# A Racing Algorithm for Configuring Metaheuristics

**Mauro Birattari**[†]
IRIDIA
Université Libre de Bruxelles
Brussels, Belgium

**Thomas Stützle, Luis Paquete, and Klaus Varrentrapp**
Intellektik/Informatik
Technische Universität Darmstadt
Darmstadt, Germany

## Abstract

This paper describes a racing procedure for finding, in a limited amount of time, a configuration of a metaheuristic that performs as good as possible on a given instance class of a combinatorial optimization problem. Taking inspiration from methods proposed in the machine learning literature for model selection through cross-validation, we propose a procedure that empirically evaluates a set of candidate configurations by discarding bad ones as soon as statistically sufficient evidence is gathered against them. We empirically evaluate our procedure using as an example the configuration of an ant colony optimization algorithm applied to the traveling salesman problem. The experimental results show that our procedure is able to quickly reduce the number of candidates, and allows to focus on the most promising ones.

## 1 INTRODUCTION

A metaheuristic is a general algorithmic template whose components need to be instantiated and properly tuned in order to yield a fully functioning algorithm. The instantiation of such an algorithmic template requires to choose among a set of different possible components and to assign specific values to all free parameters. We will refer to such an instantiation as a *configuration*. Accordingly, we call *configuration problem* the problem of selecting the optimal configuration.

Practitioners typically configure their metaheuristics in an iterative process on the basis of some runs of different configurations that are felt as promising. Usually, such a process is heavily based on personal experience and is guided by a mixture of rules of thumb. Most often this leads to tedious and time consuming experiments. In addition, it is very rare that a configuration is selected on the basis of some well defined statistical procedure.

The aim of this work is to define an automatic hands-off procedure for finding a good configuration through statistically guided experimental evaluations, while minimizing the number of experiments. The solution we propose is inspired by a class of methods proposed for solving the model selection problem in memory-based supervised learning (Maron and Moore, 1994; Moore and Lee, 1994). Following the terminology introduced by Maron and Moore (1994), we call *racing method for selection* a method that finds a good configuration (model) from a given finite pool of alternatives through a sequence of steps.[1] As the computation proceeds, if sufficient evidence is gathered that some candidate is inferior to at least another one, such a candidate is dropped from the pool and the procedure is iterated over the remaining ones. The elimination of inferior candidates, speeds up the procedure and allows a more reliable evaluation of the promising ones.

Two are the main contributions of this paper. First, we give a formal definition of the metaheuristic configuration problem. Second, we show that a metaheuristic can be tuned efficiently and effectively by a racing procedure. Our results confirm the general validity of the racing algorithms and extend their area of applicability. On a more technical level, left aside the specific application to metaheuristics, we give some contribution to the general class of racing algorithms. In particular, our method adopts blocking design (Dean and Voss, 1999) in a nonparametric setting. In some sense, therefore, the method fills the gap between Hoeffding race (Maron and Moore, 1994) and BRACE (Moore and Lee, 1994): similarly to Hoeffding race it features a nonparametric test, and similarly to BRACE it considers a

[†]This research was carried out while MB was with Intellektik, Technische Universität Darmstadt.

[1]Several metaheuristics involve continuous parameters. This would actually lead to an infinite set of candidate configurations. In practice, typically only a finite set of possible parameter values are considered by discretizing the range of continuous parameters.

blocking design.

The rest of the paper is structured as follows. Section 2 gives a formal definition of the problem of configuring a metaheuristic. Section 3 describes the general ideas behind racing algorithms and introduces F-Race, a racing method specifically designed for matching the peculiar characteristics of the metaheuristic configuration problem. Section 4 proposes some background information on $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System and on the traveling salesman problem (TSP), which are respectively the metaheuristic and the problem considered in this paper. In particular, the section gives a description of the sub class of TSP instances, and of the candidate configurations of $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System that we consider in our experimental evaluation. Section 5 proposes some experimental results, and Section 6 concludes the paper.

## 2 CONFIGURING A METAHEURISTIC

This section introduces and defines the general problem of configuring a metaheuristic. Before proposing a formal definition, it is worth outlining briefly, with the help of an example, the type of problem setting to which our procedure applies. Namely, our methodology is meant to be applied to repetitive problems, that is, problems where many similar instances appear over time.

### 2.1 An Example: Delivering Pizza

The example we propose is admittedly simplistic and does not cover all possible aspects of the configuration problem; still it has the merit of highlighting those elements that are essential for the discussion that follows.

