Chapter 13
F-Race and Iterated F-Race: An Overview

Mauro Birattari, Zhi Yuan, Prasanna Balaprakash, and Thomas Stützle

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 chapter, 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.

13.1 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, and different types of local search algorithms in

Mauro Birattari · Zhi Yuan · Prasanna Balaprakash · Thomas Stützle
IRIDIA, CoDE, Université Libre de Bruxelles, Brussels, Belgium
e-mail: {mbiro,zyuan,pbalapra,stuetzle}@ulb.ac.be

T. Bartz-Beielstein et al. (eds.), *Experimental Methods for the Analysis of Optimization Algorithms*, DOI 10.1007/978-3-642-02538-9_13,

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 (Adenso-Diaz and Laguna 2006, Birattari 2009, Birattari et al. 2002). 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 (Hoos and Stützle 2004). However, even if an algorithm is deterministic, its performance and search behavior depend 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 (Maron 1994, Maron and Moore 1994, 1997). 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 nonparametric 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 chapter, we first formally describe the algorithm configuration problem, following Birattari et al. (2002) and Birattari (2009). Next, in Sect. 13.3, we give details on `F-Race`. Section 13.4 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 by Balaprakash et al. (2007). Computational results with this new variant, which are presented in Sect. 13.5, confirm its advantage over other ways of generating the candidate configurations for `F-Race`. We end the chapter with an overview of available `F-Race` applications and outline ideas for further research.

13.2 The Algorithm Configuration Problem

F-Race is a method for 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 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 of 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.

13.2.1 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.
- $t : I \to \mathbb{R}$ 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]
- $C \subset \mathbb{R}$ 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.

[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.

- 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

$$\bar{\theta} = \arg\min_{\theta \in \Theta} \mathcal{C}(\theta). \tag{13.1}$$

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

$$\mathcal{C}(\theta) = E_{I,C}[c] = \int c \, \mathrm{d}p_C(c|\theta, i) \, \mathrm{d}p_I(i), \tag{13.2}$$

where the expectation is considered with respect to both p_I and p_C, and the integration is taken in the Lebesgue sense (Billingsley 1986). 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.

The measures p_I and p_C are usually not explicitly available and the analytical solution of the integrals in (13.2) 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 (13.2) 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 on 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 (Hutter et al. 2007). 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 no longer 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 (2004a, 2009) that sampling N instances

[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.

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 a few training instances are provided, one needs to go back to evaluating algorithm configurations on the instances more than once.

13.2.2 Types of Parameters

As said in the introduction, algorithms can have different types of parameters. There we 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 generally, 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 disruption 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 configura-

[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

tion algorithms described in this chapter are able to handle all the aforementioned types of parameters, including conditional parameters.

13.3 `F-Race`

The conceptually simplest approach for estimating the expected cost of an algorithm configuration θ, as defined by (13.2), 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.

13.3.1 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 (Maron and Moore 1994), where it was first proposed for solving the model selection problem (Burnham and Anderson 2002). 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 parallel,[5] 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 evi-

---

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.

[5] 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.

dence is gathered against them. A graphical illustration of the *racing* algorithm and the *brute-force* approach is shown in Fig. 13.1.

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 kth 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 Sect. 13.4. 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 instances have been sampled, or when the predefined computational budget B has been exhausted.[6]

13.3.2 The Peculiarity of **F−Race**

F−Race is a racing algorithm based on the nonparametric Friedman's two-way analysis of variance by ranks (Conover 1999), for short, Friedman test. This algorithm was first proposed by Birattari et al. (2002) and studied in detail in Birattari's PhD thesis (Birattari 2004b).

---

6 The computational budget may be measured as a total available computation time T (see the definition of the configuration problem on page 313). It is, however, often more convenient to define the maximum number of function evaluations, if each function evaluation is limited to the same amount of computation time.

