On the Automatic Configuration of Algorithms

Marcelo de Souza*

Abstract. This paper proposes a set of techniques for assisting the automatic configuration of algorithms. First, we present a visual tool to analyze the configuration process. Second, we describe some capping methods to speed up the configuration by early stopping poorly performing executions. Finally, we propose a strategy to automatically identify premature convergence in the configuration.

1 INTRODUCTION

Algorithm configuration is the task of finding one or more parameter settings, also called *configurations*, which optimizes the expected performance of a target algorithm for a particular set of problem instances. A configuration scenario is composed by the parameter space Θ, a set of training instances Π, and a performance metric $c(\theta, \pi)$ of executing the target algorithm under configuration $\theta \in \Theta$ on instance $\pi \in \Pi$. We usually define c as the running time of the execution or the cost of the best found solution, for decision and optimization problems, respectively.

There are different tools that automate the algorithm configuration process. This paper focuses on the irace configurator [1], which implements an iterated racing procedure to explore the parameter space. Given the configuration scenario $\langle \Theta, \Pi, c \rangle$ and a computational budget B, irace iteratively samples a set of configurations and evaluates them using racing. Statistical tests are used to eliminate configurations which perform worse during the racing phase. The surviving (elite) configurations are used by the probabilistic models to sample new configurations for the next iteration. These steps are repeated until the budget B is exhausted. B is a maximum number of evaluations or an execution time limit.

Some difficulties arise when automatically configuring algorithms. First, it is not easy to analyze the configuration process. Mistakes in the design of the configuration scenario (e.g. choosing non-representative training instances or using a too large budget) are hard to detect. Second, configuration is costly, since many candidates must be executed on different instances, and each execution is often time consuming. Finally, the sampling models of irace can converge in an intermediate point of the configuration, then the newly generated candidates are very similar to those already evaluated, loosing diversity. The current convergence detection implemented in irace is not effective in most cases.

We present strategies to deal with the aforementioned problems. First, we propose a visualization tool to analyze the configuration process (Sec. 2). Second, we present several capping methods to avoid wasting time evaluating poor performers (Sec. 3). Finally, we propose a new approach for detecting the premature convergence of the sampling models (Sec. 4).

2 VISUALIZATION TOOL

We propose a tool that produces visual representations of the configuration process performed by irace. Figure 1 presents an example

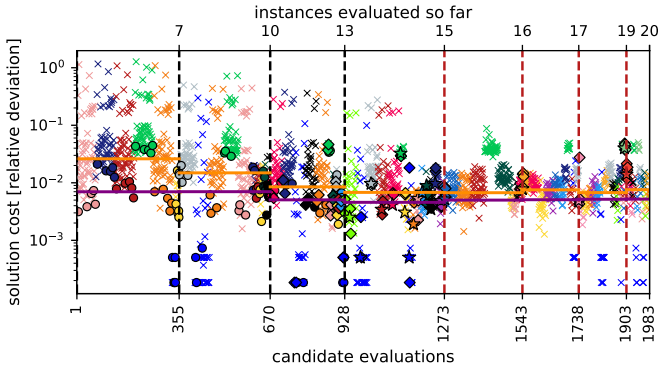

Figure 1. Visualization of the automatic configuration of ACOTSP.

of the main plot for the configuration of ACOTSP, an ant colony optimization algorithm for the traveling salesperson problem. The plot presents all candidate evaluations and indicates the beginning of each iteration (vertical dashed lines) with the corresponding number of candidate evaluations performed so far. Iterations with soft restart are presented in red. Different instances are identified by distinct colors and executions of elite configurations are represented using different markers (○ for elites of the current iteration, ◇ for elites of the last iteration, and ☆ for the best found configuration). For each iteration, we compute the expected performance of both elite and non-elite configuration sets as the median of the results obtained so far. When certain configurations were not evaluated on a given instance, we replace the missing result values by the worst performance of the elite configurations of the corresponding iteration.