Let us consider the following **pizza delivery problem**. Orders are collected for a (fixed) time period of, say, 30 minutes. At the end of the time period, a pizza delivery boy has some limited amount of time for scheduling a reasonably short tour that visits all the customers that have called in the last 30 minutes. Then the boy leaves and delivers the pizzas following a chosen route. The time available for scheduling may be constant or may be expressed as a function of some characteristic of the instance itself, for example the size which in the pizza delivery problem might be measured by the number of customers to visit.

In such a setting, every 30 minutes a new instance of an optimization problem is given, and a solution as good as possible has to be found in a limited amount of time. It is very likely that every instance will be different from all previous ones in the location of the customers that need to be visited. Further, a certain variability in the instance size, that is the number of customers to be served, is to be expected, too.

The occurrence of different instances can be conveniently represented as the result of random experiments governed by some unknown probability measure, say $P_I$, defined on the class of the possible instances. In the example discussed here, it is reasonable to assume that different experiments are independent and all governed by the same probability measure. In Section 2.3, we will briefly discuss how to possibly tackle situations in which such assumptions appear unreasonable.

Now, our pizza delivery boy loves metaheuristics and uses one to find a shortest possible tour visiting all the customers. Being such a metaheuristic a general algorithmic template, different configurations are possible (see Section 4.2 for a more detailed example). In our setting, the problem that the delivery boy has to solve is to find the configuration that is expected to yield the best solution to the instances that he *typically* faces. The concept of *typical instance*, used here informally, has to be understood in relation to the probability measure $P_I$, and will receive a clear mathematical meaning presently.

Since $P_I$ is unknown, the only information that can be used for finding the best configuration must be extracted from a sample of previously seen instances. By adopting the terminology used in machine learning, we will use the expression *training instances* to denote the available previous instances. On the basis of such training instances, we will look for the configuration that is expected to have the best performance over the *whole* class of possible instances.

The fact of extending results obtained on a usually small training set to a possibly infinite set of instances is a genuine *generalization*, as intended in supervised learning (Mitchell, 1997). In the context of metaheuristics configuration, generalization is fully justified by the assumption that the same probability measure $P_I$ governs the selection of all the instances: both those used for training and those that will be solved afterwards. The training instances are in this sense representative of the whole set of instances.

### 2.2 The Formal Statement

In order to give a formal definition of the general problem of configuring a metaheuristic, we consider the following objects:

- $\Theta$ is the finite set of candidate configurations.
- $I$ is the possibly infinite set of instances.
- $P_I$ is a probability measure over the set $I$ of instances: With some abuse of notation, we indicate with $P_I(i)$ the probability that the instance $i$ is selected for being solved.[2]

[2]Since a probability measure is associated to (sub)sets and not

- $t : I \to \Re$ is a function associating to every instance the computation time that is allocated to it.
- $\boldsymbol{c}(\theta, i) = \boldsymbol{c}(\theta, i, t(i))$ is a random variable representing the cost of the best solution found by running configuration $\theta$ on instance $i$ for $t(i)$ seconds.[3]
- $C \subset \Re$ is the range of $\boldsymbol{c}$, that is, the possible values for the cost of the best solution found in a run of a configuration $\theta \in Theta$ on an instance $i \in I$.
- $P_C$ is a probability measure over the set $C$: With the notation[4] $P_C(c|\theta, i)$, we indicate the probability that $c$ is the cost of the best solution found by running for $t(i)$ seconds configuration $\theta$ 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 $\theta$. In the most general case it measures in some sense the desirability of $\theta$.

On the basis of these concepts, the problem of configuring a metaheuristic can be formally described by the 6-tuple $\langle \Theta, I, P_I, P_C, t, \mathcal{C} \rangle$. The solution of this problem is the configuration $\theta^*$ such that:

$$\theta^* = \arg\min_{\theta} \mathcal{C}(\theta). \tag{1}$$

As far as the criterion $\mathcal{C}$ is concerned, different alternatives are possible. In this paper, we consider the optimization of the expected value of the cost $\boldsymbol{c}(\theta, i)$. Such a criterion is adopted in many different applications and, besides being quite natural, it is often very convenient from both the theoretical and the practical point of view. Formally:

$$\mathcal{C}(\theta) = E_{I,C}\Big[\boldsymbol{c}(\theta, \boldsymbol{i})\Big] = \int_I \int_C c(\theta, i)\, \mathrm{d}P_C(c|\theta, i)\, \mathrm{d}P_I(i), \tag{2}$$

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).

The measures $P_I$ and $P_C$ are usually not explicitly available and the analytical solution of the integrals in Equation 2, one for each configuration $\theta$, is not possible. In order to overcome such a limitation, the integrals defined in Equation 2 will be estimated in a Monte Carlo fashion on the basis of a training set of instances, as it will be explained in Section 3.

to single elements, the correct notation should be $P_I(\{i\})$. Our notational abuse consists therefore in using the same symbol $i$ both for the element $i \in I$, and for the singleton $\{i\} \subset I$.