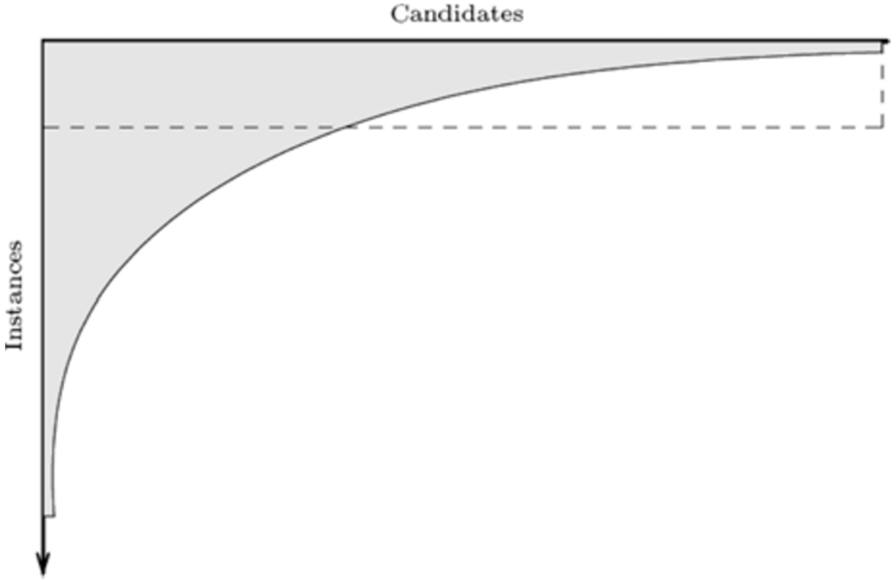

Fig. 13.1: Graphical representation of the allocation of configuration evaluations by the *racing* approach and the *brute-force* approach. In the *racing* approach, as soon as sufficient evidence is gathered that a candidate is suboptimal, such a candidate is discarded from further evaluation. As the evaluation proceeds, the *racing* approach thus focuses 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

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

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

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 nondecreasing 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 (Conover 1999):

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

Under the null hypothesis that all candidates are equivalent, T is approximately χ^2 distributed with $m-1$ degrees of freedom (Papoulis 1991). 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 (1999): 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* (Sheskin 2000). However, in the `F-Race` algorithm, the *Wilcoxon matched-pairs signed-ranks test* (Conover 1999) is adopted, for the reason that the Wilcoxon test is more powerful and data efficient than the binomial sign test in such a case (Siegel and Castellan 1988).

In `F-Race`, the test statistic is based on the ranking of the candidates. Ranking plays an important twofold role. The first one is due to the nonparametric 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 (Dean and Voss 1999, Montgomery 2000). By focusing only on the ranking of the different configurations within each instance, this blocking design becomes an effective way of normalizing the costs observed on different instances.

13.4 The Sampling Strategy for F-Race

In the previous section, the question of 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.

13.4.1 Full Factorial Design

When `F-Race` was first proposed by Birattari et al. (2002), the candidate configurations were collected by a full factorial design (*FFD*) on the parameter space. The reason for 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 with 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)`.

13.4.2 Random Sampling Design

The drawbacks of the full factorial design were described also by Balaprakash et al. (2007). 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.[7] 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

[7] 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 13.1: 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^*

assume a uniform distribution over X. We denote the random sampling version of F-Race based on uniform distributions by F-Race(RSD).

Two main advantages of the random sampling design are that of 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.

13.4.3 Iterated F-Race

As a next step, Balaprakash et al. (2007) 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 (Zlochin et al. 2004), 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 of 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. 13.1.

There are many possible ways in which iterated F-Race can be implemented. In fact, one possibility would be to use some algorithms for black-box mixed 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, Balaprakash et al. (2007) followed a different approach and an ad hoc method was proposed for biasing the sampling. Unfortunately, there the ad hoc iterated F-Race was 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 Sect. 13.4.4. 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, fewer 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 fewer 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 lth 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 quasicontinuous 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 (Balaprakash et al. 2007), 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.

13.4.4 An Example Iterated F-Race Algorithm

Here we describe one example implementation of iterated F-Race, which we refer to as I/F-Race in the following. This example implementation is based on the previous one published by Balaprakash et al. (2007). 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 is 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 $L = 2 + \mathrm{round}(\log_2 d)$.

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 $B_l = (B - B_{\mathrm{used}})/(L - l + 1)$; B_{used} denotes the total computational budget used until iteration $l - 1$.

The number of candidate configurations. We introduce a parameter 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 $N_{\min} = 2 + \mathrm{round}(\log_2 d)$ 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 $N_s = \min(N_{\text{survive}}, N_{\min})$, 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

$$w_z = \frac{N_s - r_z + 1}{N_s \cdot (N_s + 1)/2}. \tag{13.3}$$

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 being denoted by X^{num}) or a categorical one (the set of such parameters being denoted by X^{cat}). We have that the parameter space $X = X^{\text{num}} \cup X^{\text{cat}}$.
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 by $v_i = \overline{X_i} - \underline{X_i}$ 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

$$\sigma_i^{l+1} = v_i \cdot \left(\frac{1}{N_{l+1}}\right)^{\frac{l}{d}} \qquad \text{for } l = 1, \ldots, L - 1. \tag{13.4}$$

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 $X_i \in X^{\text{cat}}$ 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 lth 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

$$p_{l+1}(f_j) = p_l(f_j)\cdot(1-\frac{l}{L})+I_{j=f_i^z}\cdot\frac{l}{L} \quad \text{for } l = 1,\ldots,L-1 \text{ and } j = 1,\ldots,n_i \tag{13.5}$$

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.