The provided visualization gives an overview of the configuration process and allows one to see how the candidates evolve over the iterations. We can also observe how the computational budget is used in each iteration and compare the performance of elite and non-elite configurations. Finally, we can identify possible mistakes in the design of the configuration scenario. For example, Figure 1 shows that the instance represented in green must be hard, since all configurations present almost the same bad performance on it. On the other hand, the instance represented in blue must be easy, since most of the configurations present good performance on it, in comparison to the performances on other instances. Besides that, we observe that from the fourth iteration on, the performance of elite and non-elite configurations does not improve, at the same time that convergence has been detected in all subsequent iterations, indicating a premature convergence of the configuration process.

We provide additional features to allow the user to check more detailed information about each execution by hovering the cursor on the corresponding point. We also allow the user to control the visualization (e.g. by expanding a desired area or moving the plot) and configure the output settings (e.g. by hiding elements or exporting the plot). We also provide a second plot to see the performance of each elite configuration on the test instances, given that the test option was used when running irace. The tool is available at `https://github.com/souzamarcelo/cat`.

* Santa Catarina State University, Federal University of Rio Grande do Sul, Brazil (marcelo.desouza@udesc.br).

Proceedings of the 1st Doctoral Consortium at the European Conference on Artificial Intelligence (DC-ECAI 2020), pages 15–16, Santiago de Compostela, Spain, August 29-30, 2020.

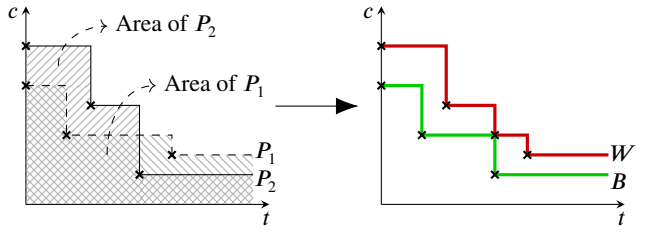

Figure 2. Original (left) and aggregated (right) performance profiles.

3 CAPPING METHODS

During the configuration, a considerable effort is usually spent evaluating poor candidates. For example, observe all the poorly performing executions of the first iterations in Figure 1. We introduce a set of capping methods to speed up the configuration of optimization algorithms. They use previously seen executions to determine a performance envelope, i.e. a minimum required performance, and then use it to evaluate how good new executions are. When the conditions of the envelope are violated, the execution is stopped.

The capping methods behave as follows. The performance envelope is computed before evaluating a new configuration on a given instance. Each previous execution is represented by its performance profile, i.e. the solution cost as a function of the running time, where $P(t) = c$ gives the cost c of the best found solution after running the algorithm for a time t (or any other measure of computational effort). Figure 2 presents two performance profiles (P_1 and P_2) to be aggregated into the performance envelope. We propose two types of envelope. The *profile-based envelope* is given by a performance profile and can be obtained by applying the worst (W) or best (B) aggregation functions to the set of performance profiles. The worst aggregation function selects the pointwise maximum solution cost among the performance profiles for each value of time t, then $W(P_1, \dots, P_n)(t) = \max\{P_1(t), \dots, P_n(t)\}$. Analogously, the best aggregation function selects the pointwise minimum solution cost, then $B(P_1, \dots, P_n)(t) = \min\{P_1(t), \dots, P_n(t)\}$. Figure 2 presents the aggregated performance profiles for both functions on the right side.

The *area-based envelope* considers the area defined by the performance profiles, instead of dealing directly with them. The area of a performance profile P is given by $A_P = \int_0^{t_f} P(t)\,\mathrm{d}t$, where t_f is the cut-off time for finished executions or the current effort of an execution in progress. In this approach, the performance envelope is given by a maximum area available for the next execution. Similarly to the previous strategies, the maximum area is obtained by the worst aggregation function $W(P_1, \dots, P_n) = \max\{A_{P_1}, \dots, A_{P_n}\}$, or the best aggregation function $B(P_1, \dots, P_n) = \min\{A_{P_1}, \dots, A_{P_n}\}$.

