

10. Parameter tuning

AEA 2026

Algorithm Configuration

The goal: find **high-performing algorithm configurations** that perform well on (a potentially infinite set of) **unseen** instances, not available when deciding on the algorithm's parameters.

The Algorithm Configuration Problem

A 7-tuple $\langle \Theta, I, p_I, p_C, t, C, T \rangle$

- ▶ Θ the possibly infinite set of candidate **configurations**
- ▶ I the possibly infinite set of **instances**
- ▶ p_I a probability measure over I
- ▶ $t : I \rightarrow R$ a function associating to every instance the **computation time** that is allocated to it
- ▶ $c(\theta, i, t(i))$ a random var: the **cost measure** of $\theta \in \Theta$ on $i \in I$ when run for time $t(i)$
- ▶ $C \subset R$ the range of c : the possible values for the cost measure of $\theta \in \Theta$ on $i \in I$
- ▶ p_C a probability measure over C : $p_C(c|\theta, i)$ the probability that c is the cost of running θ on i
- ▶ $C(\theta) = C(\theta|\Theta, I, p_I, p_C, t)$ the **criterion** that needs to be optimized w.r.t θ (measures the desirability of θ)
- ▶ T the total amount of time available for experimenting with the given candidate configurations on the available instances

The Algorithm Configuration Problem

The problem of configuring a parameterized algorithm: find the **configuration** $\bar{\theta}$ s.t. $\bar{\theta} = \operatorname{argmin}_{\theta \in \Theta} C(\theta)$.

$$C(\theta) = E_{I,C}[c] = \int \int c dp_C(c|\theta, i) dp_I(i).$$

p_I and p_C are not explicitly available and the analytical solution of the integrals is not possible.

Estimate the expected cost in a Monte Carlo fashion by running the configuration on a training set of instances.

The **cost of a configuration** θ on an instance i : the **obj. fct. value of the best** solution found in $t(i)$.

The **task** is to tune algorithms for an optimization problem; the goal is to optimize the solution quality reached within a given computation time.

Types of parameters

- ▶ *Categorical* parameters: procedures or discrete choices (type of perturbations, the LS algorithm used in ILS, the neighborhood structure)
- ▶ *Numerical* parameters
 - ▶ *continuous* numerical parameters: the pheromone evaporation rate, the cooling rate
 - ▶ *int*: the strength of a perturbation (#components that change)
- ▶ *Conditional* parameter: the tabu list length (if the categorical parameter "type of local search" indicates that TS is used)

The brute-force approach

The simplest approach for **estimating the expected cost** of a **configuration** θ : run the algorithm using a large # instances.

Estimate for (k) candidate configurations and, once the overall computational budget is consumed, choose the candidate configuration with the lowest estimate.

Problem: poor performing candidate configurations are evaluated with the same amount of computational resources as the good ones.

- ▶ **Population-based** (*CMA-ES*) for **continuous** parameters
- ▶ **Racing algorithms** (*F-Race*), **Iterated Local Search** (*ParamILS*) for **categorical** parameters
 - ▶ ILS is a *stochastic LS*: iteratively performs phases of simple LS (to rapidly reach a local optima), interspersed with perturbation phases (to escape from local optima)
- ▶ **Model-Based Parameter Optimization**: *Sequential Parameter Optimization (SPO)*, *Sequential Model-Based Algorithm Configuration (SMAC)*
 - ▶ The *model-based search paradigm* uses the information gained from the configurations evaluated so far to **build a model of the configuration space**, based on which configurations are chosen to be evaluated in the future

Racing Procedures

- ▶ Inspired from racing algorithms in ML

Racing algorithms

- ▶ evaluate a set of candidate configurations iteratively on a stream of instances
- ▶ when enough statistical evidence is gathered against some candidate configurations, these are eliminated and the race continues only with the surviving ones

First proposed for solving the **model selection problem**.

For **offline configuration of parameterized algorithms**:

- ▶ In the *training phase*, find a configuration in a limited amount of time that optimizes some measure of algorithm performance.
- ▶ The final configuration is then deployed in a *production phase* where the algorithm is used to solve **unseen instances**.