[3] In the following, for the sake of a lighter notation, the dependency of $\boldsymbol{c}$ on $t$ will be often implicit.

[4] The same remark as in Note 2 applies here.

### 2.3 Further Considerations and Possible Extensions

The formal configuration problem, as described in Section 2.2, assumes that, as far as a given instance is concerned, no information on the performance of the various candidate configurations can be obtained prior to their actual execution on the instance itself. In this sense, the instances are *a priori* indistinguishable.

In many practical situations, it is known *a priori* that various types of instances with different characteristics may arise. In such a situation all possible prior knowledge should be used to cluster the instances into homogeneous classes and to find, for each class, the most suitable configuration.

The case mentioned in Section 2.1, in which it is not reasonable to accept that all instances are extracted independently and according to the same probability measure, can possibly be handled in a similar way. Often, some temporal correlation is observed among instances. In other words, temporal patterns can be observed on previous instances that bring *a priori* information on the characteristics of the current instance. This phenomenon can be handled by assuming that the instances are generated by a process akin to a time-series. Also in this case, different configuration problems should be formulated: Each class of instances to be treated separately would be composed by instances that follow in time a given pattern and that are therefore supposed to share similar characteristics. The aim is again to match the hypothesis of *a priori* indistinguishability of instances within each of the different configuration problems in which the original one is reformulated.

## 3 A RACING ALGORITHM

Before giving a definition of a racing algorithm for solving the problem given in Equation 1, it is convenient to describe a somewhat naive *brute-force* approach for highlighting some of the difficulties associated with the configuration problem.

A brute-force approach to the problem defined in Equation 1 consists in estimating the quantities defined in Equation 2 by means of a *sufficiently large* number of runs of each candidate on a *sufficiently large* set of training instances. The candidate configuration with the smallest estimated quantity is then selected.

However, such a *brute-force* approach presents some drawbacks: First, the size of the training set must be defined prior to any computation. A criterion is missing to avoid considering, on the one hand, too few instances, which could prevent from obtaining reliable estimates, and on the other hand, too many instances, which would then require a great deal of useless computation. Second, no criterion

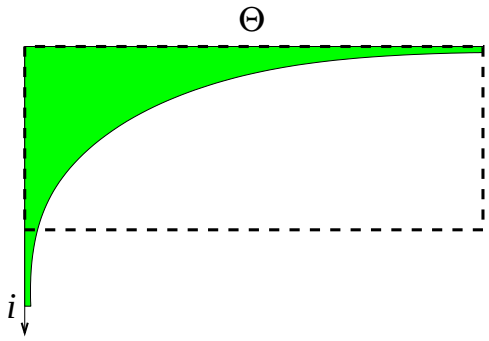


Figure 1: A visual representation of the amount of computation needed by the two methods. The surface of the dashed rectangle represents the amount of computation for brute-force, the shadowed area the one for racing.

is given for deciding how many runs of each configuration on each instance should be performed in order to cope with the stochastic nature of metaheuristics. Finally, the same computational resources are allocated to each configuration: manifestly poor configurations are thoroughly tested to the same extent as the best ones are.

### 3.1 Racing Algorithms: The Idea

Racing algorithms are designed to provide a better allocation of computational resources among candidate configurations and therefore to overcome the last of the three above described drawbacks of brute-force. At the same time, the racing framework indirectly allows for a clean solution to the first two problems of brute-force, that is the problems of fixing the number of instances and the number of runs to be considered.

To do so, racing algorithms sequentially evaluate candidate configurations and discard poor ones as soon as statistically sufficient evidence is gathered against them. The elimination of inferior candidates speeds up the procedure and allows to evaluate the promising configurations on more instances and to obtain more reliable estimates of their behavior. Figure 1 visualizes the two different ways of allocating computational resources to candidate configurations that are adopted by brute-force and by racing algorithms.

Let us suppose that a random sequence of training instances $\underline{\boldsymbol{i}}$ is available, where the generic $k$-th term $\underline{\boldsymbol{i}}_k$ is drawn from $I$ according to $P_I$, independently for each $k$. We assume that $\underline{\boldsymbol{i}}$ can be extended at will and at a negligible cost, by sampling further from $I$.

