Parameter Control in Evolutionary Algorithms

Jiří Kubalík
Department of Cybernetics, CTU Prague

http://cw.felk.cvut.cz/doku.php/courses/a0m33eoa/start

Algorithm Configuration: Motivation

Often, finding performance-optimizing **parameter configurations of heuristic algorithms requires considerable effort**. In many cases, this task is performed **manually in an ad-hoc way**.

Automating this task is of high practical relevance in several contexts:

- **Development of complex algorithms** - setting the parameters of a heuristic algorithm is a highly labour-intensive task, and indeed can consume a large fraction of overall development time. The use of automated algorithm configuration methods can lead to significant time savings and potentially achieve better results than manual, ad-hoc methods.
- **Empirical studies, evaluations, and comparisons of algorithms** - a central question in comparing heuristic algorithms is whether one algorithm outperforms another because it is fundamentally superior, or because its developers more successfully optimized its parameters. Automatic algorithm configuration methods can mitigate this problem of unfair comparisons and thus facilitate more meaningful comparative studies.
- **Practical use of algorithms** - the ability of complex heuristic algorithms to solve large and hard problem instances often depends critically on the use of suitable parameter settings. End users often have little or no knowledge about the impact of an algorithm's parameter settings on its performance, and thus simply use default settings. Automatic algorithm configuration methods can be used to improve performance in a principled and convenient way.

Parameter Tuning

Typically done by experimenting with different values and selecting the ones that give the best results on the test problems at hand.

Technical drawbacks to parameter tuning:

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- It is heavily based on personal experience and is guided by a mixture of rules of thumb.
- The process of parameter tuning is time consuming, even if parameters are optimised one by one, regardless of their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.
- There are no generally good parameter settings since specific problems require specific setups for satisfactory performance.

Parameter Tuning

Typically done by experimenting with different values and selecting the ones that give the best results on the test problems at hand.

Technical drawbacks to parameter tuning:

- Parameters are not independent, but trying all different combinations systematically is practically impossible.
- It is heavily based on personal experience and is guided by a mixture of rules of thumb.
- The process of parameter tuning is time consuming, even if parameters are optimised one by one, regardless of their interactions.
- For a given problem the selected parameter values are not necessarily optimal, even if the effort made for setting them was significant.
- There are no generally good parameter settings since specific problems require specific setups for satisfactory performance.
- A run of an EA is an intrinsically dynamic, adaptive process. Thus, different values of parameters might be optimal at different stages of the evolutionary process.

  Ex.: Large mutation steps can be good in the early generations, helping the exploration.

  Small mutation steps do better in the late generations fine-tuning the suboptimal solution.

Parameter Control in Evolutionary Algorithms

An EA is a metaheuristic whose components need to be instantiated and properly tuned in order to yield a fully functioning algorithm:

- components – representation, selection and replacement strategy, recombination and mutation operators, ...
- strategy parameters – population size, probability of crossover and mutation, parameter of selection, etc.

The values of these algorithms greatly determine whether the algorithm will find an optimal (or near-optimal) solution, and whether it will find such a solution effectively.

Two major forms of setting the parameter values

- **Parameter tuning** – finding good values for the parameters before the run of the algorithm, and then running the algorithm with these values, which remain fixed during the run.
- **Parameter control** – starts a run with initial parameter values that change during the run.

Parameter Tuning: F-Race

F-Race [Birattari02] – procedure that empirically evaluates a set of candidate configurations by discarding bad ones as soon as statistically sufficient evidence is gathered against them.

- The process starts from a given finite pool of candidate configurations.
- 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 methodology can be applied to repetitive problems – problems where many similar instances appear over time.

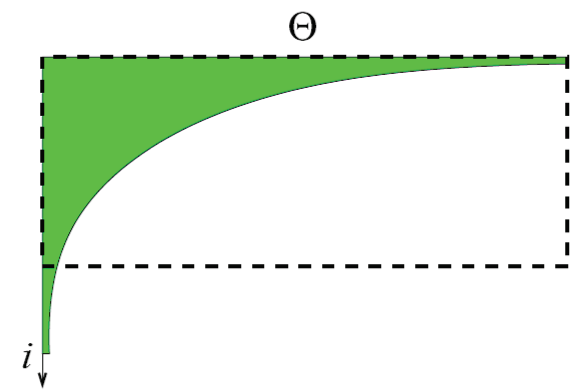
Automatic Algorithm Configuration and Parameter Tuning

- Θ 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 – indicates the probability that the instance i is selected for being solved.
- $t : I \longrightarrow \Re$ is a function associating to every instance the computational time allocated to it.
- $c(\theta, i) = c(\theta, i, t(i))$ is a random variable representing the cost of the best solution found by running configuration θ on instance i for $t(i)$ seconds.
- $C \subset \Re$ is the range of 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: $P_C(c|\theta, i)$ indicates the probability that c is the cost of the best solution found by running configuration θ on instance i for $t(i)$ seconds.
- $C(\theta) = C(\theta|\Theta, I, P_I, P_C, t)$ is the criterion that needs to be optimized with respect to θ.
  - P_I and P_C are unknown,
  - we can only estimate them.