The irace package: <http://iridia.ulb.ac.be/irace/>

F-Race: overview

After each evaluation round of the candidate configurations, F-Race uses the **non-parametric Friedman test** as a family-wise test:

- ▶ It checks whether there is evidence that at least one of the configurations is significantly different from others.
- ▶ If H_0 of no differences is rejected, Friedman **post-tests** are applied to **eliminate** those candidate configurations that are significantly worse than the best one.

Friedman Two-Way Analysis of Variance By Ranks

The Friedman test is a non-parametric test to determine if **there are statistically significant differences between groups**

- ▶ use it instead of ANOVA if you don't know the distribution of the data
- ▶ is an extension of the *Wilcoxon signed-rank test* used when there are multiple groups
- ▶ the hypotheses:
 H_0 : The distributions are the same across repeated measures
 H_A : The distributions across repeated measures are different

Friedman Two-Way Analysis of Variance By Ranks: algorithm comparison

- ▶ Rank the algorithms for a data set; the best performing algorithm: rank 1, the 2nd best: rank 2, etc. In case of ties, average ranks are assigned.
- ▶ The average ranks of algorithms, $R_j = 1/N \sum_i r_i^j$
- ▶ The **Friedman statistic**: $\chi_F^2 = \frac{12N}{k(k+1)} \sum_j (R_j - \frac{k+1}{2})^2$
 χ_F^2 distributed according to a χ^2 distribution with $k - 1$ degrees of freedom, when n and k are big enough ($n > 10$, $k > 5$)

The Racing Approach

Evaluate a set of candidate configurations using a systematic way to allocate the computational resources (*step by step* evaluation):

- ▶ At each step, all the remaining candidate configurations are *evaluated in parallel*.

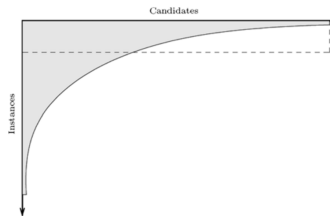
Poor candidate configurations are discarded as soon as sufficient statistical evidence is gathered against them.

- ▶ doesn't require a fixed # steps for each candidate configuration (it determines adaptively based on statistical evidence)

Racing vs. Brute-force

- ▶ **Racing**: as soon as sufficient evidence is gathered that a candidate is suboptimal, **discard it** from further evaluation (as the evaluation proceeds, the approach focuses more on the most promising candidates)
- ▶ **Brute-force**: tests all given candidates on the same #instances

Figure 1: The allocation of configuration evaluations. The shadowed figure - the computation performed by racing, the dashed rectangle - brute-force (the same surface - the same total #experiments).



The racing approach - formal model

- ▶ Randomly generate a sequence of training **instances** i_k , $k = 1, 2, \dots$, from the target class of instances I following the probability model p_I
- ▶ c_k^θ the **cost** of a run of a configuration θ on i_k
- ▶ The evaluation of the candidate configurations is performed incrementally s.t.
at the k^{th} step, the array of **observations for evaluating** θ , $c^k(\theta) = (c_1^\theta, c_2^\theta, \dots, c_k^\theta)$, is obtained by appending c_k^θ to the end of the array $c^{k-1}(\theta)$.

The racing approach - formal model

A *racing* algorithm generates a sequence of nested **sets of candidate configurations** $\Theta_0 \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \dots$

Θ_k the set of the surviving candidate configurations after step k .

- ▶ start from a finite set $\Theta_0 \subseteq \Theta$, obtained by sampling $|\Theta_0|$ candidate configurations from Θ
- ▶ suboptimal configurations are discarded at each step (from set Θ_{k-1} to Θ_k)

The racing approach - formal model

- ▶ At step k , the set of the surviving candidates is Θ_{k-1} ; consider a new instance i_k . Each candidate $\theta \in \Theta_{k-1}$ is tested on i_k and each cost c_k^θ is appended to $c^{k-1}(\theta)$ to form the arrays $c^k(\theta)$, $\forall \theta \in \Theta_{k-1}$.
- ▶ Θ_k : **drop** from Θ_{k-1} the candidate configurations that appear to be suboptimal, based on the statistical test that compares the arrays $c^k(\theta)$, $\forall \theta \in \Theta_{k-1}$.

The above procedure is iterated and stops either when all candidate configurations but one are discarded, a max #instances have been sampled, or the predefined computational budget has been exhausted.

F-Race

F-Race is a racing algorithm based on the *non-parametric Friedman's two-way analysis of variance by ranks*.