With the notation $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ we indicate an array of $k$ terms whose generic $l$-th one is the cost $\boldsymbol{c}(\theta, \underline{\boldsymbol{i}}_l)$ of the best solution found by configuration $\theta$ on instance $\underline{\boldsymbol{i}}_l$ in a run of $t(\underline{\boldsymbol{i}}_l)$ seconds. It is clear therefore that, for a given $\theta$, the array $\underline{\boldsymbol{c}}^k$ of length $k$ can be obtained from $\underline{\boldsymbol{c}}^{k-1}$ by appending to the latter the cost concerning the $k$-th instance in $\underline{\boldsymbol{i}}$.

A racing algorithm tackles the optimization problem in Equation 1 by generating a sequence of nested sets of candidate configurations:

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

starting from $\Theta_0 = \Theta$. The step from a set $\Theta_{k-1}$ to $\Theta_k$ is obtained by possibly discarding some configurations that appear to be suboptimal on the basis of information available at step $k$.

At step $k$, when the set of candidates still in the race is $\Theta_{k-1}$, a new instance $\underline{\boldsymbol{i}}_k$ is considered. Each candidate $\theta \in \Theta_{k-1}$ is executed on $\underline{\boldsymbol{i}}_k$ and each observed cost $\boldsymbol{c}(\theta, \underline{\boldsymbol{i}}_k)$ is appended to the respective $\underline{\boldsymbol{c}}^{k-1}$ to form the different arrays $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$, one for each $\theta$. Step $k$ terminates defining set $\Theta_k$ by dropping from $\Theta_{k-1}$ the configurations that appear to be suboptimal in the light of some statistical test that compares the arrays $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ for all $\theta \in \Theta$. The description of the test considered in this paper is given in Section 3.2. It should be noticed here that, for any $\theta$, each component of the array $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$, that is, any cost $\boldsymbol{c}(\theta, \boldsymbol{i})$ of the best solution found by a single run of $\theta$ over one generic $\boldsymbol{i}$ extracted according to $P_I$, is an estimate of $\mathcal{C}(\theta)$, as defined in Equation 2. The sampling average of $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}})$ is therefore itself an estimate of $\mathcal{C}(\theta)$ and can be used for comparing the performance yielded by different configurations.

The above described procedure is iterated and stops either when all configurations but one are discarded, or when some predefined total time $T$ of computation is reached. That is, the procedure would stop before considering the $(k+1)$-th instance if $\sum_{l=1}^{k} t(\underline{\boldsymbol{i}}_{l+1}) \left|\Theta_l\right| > T$.

### 3.2 F-Race

The racing algorithm we propose, F-Race in the following, is based on the Friedman test, a statistical method for hypothesis testing also known as Friedman two-way analysis of variance by ranks (Conover, 1999).

For giving a description of the test, let us assume that F-Race has reached step $k$, and $n = |\Theta_{k-1}|$ configurations are still in the race. The Friedman test assumes that the observed costs are $k$ mutually independent $n$-variate random variables $(\underline{\boldsymbol{c}}^k(\theta_1, \underline{\boldsymbol{i}}_l), \underline{\boldsymbol{c}}^k(\theta_2, \underline{\boldsymbol{i}}_l), \ldots, \underline{\boldsymbol{c}}^k(\theta_n, \underline{\boldsymbol{i}}_l))$ called *blocks* (Dean and Voss, 1999) where each block corresponds to the computational results on instance $\underline{\boldsymbol{i}}_l$ for each configuration in the race at step $k$. Within each block the quantities $\underline{\boldsymbol{c}}^k(\theta, \underline{\boldsymbol{i}}_l)$ are ranked from the smallest to the largest. Average ranks are used in case of ties. For each configuration $\theta_j \in \Theta_{k-1}$, let $R_{lj}$ be the rank of $\theta_j$ within block $l$, and $R_j = \sum_{l=1}^{k} R_{lj}$ the sum of the ranks over all

instances $\underline{\boldsymbol{i}}_l$, with $1 \leq l \leq k$. The Friedman test considers the following statistic (Conover, 1999):

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

Under the null hypothesis that all possible rankings of the candidates within each block are equally likely, $T$ is approximatively $\chi^2$ distributed with $n-1$ degrees of freedom. If the observed $T$ exceeds the $1-\alpha$ quantile of such a distribution, the null is rejected, at the approximate level $\alpha$, in favor of the hypothesis that at least one candidate tends to yield a better performance than at least one other.

If the null is rejected, we are justified in performing pairwise comparisons between individual candidates. Candidates $\theta_j$ and $\theta_h$ are considered different if

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

where $t_{1-\alpha/2}$ is the $1-\alpha/2$ quantile of the Student's $t$ distribution (Conover, 1999).

In F-Race, if at step $k$ the null of the aggregate comparison is not rejected, all candidates in $\Theta_{k-1}$ pass to $\Theta_k$. On the other hand, if the null is rejected, pairwise comparisons are executed between the best candidate and each other one. All candidates that result significatively worse than the best are discarded and will not appear in $\Theta_k$.