13.5 Case Studies

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* (MMAS), a particularly successful ACO algorithm (Stützle and Hoos 2000), using four categorical parameters and configuring MMAS using seven categorical parameters. Both case studies use the MMAS implementation available in the ACOTSP software package.[8] 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 $10,000$. We generated $1,000$ instances for training and 300 for evaluating the winning configurations using the DIMACS instance generator (Johnson et al. 2001). 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 from all six combinations of two different computation time limits allocated for each function evaluation to the ACOTSP software (5 and 20 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, ten trials were run. Each trial is the execution of the configuration process, in our case, either F-Race(FFD),

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

Parameter	Range	No. of levels
α	$[0.01, 5.00]$	11
β	$[0.01, 10.00]$	11
ρ	$[0.00, 1.00]$	10
m	$[5, 100]$	10

Table 13.1: 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 $12,100$

`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.

13.5.1 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 13.1.[9] 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 (Balaprakash et al. 2007).

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)`.

The experimental results are given in Table 13.2. 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 nonparametric 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.

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

	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

Table 13.2: Computational results for configuring $\mathcal{MM}$AS for the TSP with four discretized parameters for a computation time bound of 5 and 20 s, respectively. The column entries with the label per.dev show 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

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 in 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 ($12,100$ candidate configurations) and that the number of levels (10 or 11) for each parameter is large.

13.5.2 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 pseudorandom proportional action choice rule; for a detailed definition see Dorigo and Stützle (2004).

The parameters are discretized using the ranges and number of levels given in Table 13.3. Note that, in comparison with the previous experiment, the parameter space is more than one order of magnitude larger ($259,200 \gg 12,100$). Besides, there is a 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

Parameter	Range	No. of 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

Table 13.3: 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 $259,200$

	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

Table 13.4: Computational results for configuring $\mathcal{MM}$AS for TSP with seven categorical parameters in 5 and 20 CPU s. For an explanation of the table entries see the caption of Table 13.2

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 13.4 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.

13.5.3 Case Study 3: ACOTSP under 12 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), and

	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

Table 13.5: Computational results for configuring $\mathcal{MM}$AS for TSP with 12 parameters in 5 and 20 CPU s. For an explanation of the table entries see the caption of Table 13.2

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 quasicontinuous parameters m and nn, which have been introduced before. Moreover, five conditional parameters are considered: (i) the continuous parameter q_0 (introduced in Sect. 13.5.2) 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 13.5. The two algorithms F−Race(RSD) and I/F−Race are compared using the nonparametric pairwise Wilcoxon test using a 0.05 significance level. The statistical comparisons show that I/F−Race is again dominating. It is significantly better performing in five out of six experiments; only in one case can no statistically significant difference be identified. However, the quality differences in this set of experiments are quite small, usually below 0.1% in the 5 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.

13.6 A Review of F−Race Applications

F−Race has received significant attention, as witnessed by the 99 citations to the first article on F−Race (Birattari et al. 2002) in the Google Scholar database as of June 2009. In what follows, we give an overview of research 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, 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 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 (Chiarandini and Stützle 2002, Manfrin 2003, Rossi-Doria et al. 2003) and also for various other problems (Chiarandini 2005, Chiarandini and Stützle 2007, den Besten 2004, Risler et al. 2004, Schiavinotto and Stützle 2004).

Soon after these initial applications, F-Race was also adopted by a number of other researchers. Most applications focus on configuring SLS methods for combinatorial optimization problems (Bin Hussin et al. 2007, Balaprakash et al. 2009a, Di Gaspero and Roli 2008, Di Gaspero et al. 2007, Lenne et al. 2007, Pellegrini 2005, Philemotte and Bersini 2008). However, also other applications have been considered, including the tuning of algorithms for training neural networks (Blum and Socha 2005, Socha and Blum 2007) or the tuning of parameters of a control system for simple robots (Nouyan 2008, Nouyan et al. 2008).

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 company 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 with 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 by Becker et al. (2005); more details are available in a master thesis (Becker 2004).

Yuan et al. (2008) 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. (2006) 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 den Besten (2004) 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 with 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 (2004b, 2009), showing that F-Race typically performs best.

Yuan and Gallagher (2004) 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 F-Race obtains better results than *A-Race*.

