How to Speed-Up the Automated Configuration of Optimization Algorithms

Marcelo de Souza
Santa Catarina State University, Brazil
Federal University of Rio Grande do Sul, Brazil
marcelo.desouza@udesc.br

Abstract

This paper presents a set of capping methods to speed-up the automated configuration of optimization algorithms. These methods use known previous executions to compute a performance envelope, which is used to evaluate new executions and early stop those with unsatisfactory performance. Preliminary experiments on six scenarios show that the capping methods save up to 78% of the configuration effort, while finding configurations of the same quality.

Introduction

The algorithm configuration task is to find one or more parameter configurations that optimize the algorithm's expected performance on a given set of problem instances. The performance of executing the algorithm on a particular instance is given by a predefined performance metric, which is usually the running time for decision algorithms, or the cost of the best found solution for optimization algorithms.

Automated algorithm configuration techniques (e.g. Hutter et al. (2009); Hutter, Hoos, and Leyton-Brown (2011); Ansótegui, Sellmann, and Tierney (2009)) free researchers from the time-consuming, tedious and often biased task of manually testing several parameter configurations on different instances to find the best ones. A widely used algorithm configurator is irace (López-Ibáñez et al. 2016), which implements an iterated method based on the Friedman-Race. It iteratively samples a set of configurations and evaluates them on a subset of the instances using a racing procedure. During the racing, configurations that perform statistically worse than others are eliminated. The surviving (elite) configurations are used to update the probabilistic models and guide the sampling of new configurations in subsequent iterations.

The automated configuration process is costly, since irace executes many parameter configurations on several instances. Capping methods can be used to reduce the configuration time of decision algorithms (Pérez Cáceres et al. 2017; Hutter et al. 2009). These methods determine a bound on the running time based on the best-performing configuration found so far, and then discard configurations whose execution reaches this running time bound. Unfortunately, these methods cannot be directly applied in the configuration of optimization algorithms. Therefore, we propose a set of capping methods for optimization scenarios, which compute a performance envelope based on previous executions, then use it to evaluate new executions and early stop poor performers. The next sections detail the proposed methods and present experimental results.

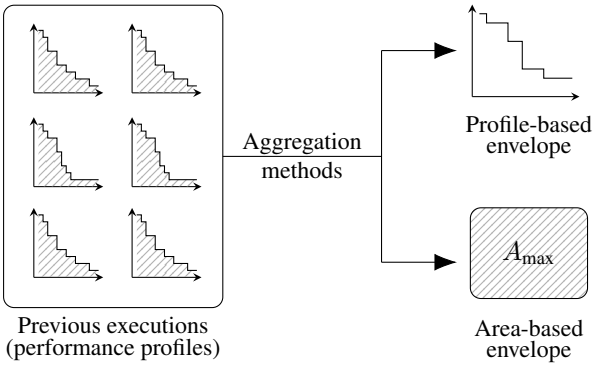

Figure 1: Overview of the capping methods.

Capping Methods

The capping methods behave as follows. An execution is represented by its performance profile, i.e. a function $P(t) = c$ that gives the cost c of the best solution found after running the algorithm for a time t (or any other measure of computational effort). Before starting each execution, the performance profiles of previous executions on the instance at hand are aggregated into the performance envelope. First, we aggregate different replications of a same configuration, and then we aggregate the resulting performance profiles of different configurations into the performance envelope. The proposed capping methods differs from each other by their inner aggregation strategies.

Figure 1 summarizes the idea behind the profile- and area-based performance envelopes. In the profile-based approach, the envelope is a performance profile that determines the maximum allowed solution cost (considering a minimization problem) for each value of time. On the other hand, in the area-based approach the area below the performance pro-

Capping method	ACOTSP		HEACOL		TSBPP		HHBQP		LKH		SCIP	
	r. e.	r. d.	r. e.	r. d.	r. e.	r. d.	r. e.	a. d.	r. e.	r. d.	r. e.	r. d.
No capping	100.0	**0.33**	100.0	**4.14**	100.0	1.31	100.0	49.72	100.0	**0.04**	100.0	**0.04**
PEWW	**40.3**	0.37	**38.7**	4.22	87.4	**1.25**	**55.7**	65.16	**37.8**	**0.04**	69.0	**0.05**
PEBB	**22.3**	0.52	**25.2**	4.48	**61.9**	**1.27**	**25.1**	58.38	**24.3**	0.08	**21.7**	0.11
PD.4	63.7	0.38	61.9	4.27	88.3	1.28	74.6	**44.71**	62.1	**0.04**	**63.9**	0.16
AEWW	73.2	**0.35**	72.8	**4.18**	87.6	1.28	82.7	**46.97**	52.5	**0.04**	94.0	**0.04**
AEBB	**47.3**	0.38	**53.0**	**4.18**	**58.6**	1.35	**34.1**	68.56	**33.8**	0.06	**62.9**	0.08
AD.4	77.9	**0.35**	81.8	**4.16**	**71.5**	**1.26**	72.9	**37.48**	72.4	**0.04**	80.3	0.06

Table 1: Average relative effort and average solution cost deviation for each capping method.

file of each previous execution is computed, and these area values are aggregated into a maximum allowed area $A_{\max}$ for new executions. Given a set of performance profiles to be aggregated, the capping methods use the following aggregation functions.