### 3.3 Discussion on the Role of Ranking in F-Race

In F-Race, ranking plays an important two-fold role. The first one is connected with the nonparametric nature of a test based on ranking. The main merit of nonparametric analysis is that it does not require to formulate hypotheses on the distribution of the observations. Discussions on the relative pros and cons of the parametric and nonparametric approaches can be found in most textbooks on statistics (Larson, 1982). For an organic presentation of the topic, we refer the reader, for example, to Conover (1999). Here we limit ourselves to mention some widely accepted facts about parametric and nonparametric hypothesis testing: When the hypotheses they formulate are met, parametric tests have a higher power than nonparametric ones and usually require much less computation. Further, when a large amount of data is available the hypotheses for the application of parametric tests tend to be met in virtue of the central limit theorem. Finally, it is well known that the t-test, the classical parametric test that is of interest here, is robust against departure from some of its hypotheses, namely the normality of data: When the hypothesis of normality is not strictly met t-test *gracefully* looses power.

For what concerns the metaheuristics configuration problem, we are in a situation in which these arguments look suspicious. First, since we wish to reduce as soon as possible the number of candidates, we deal with very small samples and it is exactly on these small samples, for which the central limit theorem cannot be advocated, that we wish to have the maximum power. Second, the computational costs are not really relevant since in any case they are negligible compared to the computational cost of executing configurations of the metaheuristic in order to enlarge the available samples. Section 5 shows that the doubts expressed here find some evidential support in our experiments.

A second role played by ranking in F-Race is to implement in a natural way a blocking design (Dean and Voss, 1999). The variation in the observed costs $\boldsymbol{c}$ is due to different sources: Metaheuristics are intrinsically stochastic algorithms, the instances might be very different one from the other, and finally some configurations perform better than others. This last source of variation is the one that is of interest in the configuration problem while the others might be considered as disturbing elements. Blocking is an effective way for normalizing the costs observed on different instances. By focusing only on the ranking of the different configurations within each instance, blocking eliminates the risks that the variation due to the difference among instances washes out the variation due to the difference among configurations.

The work proposed in this paper was openly and largely inspired by some algorithms proposed in the machine learning community (Maron and Moore, 1994; Moore and Lee, 1994) but it is precisely in the adoption of a statistical test based on ranking that it diverges from previously published works. Maron and Moore (1994) proposed Hoeffding Race that adopts a nonparametric approach but does not consider blocking. In a following paper, Moore and Lee (1994) describe BRACE that adopts blocking but discards the nonparametric setting in favor of a Bayesian approach. Other relevant work was proposed by Gratch et al. (1993) and by Chien et al. (1995) who consider blocking in a parametric setting.

This paper, to the best of our knowledge, is the first work in which blocking is considered in a nonparametric setting. Further, in all the above mentioned works blocking was always implemented through multiple pairwise paired comparisons (Hsu, 1996), and only in the more recent one (Chien et al., 1995) correction for multiple tests is considered. F-Race is the first racing algorithm to implement blocking through ranking and to adopt an aggregate test over all candidates, to be performed prior to any pairwise test.

## 4 $\mathcal{MAX}$–$\mathcal{MIN}$-ANT-SYSTEM FOR TSP

In this paper we illustrate F-Race by using as an example the configuration of $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System ($\mathcal{MM}$AS) (Stützle and Hoos, 1997; Stützle and Hoos, 2000), a particular Ant Colony Optimization algorithm (Dorigo and Di Caro, 1999; Dorigo and Stützle, 2002), over a class of instances of the Traveling Salesman Problem (TSP).

### 4.1 A Class of TSP Instances

Given a complete graph $G = (N, A, d)$ with $N$ being the set of $n = |N|$ nodes, $A$ being the set of arcs fully connecting the nodes, and $d$ being the weight function that assigns each arc $(i, j) \in A$ a length $d_{ij}$, the Traveling Salesman Problem (TSP) is the problem of finding a shortest closed tour visiting each node of $G$ once. We assume the TSP is symmetric, that is, we have $d_{ij} = d_{ji}$ for every pair of nodes $i$ and $j$.

The TSP is extensively studied in literature and that serves as a standard benchmark problem (Johnson and McGeoch, 1997; Lawler et al., 1985; Reinelt, 1994). For our study we randomly generate Euclidean TSP instances with a random distribution of city coordinates and a random number of cities. Euclidean TSPs were chosen because such instances are used in a large number of experimental researches on the TSP (Johnson and McGeoch, 1997; Johnson et al., 2001). In our case, city locations are randomly chosen according to a uniform distribution in a square of dimension $10.000 \times 10.000$, and the resulting distances are rounded to the nearest integer. The number of cities in each instance is chosen as an integer randomly sampled according to a uniform distribution in the interval $[300, 500]$. We generated a total number of 400 such instances for our experiments reported in Section 5.