Idea of Racing Algorithms

Brute force approach – estimate the quantities P_C and P_I by means of a sufficiently large number of runs of each candidate on a sufficiently large set of training instances.

- The training set must be defined prior the computation – how large?
- How many runs of each configuration on each instance should be performed?
- The same computational resources are allocated to each configuration – wasting time on poor configs!



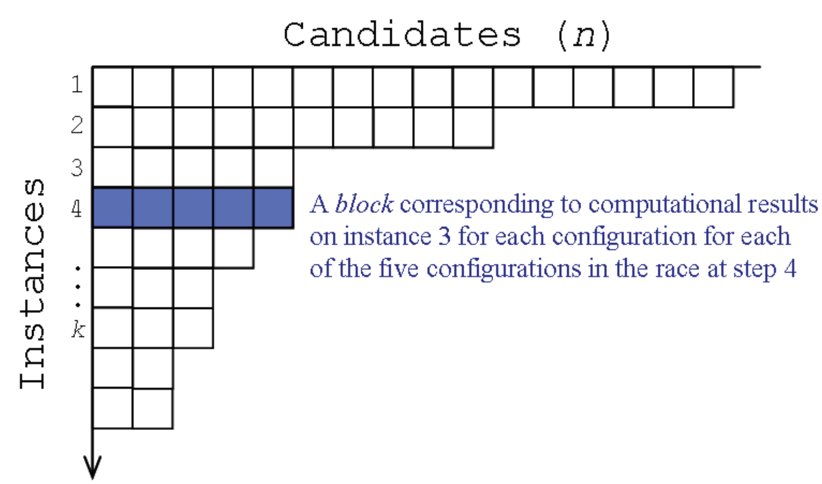
Racing algorithm – provides a better allocation of computational resources among candidate configurations and allows for a clean solution to the problems with fixing the number of instances and the number of runs to be considered.

- Sequentially evaluates candidate configs and discards poor ones as soon as statistically sufficient evidence is gathered against them.
- Elimination of the inferior candidates speeds up the procedure and allows to evaluate the promising ones on more instances.
- As the evaluation proceeds, the race focuses more and more on the promising configurations.

F-Race: Algorithm

- k is the current step of the race process and $n = |\Theta_{k-1}|$ configurations are still in the race.
- $\underline{i}$ is a random sequence of training instances; $\underline{i}_k$ is drawn from I according to P_I, independently for each k.
- $\underline{c}^k(\theta, \underline{i})$ is an array of k terms; $c(\theta, \underline{i}_l)$ is the cost of the best solution found by configuration θ on instance $\underline{i}_l$.

  For a given θ, the array $\underline{c}^k$ of length k can be obtained from $\underline{c}^{k-1}$ by appending the cost concerning the k-th instance in $\underline{i}$.
- A *block* is n-variate random variable $(\underline{c}^k(\theta_1, \underline{i}_l), \underline{c}^k(\theta_2, \underline{i}_l), \ldots, \underline{c}^k(\theta_n, \underline{i}_l))$ that corresponds to the computational results on instance $\underline{i}_l$ for each configuration in the race at step k.



- Null hypothesis – all possible rankings of the candidates within each block are equally likely.

F-Race: Algorithm

The optimization problem is tackled by generating a sequence $\Theta_0 = \Theta \supseteq \Theta_1 \supseteq \Theta_2 \supseteq \ldots$

The step from a set Θ_{k-1} to Θ_k is realized as follows

1. At step k, a new instance $\underline{i}_k$ is considered; each candidate $\theta \in \Theta_{k-1}$ still in the race is executed on $\underline{i}_k$ and each observed cost $c(\theta, \underline{i}_k)$ is appended to its $\underline{c}^{k-1}(\theta, \underline{i})$.
2. An aggregate comparison of the arrays $\underline{c}^k(\theta, \underline{i})$ for all $\theta \in \Theta_{k-1}$ is carried out by a statistical test – non-parametric Friedman 2-way analysis of variance by ranks.

   The null hypothesis being that all possible rankings of the candidates within each block are equally likely.
3. If the null hypothesis is rejected, pairwise comparisons between the best candidate and each other one are carried out by means of the t-test. All candidates that result significantly worse than the best one are discarded.

   Otherwise, all candidates in Θ_{k-1} pass to Θ_k.

F-Race: Algorithm

$\Theta^* = \Theta_0$, $ni = 0$
repeat
    randomly choose instance i from set I; run all configurations on Θ^* on i
    $ni = ni + 1$
    if $(ni \geq ni_{min})$
        perform rank-based Friedman test on results for configurations in Θ^*
        on all instances in I evaluated so far
        if (test indicates significant performance differences)
            $\theta^* =$ best configuration in Θ^* according to the statistical population parameter
            over instances evaluated so far
                for all $\theta \in \Theta^* \setminus \{\theta^*\}$ **do**
                    perform pairwise t-test on θ and θ^*
                    if (test indicates significant performance differences)
                        eliminate θ from Θ^*
                    end if
                end for
        end if
    end if
until (termination condition)
return Θ^*

F-Race: Final Remarks

Good technique, but:

- not suited for applications with large configuration spaces;
- thus, mainly used for configuration problems with few parameters and rather small configuration spaces.