Step k : $m = |\Theta_{k-1}|$ candidate configurations are still in the race. The Friedman test assumes that the observed costs are k mutually independent m -variate random variables (blocks). The block b_l corresponds to the computational results obtained on i_l by each surviving configuration at step k).

$$\begin{aligned} b_1 &= (c_1^{\theta_{v_1}}, c_1^{\theta_{v_2}}, \dots, c_1^{\theta_{v_m}}) \\ b_2 &= (c_2^{\theta_{v_1}}, c_2^{\theta_{v_2}}, \dots, c_2^{\theta_{v_m}}) \\ &\vdots \\ b_k &= (c_k^{\theta_{v_1}}, c_k^{\theta_{v_2}}, \dots, c_k^{\theta_{v_m}}) \end{aligned}$$

Within each block, the costs c_l^θ are ranked in nondecreasing order.

F-Race

For each configuration $\theta_{v_j} \in \Theta_{k-1}$, R_{lj} the rank of θ_{v_j} in block b_l ,
 $R_j = \sum_{l=1}^k R_{lj}$ the sum of ranks for θ_{v_j} , over all instances l ,
 $1 \leq l \leq k$. The test statistic used by the Friedman test is:

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

Under H_0 (that all candidates are equivalent), T is approximately χ^2 distributed with $m-1$ degrees of freedom.

If the observed value of T is larger than the $1-\alpha$ quantile of this distribution, H_0 is rejected: at least one candidate configuration gives better performance than at least one of the others.

If H_0 is rejected in this *family-wise test*, do pair-wise comparisons between individual candidates.

F-Race

For F-Race, the Friedman *post hoc* tests: candidates θ_j and θ_h are statistically significantly different if

$$\frac{|R_j - R_h|}{\sqrt{\frac{2k(1 - \frac{\tau}{k(m-1)}) (\sum_{i=1}^k \sum_{j=1}^m R_{ij}^2 - \frac{km(m+1)^2}{4})}{(k-1)(m-1)}}} > t_{1-\alpha/2}$$

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

If F-Race doesn't reject H_0 at step k , all configurations in Θ_{k-1} pass to Θ_k ; if H_0 is rejected, pairwise comparisons are performed between the best candidate configuration (with the lowest expected rank) and each other one. The configurations 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. The Wilcoxon matched-pairs signed-ranks test is adopted.

F-Race algorithm

procedure *F-Race*

input target algorithm A , set of configurations C , set of problem instances I ,
performance metric m ;

parameters integer ni_{min} ;

output set of configurations C^* ;

$C^* := C$; $ni := 0$;

repeat

randomly choose instance i from set I ;

run all configurations of A in C^* on i ;

$ni := ni + 1$;

if $ni \geq ni_{min}$ **then**

perform rank-based Friedman test on results for configurations in C^* on all instances
in I evaluated so far;

if test indicates significant performance differences **then**

$c^* :=$ best configuration in C^* (according to m over instances evaluated so far);

for all $c \in C^* \setminus \{c^*\}$ **do**

perform pairwise Friedman *post hoc* test on c and c^* ;

if test indicates significant performance differences **then**

eliminate c from C^* ;

end if;

end for;

end if;

end if;

until termination condition met;

return C^* ;

end *F-Race*

The Sampling Strategy for F-Race

Full Factorial Design: determine #levels for each parameter;
a configuration: a combination of levels; Θ_0 all combinations.

Drawbacks:

- ▶ requires expertise to select the levels of each parameter
- ▶ the set of candidate configurations grows exponentially with #parameters
 - ▶ d the dimension of the parameter space,
 - ▶ each dimension has l levels
 - ▶ #candidate configurations: l^d

The Sampling Strategy for F-Race

Random Sampling Design: the initial elements are sampled according to a *probability model* p_X defined over the parameter space X .

- ▶ If a priori information is available (the effects of certain parameters or their interactions), define p_X accordingly.
- ▶ Default way of defining p_X : assume a uniform distribution over X .

Advantages:

- ▶ numerical parameters, no a priori definition of the levels
- ▶ arbitrary # of candidate configurations can be sampled (uniformly covering the parameter space)

Iterated F-Race

Iterated F-Race: iterative application of F-Race;
an iteration: **surviving** candidate configurations of the previous iteration **influences the sampling** of new configurations (focus the sampling around the most promising ones).