### 4.2 $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System

Ant Colony Optimization (ACO) (Dorigo et al., 1999; Dorigo and Di Caro, 1999; Dorigo and Stützle, 2002) is a population-based approach inspired by the foraging behavior of ants for the solution of hard combinatorial optimization problems. In ACO, artificial ants implement stochastic construction procedures that are biased by pheromone trails and heuristic information on the problem being solved. The solutions obtained by the ants may then be improved by applying some local search routine. ACO algorithms typically follow the high-level procedure given in Figure 2. $\mathcal{MM}$AS (Stützle and Hoos, 1996, 1997; Stützle and Hoos, 2000) is currently one of the best performing ACO algorithms for the TSP.

$\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System constructs tours as follows: Initially, each of the $m$ ants is put on some randomly chosen city. At each construction step, ant $k$ applies a probabilistic action choice rule. In particular, when being at city $i$, ant $k$ chooses to go to a yet unvisited city $j$ at the $t$th iteration with a probability of

**procedure** *Ant Colony Optimization*
  Init pheromones, calculate heuristic
  **while**(termination condition not met) **do**
    $p =$ ConstructSolutions(*pheromones, heuristic*)
    $p =$ LocalSearch$(p)$ *% optional*
    GlobalUpdateTrails$(p)$
  **end**
**end** *Ant Colony Optimization*

Figure 2: Algorithmic skeleton of ACO for static combinatorial optimization problems.

$$p_{ij}^k(t) = \frac{[\tau_{ij}(t)]^\alpha \cdot [\eta_{ij}]^\beta}{\sum_{l \in \mathcal{N}_i^k} [\tau_{il}(t)]^\alpha \cdot [\eta_{il}]^\beta}, \qquad \text{if } j \in \mathcal{N}_i^k; \quad (3)$$

where $\eta_{ij} = 1/d_{ij}$ is an *a priori* available heuristic value, $\alpha$ and $\beta$ are two parameters which determine the relative influence of the pheromone trail and the heuristic information, and $\mathcal{N}_i^k$ is the feasible neighborhood of ant $k$, that is, the set of cities which ant $k$ has not yet visited; if $j \notin \mathcal{N}_i^k$, we have $p_{ij}^k(t) = 0$.

After all ants have constructed a solution, the pheromone trails are updated according to

$$\tau_{ij}(t+1) = (1-\rho) \cdot \tau_{ij}(t) + \Delta\tau_{ij}^{best}, \qquad (4)$$

where $\Delta\tau_{ij}^{best} = 1/L^{best}$ if arc $(i, j) \in T^{best}$ and zero otherwise. Here $T^{best}$ is either the *iteration-best* solution $T^{ib}$, or the *global-best* solution $T^{gb}$ and $L^{best}$ is the corresponding tour length. Experimental results showed that the best performance is obtained by gradually increasing the frequency of choosing $T^{gb}$ for the pheromone trail update (Stützle and Hoos, 2000).

In $\mathcal{MM}$AS, lower and upper limits $\tau_{min}$ and $\tau_{max}$ on the possible pheromone strengths on any arc are imposed to avoid search stagnation. The pheromone trails in $\mathcal{MM}$AS are initialized to their upper pheromone trail limits $\tau_{max}$, leading to an increased exploration of tours at the start of the algorithms.

In our experimental study, we have chosen a number of configurations that differ in particular parameter settings for $\mathcal{MM}$AS. We focused on alternative settings for the main algorithm parameters as they were identified in earlier studies, in particular we considered values of $\alpha \in \{1, 1.25, 1.5, 2\}$, $m \in \{1, 5, 10, 25\}$, $\beta \in \{0, 1, 3, 5\}$, $\rho \in \{0.6, 0.7, 0.8, 0.9\}$. Each possible combination of the parameter settings leads to one particular algorithm config-

uration, leading to a total number of $4 \times 4 \times 4 \times 4 = 256$ configurations. In our experiments each solution is improved by a 2.5-opt local search procedure (Bentley, 1992).

## 5 EXPERIMENTAL RESULTS

In this section we propose a Monte Carlo evaluation of F-Race based on a resampling technique (Good, 2001).

For comparison, we consider two other instances of racing algorithms both based on a paired t-test. They are therefore parametric, and they adopt a blocking design. We refer to them as *tn-Race* and *tb-Race*. The first does not adopt any correction for multiple-tests, while the second adopts the Bonferroni correction and is therefore *not unlike* the method described by Chien et al. (1995).