In their work Caelen and Bontempi (2005) 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, and (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 (Birattari et al. 2007). 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. (2009b) 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 (2005, 2007) 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 effectively handle 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.

13.7 Summary and Outlook

In this chapter, we have presented the algorithm configuration problem that F−Race tackles and have given a detailed review of the method. F−Race is essentially a method for selecting 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.

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 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 (Bartz-Beielstein 2006) and local search approaches (Hutter et al. 2007) may be also useful. Finally, we believe that the ideas on which F−Race is based can be also fruitful for tasks other than algorithm tuning. In fact, we envision that especially applications to stochastic optimization problems may benefit greatly, ACO/F−Race (Birattari et al. 2007) being a first such successful example.

Acknowledgements This work has been supported by META-X, an ARC project funded by the French Community of Belgium. The authors acknowledge support from the fund for scientific research FRS-FNRS of the French Community of Belgium, of which they are research associates (M.B. and T.S.), or aspirant (Z.Y.), respectively.

References

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

Balaprakash P, Birattari M, Stützle T (2007) Improvement strategies for the F-Race algorithm: Sampling design and iterative refinement. In: Bartz-Beielstein T, et al. (eds) Hybrid Metaheuristics, 4th International Workshop, HM 2007, Lecture Notes in Computer Science, vol 4771, Springer, Berlin, Germany, pp. 108–122

Balaprakash P, Birattari M, Stützle T, Dorigo M (2009a) 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

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

Bartz-Beielstein T (2006) Experimental Research in Evolutionary Computation. Springer, Berlin, Germany

Becker S (2004) Racing-Verfahren für Tourenplanungsprobleme. Diplomarbeit, Technische Universität Darmstadt, Darmstadt, Germany

Becker S, Gottlieb J, Stützle T (2005) Applications of racing algorithms: An industrial perspective. In: Talbi EG, et al. (eds) Artificial Evolution: 7th International Conference, Evolution Artificielle, EA 2005, Springer Verlag, Berlin, Germany, Lille, France, Lecture Notes in Computer Science, vol 3871, pp. 271–283

den Besten ML (2004) 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

Billingsley P (1986) Probability and Measure, 2nd edn. Wiley, New York, NY, USA

Bin Hussin MS, Stützle T, Birattari M (2007) A study of stochastic local search algorithms for the quadratic assignment problems. In: Ridge E, et al. (eds) Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms, Brussels, Belgium, pp. 11–15

Birattari M (2004a) On the estimation of the expected performance of a metaheuristic on a class of instances. How many instances, how many runs? Tech. Rep. TR/IRIDIA/2004-001, IRIDIA, Université Libre de Bruxelles, Brussels, Belgium

Birattari M (2004b) The problem of tuning metaheuristics as seen from a machine learning perspective. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium

Birattari M (2009) Tuning Metaheuristics: A Machine Learning Perspective, Studies in Computational Intelligence, vol 197. Springer, Berlin, Germany

Birattari M, Stützle T, Paquete L, Varrentrapp K (2002) A racing algorithm for configuring metaheuristics. In: Langdon WB, et al. (eds) GECCO 2002: Proceedings of the Genetic and Evolutionary Computation Conference, Morgan Kaufmann Publishers, San Francisco, CA, pp. 11–18

Birattari M, Balaprakash P, Dorigo M (2007) The ACO/F-Race algorithm for combinatorial optimization under uncertainty. In: Doerner KF, et al. (eds) Metaheuristics - Progress in Complex Systems Optimization, Operations Research/Computer Science Interfaces Series, Springer, Berlin, Germany, pp. 189–203

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

Burnham K, Anderson D (2002) Model selection and multimodel inference: a practical information-theoretic approach. Springer

Caelen O, Bontempi G (2005) 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: Verbeeck K, et al. (eds) Proceedings of the 17th Belgian-Dutch Conference on Artificial Intelligence (BNAIC'05), Brussels, Belgium, pp. 51–58

Chiarandini M (2005) Stochastic local search methods for highly constrained combinatorial optimisation problems. PhD thesis, Technische Universität Darmstadt, Darmstadt, Germany

Chiarandini M, Stützle T (2002) Experimental evaluation of course timetabling algorithms. Tech. Rep. AIDA-02-05, FG Intellektik, FB Informatik, Technische Universität Darmstadt, Darmstadt, Germany

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

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

Conover WJ (1999) Practical Nonparametric Statistics, 3rd edn. Wiley, New York, NY, USA

Dean A, Voss D (1999) Design and Analysis of Experiments. Springer, New York, NY, USA