Iterated F-Race, a **model-based search**:

1. construct candidate solutions based on a probability model
2. evaluate all candidates
3. update the probability model (bias the next sampling towards better candidate solutions)

The steps are iterated until some termination criterion is satisfied.

Iterated F-Race

In each *iteration*: sample a set of candidate configurations and run F-Race on the sampled configurations.

Require: parameter space X , a noisy obj. black-box fct. 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*;

return x^* the best parameter configuration;

Iterated F-Race

- ▶ Effective F-Race: #candidate configurations should be reasonably large
- ▶ Few iterations (a iteration: an execution of F-Race) due to limited # of function evaluations
- ▶ Assume: the total computational budget (# of function evaluations) is given

Iterated F-Race

- ▶ *How many iterations?*
 - ▶ few iterations → sample more configurations at each iteration (more exploration, at the cost of less possibilities of refining the model)
 - ▶ intuitively, #iterations depend on #parameters: for few parameters, expect a less difficult problem to optimize and, hence, fewer iterations required
- ▶ *Which computational budget at each iteration?*
 - ▶ divide equally the computational budget among all iterations
 - ▶ decrease #evaluations available with an increase of iteration counter (increased exploration in the first iterations)

Iterated F-Race

- ▶ *When does F-Race terminate?*

At each iteration l , F-Race terminates if: (i) computational budget for the l^{th} iteration is spent; (ii) a min # of candidate configurations, N_{min} , remains.

F-Race terminates by default if a unique survivor is identified.

- ▶ *Iterated F-Race*: to maintain sufficient **exploration** of the parameter space, keep a # of survivors at each iteration and sample around these survivors the candidate configurations for the next iteration.

To set N_{min} , take into account the # of dimensions in the parameter space X

- ▶ the larger the parameter space, the more survivors should remain (to ensure sufficient exploration)

Iterated F-Race

- ▶ *How should the candidate configurations be generated?*
Randomly sampled in the parameter space according to some probability distribution.
 - ▶ Continuous probability distributions for continuous and quasicontinuous parameters
 - ▶ Discrete probability distributions for categorical and ordinal parameters

Which type of probability distributions? Ex: normal distributions.

- ▶ How should the probability distributions be updated?
How strong the bias towards the surviving configurations of the current iteration should be?
 - ▶ the trade-off between exploration and exploitation

Case study. The configuration of ant colony optimization

Euclidian TSP instances with 750 nodes which are uniformly distributed in a square of side length 10,000. 1,000 instances generated for training, 300 for evaluating the winning configurations (using the DIMACS instance generator).

7 parameters of **Max-Min Ant System**:

- ▶ the relative influence of pheromone trails α , and of heuristic information β
- ▶ the pheromone evaporation rate ρ
- ▶ the # of ants m
- ▶ γ controls the gap between the minimum and maximum pheromone trail value, $\gamma = \tau_{max}/(\tau_{min} \cdot instance_size)$
- ▶ nn the # of nearest neighbors (in the solution construction phase)
- ▶ q_0 the probability of selecting the best neighbor deterministically (in the pseudorandom proportional action choice rule)

Case study

The parameters are discretized using the ranges and # of levels.

| 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 |

The # of candidate parameter settings is 259.200.

Case study

Figure 2: Computational results for configuring MMAS

| algo | 5 seconds | 20 seconds | |
|--------------|--------------|--------------|---------|
| | 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 | 1728 |
| F-Race (RSD) | -0.49 | -0.78 | 1728 |
| I/F-Race | -1.10 | -1.33 | 1728 |
| F-Race (FFD) | +0.90 | +2.38 | 3888 |
| F-Race (RSD) | -0.27 | -0.33 | 3888 |
| I/F-Race | -0.63 | -2.05 | 3888 |

per.dev: the mean percentage deviation of each algorithm from the reference cost ($+x$, $-x$) the solution cost of the algorithm is $x\%$ more (less) than the reference cost)

max.bud: max # of evaluations

- ▶ *Experimental Methods for the Analysis of Optimization Algorithms*. Chapter 13 M. Birattari et al. *F-Race and Iterated F-Race: An Overview*, 2010 https://link.springer.com/chapter/10.1007/978-3-642-02538-9_13