The goal is to select an *as good as possible* configuration out of the 256 configurations of the $\mathcal{MAX}$–$\mathcal{MIN}$-Ant-System described in Section 4.2.

Each configuration was executed once on each of the 400 instances for $10s$ on a CPU Athlon 1.4GHz with 512 MB of RAM, for a total time of about 12 days to allow in a following phase the application of the resampling analysis. The costs of the best solution found in each of these experiments were stored in a two-dimensional $400 \times 256$ array. In the following, when saying that we *run* configuration $j$ over instance $i$, we will simply mean that we execute the *pseudo-experiment* that consists in reading the value in position $(i, j)$ from the array of the results.

From the 400 instances, we extract 1000 pseudo-samples each of which is obtained by re-ordering randomly the original instances. Each pseudo-sample is used for a *pseudo-trial*, that is, for simulating a run of a racing algorithm: One after the other the instances are considered and, on the basis of the results of pseudo-experiments, configurations are progressively discarded. Each algorithm stops after executing $5 \times 256$ pseudo-experiments.[5] Upon time expiration, the best candidate in the pseudo-trial is selected and it is tested on 10 instances that were ***not*** used during the selection itself. The results obtained on these previously unseen instances are recorded and are used for comparing the three racing methods. To summarize, after 1000 pseudo-trials a vector of $10 \times 1000$ components is obtained for each of F-Race, tn-Race, and tb-Race. It is important to note that the three algorithms face the same pseudo-samples and that the candidates selected in each pseudo-trial by each algorithm are tested on the same unseen instances. The generic $i$-th components of the three $10 \times 1000$ vectors refers therefore to the results obtained by the champions of the three races respectively, where the three races were conducted on the basis of the same pseudo-sample: We are therefore justified in using paired statistical tests when comparing the three races among them.

On the basis of a paired Wilcoxon test we can state that F-Race is significatively better, at a significance level of 5%, than both tn-Race and tb-Race.[6]

Some insight on this result can be obtained from the following observation. By early dropping the less interesting candidates, F-Race is able to perform more experiments on the more promising candidates. On the 1000 pseudo-trials considered, at the moment in which the computation time was up and a decision among the surviving candidate had to be taken, the set of survivors was on average composed by $7.9$ candidates and such survivors had been tested on average on $77.9$ instances. In the case of tn-Race, the average size of the set of survivors upon expiration of computation time was $31.1$, while the number of instances seen by such survivors was on average $18.2$. For tb-Race the numbers are $253.8$ and $5$, respectively. In this sense, F-Race proved to be the bravest of the three, while tb-Race appeared to be extremely conservative and on average it dropped only slightly more than $2$ candidates before the time limit.

On the basis of our Monte Carlo evaluation, some stronger statement can be pronounced on the quality of the results obtained by F-Race. We have shown above that the performance of F-Race was good in a *relative* sense: F-Race produced better results than its competitors. We state now that, in a precise sense to be defined presently, the performance of F-Race was *absolutely* good. We compare F-Race with *Cheat*, a brute-force method that, rather unfairly, uses in each pseudo-trial the same number of instances used by F-Race and on these instances runs all the candidate configurations. In doing so, Cheat allows itself an enormously large amount of computation time. In our experiments, Cheat has performed on average about $19950$ experiments per trial which is equivalent to about $55$ hours of computation against the $3.5$ hour available to F-Race. The selection operated by Cheat is the *optimum* that can be obtained from the fixed set of training instances, and considering only one run of each configuration on each instance. F-Race can be seen as an approximation of Cheat: The set of experiment performed by F-Race is a *proper subset* of the experiments performed by Cheat.

Now, in the statistical analysis of the results obtained by our Monte Carlo experiments, we were not able to reject the null that F-Race and Cheat produce equivalent results. Also in this case, we have worked at the significance level of 5%: neither Wilcoxon test nor t-test were able to show significance.

[5] In such a time, by definition, brute-force would be able to test the 256 candidates on only 5 instances. The $5 \times 256$ pseudo-experiments simulate 3.5 hours of actual computation on the computer used for producing the results proposed here.

[6] The same conclusion can be drawn on the basis of a paired t-test.

## 6 CONCLUSIONS

The paper has given a formal definition of the problem of configuring a metaheuristic and has presented F-Race, an algorithm belonging to the class of racing algorithms proposed in the machine learning community for solving the model selection problem (Maron and Moore, 1994).

In giving a formal definition of the configuration problem, we have stressed the important role played by the probability measure defined on the class of the instances. Without such a concept, it is impossible to give a meaning to the generalization process that is implicit when a configuration is selected on the basis of its performance on a limited set of instances.

