 2006.

[55] K. Socha and C. Blum. An ant colony optimization algorithm for continuous optimization: application to feed-forward neural network training. *Neural Computing and Applications*, 16(3):235–247, 2007.

[56] T. Stützle and H. H. Hoos. $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. *Future Generation Computer Systems*, 16(8):889–914, 2000.

[57] B. Yuan and M. Gallagher. Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In X. Yao et al., editors, *Parallel Problem Solving from Nature - PPSN VIII*, volume 3242 of *Lecture Notes in Computer Science*, pages 172–181. Springer Verlag, Berlin, Germany, 2004.

[58] B. Yuan and M. Gallagher. A hybrid approach to parameter tuning in genetic algorithms. In *Proceedings of the IEEE Congress in Evolutionary Computation (CEC'05)*, volume 2, pages 1096–1103. IEEE Press, Piscataway, NJ, 2005.

[59] B. Yuan and M. Gallagher. Combining Meta-EAs and racing for difficult ea parameter tuning tasks. In *Parameter Setting in Evolutionary Algorithms*, volume 54 of *Studies in Computational Intelligence*, pages 121–142. Springer Verlag, Berlin, Germany, 2007.

[60] Z. Yuan, A. Fügenschuh, H. Homfeld, P. Balaprakash, T. Stützle, and M. Schoch. Iterated greedy algorithms for a real-world cyclic train scheduling problem. In M. J. Blesa et al., editors, *Hybrid Metaheuristics, 5th International Workshop, HM 2008*, volume 5296 of *Lecture Notes in Computer Science*, pages 102–116. Springer Verlag, Berlin, Germany, 2008.

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

Index