We applied the proposed capping methods for the configuration of the following algorithms: ACOTSP, HEACOL (a hybrid evolutionary algorithm for the graph coloring), TSBPP (a tabu search for the bin packing problem), and HHBQP (a hybrid heuristic for the unconstrained binary quadratic programming). We executed irace 20 times for each capping method and computed the mean effort savings in comparison to configuring without capping. We also executed the resulting algorithms 5 times to compute the mean relative deviation from the best known solutions (for HHBQP, we computed the mean absolute deviation). Table 1 presents the results when using no capping (first line), for profile- and area-based approaches (P and A) and both aggregation functions (W and B). The best values of each scenario are in boldface. We observe that all capping methods produce significant effort savings, while the quality of the final configurations does not degrade. As expected, the best aggregation function is more aggressive, producing greater effort savings but worse final configurations in comparison to the worst aggregation function. More detailed results and the source code of these and additional capping strategies can be found at `https://capopt.github.io`.

Table 1. Mean effort savings and mean deviation for each capping method.

Cap.	ACOTSP		HEACOL		TSBPP		HHBQP	
	sav.	r. dev.	sav.	r. dev.	sav.	r. dev.	sav.	a. dev.
-	-	**0.33**	-	**4.14**	-	1.31	-	49.72
PW	59.7	0.37	61.3	4.22	22.6	**1.25**	44.3	65.16
PB	**77.7**	0.52	**74.8**	4.48	38.1	1.27	**74.9**	58.38
AW	26.8	0.35	27.2	4.18	12.4	1.28	17.3	**46.97**
AB	52.7	0.38	47.0	4.18	**41.4**	1.35	65.9	68.56

4 CONVERGENCE DETECTION

The current method of convergence detection implemented in irace is based on a distance measure between elites and each generated configuration. If the distance is less than a threshold, the associated sampling models are restarted. We present an alternative approach to detect convergence, which compares the performance of the configurations of the current and previous iterations, instead of analyzing the configurations individually. If the difference in the observed quality (i.e. the mean running time or the mean relative deviation from a lower bound) of both iterations is smaller than a threshold, the sampling models are restarted.

We analyzed previous runs of irace and applied our approach to the available data. We observe that the proposed method is able to detect the convergence almost whenever the current method detects. Besides that, the proposed approach is able to detect convergence in some cases when no improvement is reached, but the current method does not detect any convergence. We plan to extend this method to consider not only the observed performances, but the evolution of the probability distributions used to sample new configurations.

5 CONCLUDING REMARKS

The preliminary experiments indicate that the proposed methods are effective for improving the automatic configuration of algorithms. The visualization provides useful information and can be helpful to understand the configuration process and to identify possible mistakes in the configuration scenario. The capping methods led to a significant reduction in the configuration time for different scenarios. This allows us to scale the automatic methods to challenging configuration tasks. Finally, an effective method to detect convergence can help to diversify the parameter space exploration when appropriate, and then make a better use of the available computational budget.

For the next steps, we plan to extend the experiments. We will use the visualizations to analyze an extensive set of configuration scenarios, identifying common made mistakes and how to visualize them. We also want to apply the capping methods using the total configuration time as budget and allowing irace to use the saved effort to further explore the parameter space.

REFERENCES

[1] Manuel López-Ibáñez, Jérémie Dubois-Lacoste, Leslie Pérez Cáceres, Thomas Stützle, and Mauro Birattari, 'The irace package: Iterated racing for automatic algorithm configuration', *Operations Research Perspectives*, **3**, 43–58, (2016).

Proceedings of the 1st Doctoral Consortium at the European Conference on Artificial Intelligence (DC-ECAI 2020), pages 15–16, Santiago de Compostela, Spain, August 29-30, 2020.