F-Race, the algorithm we propose in this paper, is the specialization of the generic class of racing algorithms to the configuration of metaheuristics. The adoption of the Friedman test, which is nonparametric and two-way, matches indeed the specificities of the configuration problem. As shown by the experimental results proposed in Section 5, F-Race obtains better results than its competitors that adopt a parametric approach. This better performance can be indeed explained by the ability of discarding inferior candidates earlier and faster than the competitors. Still, we do not feel like using these results for claiming a general *presumed* superiority of F-Race against its fellow racing algorithms. Rather, we wish to stress the appeal of the racing idea in itself, and we want to interpret our results as an evidence that this idea is extremely promising for configuring metaheuristics and should be further investigated.

**Acknowledgments**

This work was supported by the "Metaheuristics Network", a Research Training Network funded by the Improving Human Potential programme of the CEC, grant HPRN-CT-1999-00106. The information provided is the sole responsibility of the authors and does not reflect the Community's opinion. The Community is not responsible for any use that might be made of data appearing in this publication.

## References

Bentley, J. L. (1992). Fast algorithms for geometric traveling salesman problems. *ORSA Journal on Computing*, 4(4):387–411.

Billingsley, P. (1986). *Probability and Measure*. John Wiley & Sons, New York, NY, USA, second edition.

Chien, S., Gratch, J., and Burl, M. (1995). On the efficient allocation of resources for hypothesis evaluation: A statistical approach. *Pattern Analysis and Machine Intelligence*, 17(7):652–665.

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

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

Dorigo, M. and Di Caro, G. (1999). The Ant Colony Optimization meta-heuristic. In Corne, D., Dorigo, M., and Glover, F., editors, *New Ideas in Optimization*, pages 11–32. McGraw Hill, London, UK.

Dorigo, M., Di Caro, G., and Gambardella, L. M. (1999). Ant algorithms for discrete optimization. *Artificial Life*, 5(2):137–172.

Dorigo, M. and Stützle, T. (2002). The ant colony optimization metaheuristic: Algorithms, applications and advances. In *Metaheuristics Handbook*. Kluwer Academic Publishers. In press.

Good, P. I. (2001). *Resampling Methods*. Birkhauser, Boston, MA, USA, second edition.

Gratch, J., Chien, S., and DeJong, G. (1993). Learning search control knowledge for deep space network scheduling. In *International Conference on Machine Learning*, pages 135–142.

Hsu, J. (1996). *Multiple Comparisons*. Chapman & Hall/CRC, Boca Raton, Fl, USA.

Johnson, D. S. and McGeoch, L. A. (1997). The travelling salesman problem: A case study in local optimization. In Aarts, E. H. L. and Lenstra, J. K., editors, *Local Search in Combinatorial Optimization*, pages 215–310. John Wiley & Sons, Chichester, UK.

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

Larson, H. (1982). *Introduction to Probability Theory and Statistical Inference*. John Wiley & Sons, New York, NY, USA.

Lawler, E. L., Lenstra, J. K., Kan, A. H. G. R., and Shmoys, D. B. (1985). *The Travelling Salesman Problem*. John Wiley & Sons, Chichester, UK.

Maron, O. and Moore, A. W. (1994). Hoeffding races: Accelerating model selection search for classification and function approximation. In Cowan, J. D., Tesauro, G., and Alspector, J., editors, *Advances in Neural Information Processing Systems*, volume 6, pages 59–66. Morgan Kaufmann Publishers, Inc.

Mitchell, T. M. (1997). *Machine Learning*. McGraw-Hill, New York, NY, USA.

Moore, A. W. and Lee, M. S. (1994). Efficient algorithms for minimizing cross validation error. In *International Conference on Machine Learning*, pages 190–198. Morgan Kaufmann Publishers, Inc.

Reinelt, G. (1994). *The Traveling Salesman: Computational Solutions for TSP Applications*, volume 840 of *Lecture Notes in Computer Science*. Springer Verlag, Berlin, Germany.

Stützle, T. and Hoos, H. (1996). Improving the Ant-System: A detailed report on the $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System. Technical Report AIDA–96–12, FG Intellektik, TU Darmstadt, Germany.

Stützle, T. and Hoos, H. H. (1997). The $\mathcal{MAX}$–$\mathcal{MIN}$ Ant System and local search for the traveling salesman problem. In Bäck, T., Michalewicz, Z., and Yao, X., editors, *Proceedings of the 1997 IEEE International Conference on Evolutionary Computation (ICEC'97)*, pages 309–314. IEEE Press, Piscataway, NJ, USA.

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