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

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

Dorigo M, Stützle T (2004) Ant colony optimization. MIT Press, Cambridge, MA

Hoos HH, Stützle T (2004) Stochastic Local Search. Foundations and Applications. Morgan Kaufmann, San Francisco, CA, USA

Hutter F, Hoos HH, Stützle T (2007) Automatic algorithm configuration based on local search. In: Holte RC, et al. (eds) Proceedings of the 22nd Conference on Artificial Intelligence (AAAI), AAAI Press / The MIT Press, Menlo Park, CA, USA, pp. 1152–1157

Johnson DS, McGeoch LA, Rego C, Glover F (2001) 8th DIMACS implementation challenge. `http://www.research.att.com/~dsj/chtsp/` (webpage last visited in April 2009)

Lenne R, Solnon C, Stützle T, Tannier E, Birattari M (2007) Effective stochastic local search algorithms for the genomic median problem. In: Ridge E, et al. (eds) Proceedings of SLS-DS 2007, Doctoral Symposium on Engineering Stochastic Local Search Algorithms, Brussels, Belgium, pp. 1–5

Manfrin M (2003) Metaeuristiche per la costruzione degli orari dei corsi universitari. Tesi di Laurea, Università degli Studi di Firenze, Firenze, Italy, in Italian

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

Maron O, Moore AW (1994) Hoeffding races: Accelerating model selection search for classification and function approximation. In: Cowan JD, et al. (eds) Advances in Neural Information Processing Systems, Morgan Kaufmann, San Francisco, CA, USA, vol 6, pp. 59–66

Maron O, Moore AW (1997) The racing algorithm: Model selection for lazy learners. Artificial Intelligence Review 11(1–5):193–225

Montgomery DC (2000) Design and Analysis of Experiments, 5th edn. Wiley, New York, NY, USA

Nouyan S (2008) Teamwork in a swarm of robots – an experiment in search and retrieval. PhD thesis, Université Libre de Bruxelles, Brussels, Belgium

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

Papoulis A (1991) Probability, Random Variables, and Stochastic Processes, 3rd edn. McGraw-Hill, New York, NY, USA

Pellegrini P (2005) Application of two nearest neighbor approaches to a rich vehicle routing problem. Tech. Rep. TR/IRIDIA/2005-15, IRIDIA, Université Libre de Bruxelles, Belgium

Philemotte C, Bersini H (2008) The gestalt heuristic: learning the right level of abstraction to better search the optima. Tech. Rep. TR/IRIDIA/2008-021, IRIDIA, Université Libre de Bruxelles, Belgium

Risler M, Chiarandini M, Paquete L, Schiavinotto T, Stützle T (2004) An algorithm for the car sequencing problem of the ROADEF 2005 challenge. Tech. Rep. AIDA–04–06, FG Intellektik, TU Darmstadt, Darmstadt, Germany

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

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

Sheskin D (2000) Handbook of Parametric and Nonparametric Statistical Procedures, 2nd edn. Chapman & Hall/CRC, Boca Raton, FL, USA

Siegel S, Castellan NJ Jr (1988) Nonparametric Statistics for the Behavioral Sciences, 2nd edn. McGraw-Hill, New York, NY, USA

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

Stützle T, Hoos HH (2000) $\mathcal{MAX}$–$\mathcal{MIN}$ ant system. Future Generation Computer Systems 16(8):889–914

Yuan B, Gallagher M (2004) Statistical racing techniques for improved empirical evaluation of evolutionary algorithms. In: Yao X, et al. (eds) Parallel Problem Solving from Nature - PPSN VIII, Lecture Notes in Computer Science, vol 3242, Springer Verlag, Berlin, Germany, pp. 172–181

Yuan B, Gallagher M (2005) A hybrid approach to parameter tuning in genetic algorithms. In: Proceedings of the IEEE Congress in Evolutionary Computation (CEC'05), IEEE Press, Piscataway, NJ, vol 2, pp. 1096–1103

Yuan B, Gallagher M (2007) Combining Meta-EAs and racing for difficult EA parameter tuning tasks. In: Parameter Setting in Evolutionary Algorithms, Studies in Computational Intelligence, vol 54, Springer Verlag, Berlin, Germany, pp. 121–142

Yuan Z, Fügenschuh A, Homfeld H, Balaprakash P, Stützle T, Schoch M (2008) Iterated greedy algorithms for a real-world cyclic train scheduling problem. In: Blesa MJ, et al. (eds) Hybrid Metaheuristics, 5th International Workshop, HM 2008, Springer Verlag, Berlin, Germany, Lecture Notes in Computer Science, vol 5296, pp. 102–116

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