Worst (W). Selects the worst solution cost for each value of time (profile-based); or selects the worst area value (area-based).

Best (B). Analogous to the worst function, selects the best solution cost for each value of time (profile-based) or the best area value (area-based).

The capping methods that use W and B aggregation functions filter the previous executions and use only executions of elite configurations, since they are the best configurations found so far and should present good performance. Therefore, they are named elitist capping methods. A second approach is called adaptive capping. It considers all previous executions on the instance at hand, and computes the performance envelope based on an aggressiveness goal parameter a_g. In the beginning of the configuration process, the method's aggressiveness a is set to a_g. Before starting each execution, the adaptive capping method computes the exact envelope (performance profile or maximum allowed area) that would cap the $\lceil (1-a)k \rceil$ worst executions from all k previous executions. When an iteration finishes, the value of a is adapted for the next iteration. If the method capped less than $a_g - \varepsilon$, a is increased. If there was more capping than $a_g + \varepsilon$, a is decreased. The update procedure determines the value of a that would be capped the desired amount of executions in the previous iteration.

Experimental Results

The capping methods are implemented by the *capopt* package. The source code, instructions of use, examples and additional details can be found in De Souza, Ritt, and López-Ibáñez (2020). We applied the capping methods in the configuration of the following algorithms: ACOTSP (Dorigo and Stützle 2004), an ant colony optimization framework for the traveling salesperson problem; HEACOL (Galinier and Hao 1999; Lewis 2016), a hybrid evolutionary algorithm for the graph coloring; TSBPP (Lodi, Martello, and Vigo 1999, 2004), a tabu search for the bin packing problem; HHBQP (De Souza and Ritt 2018), a hybrid heuristic for the unconstrained binary quadratic programming; LKH (Lin and Kernighan 1973; Helsgaun 2000, 2009, 2018), the Lin-Kernighan-Helsgaun algorithm for the traveling salesperson problem; and SCIP (Achterberg 2009), an open-source exact solver for mixed integer programming applied for solving the combinatorial auction winner determination problem. 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 configurations 5 times to compute the mean relative deviation from the best known solutions (for HHBQP, we computed the mean absolute deviation).

Table 1 shows the relative effort (columns "r. e.") and the deviation from the best known solutions (columns "r. d." and "a. d.") for each capping method on each configuration scenario. The method description presents the type of envelope (P for profile-based and A for area-based), followed by the elitist (E) or adaptive (D) strategies. For elitist methods, it also shows the aggregation functions (W or B). The first one is the function used for aggregating replications of the same configuration, and the second one is the function used for aggregating executions of different configurations. For the adaptive methods, it also shows the aggressiveness goal (in this case, we tested the adaptive methods with $a_g = 0.4$ and $\varepsilon = 0.05$ only). The three best values are presented in bold.

We observe that all capping methods reduce the configuration effort. The reduction ranges from 6% (method AEWW on SCIP) to about 78% (method PEBB on ACOTSP and SCIP) of the effort required when configuring without capping. At the same time, the configurations found when using capping are competitive in comparison to those obtained without capping. In some cases (see the results on TSBPP and HHBQP), the use of capping led to better final configurations. As expected, the more aggressive methods (PEBB and AEBB) save more effort, but produces worse configurations. Finally, we can recommend methods PEWW and AD.4 for use, since they present satisfactory savings (averages of 45% and 23%, respectively) and always produce good configurations.

As future directions, we plan to extend our experiments and analyze in detail the behavior of each capping method, as well as apply their fundamental ideas on the configuration of decision algorithms.

References

Achterberg, T. 2009. SCIP: Solving constraint integer programs. *Mathematical Programming Computation* 1(1): 1–41.

Ansótegui, C.; Sellmann, M.; and Tierney, K. 2009. A Gender-Based Genetic Algorithm for the Automatic Configuration of Algorithms. In Gent, I. P., ed., *Principles and Practice of Constraint Programming, CP 2009*, volume 5732 of *Lecture Notes in Computer Science*, 142–157. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-04244-7_14.

De Souza, M.; and Ritt, M. 2018. Automatic Grammar-Based Design of Heuristic Algorithms for Unconstrained Binary Quadratic Programming. In *Evolutionary Computation in Combinatorial Optimization*, 67–84. Springer International Publishing. doi:10.1007/978-3-319-77449-7_5.

De Souza, M.; Ritt, M.; and López-Ibáñez, M. 2020. CAPOPT: Capping Methods for the Automatic Configuration of Optimization Algorithms. https://github.com/souzamarcelo/capopt. Date: Apr, 2021.

Dorigo, M.; and Stützle, T. 2004. *Ant Colony Optimization*. Cambridge, MA: MIT Press.

Galinier, P.; and Hao, J.-K. 1999. Hybrid evolutionary algorithms for graph coloring. *Journal of Combinatorial Optimization* 3(4): 379–397. doi:10.1023/A:1009823419804.

Helsgaun, K. 2000. An Effective Implementation of the Lin-Kernighan Traveling Salesman Heuristic. *European Journal of Operational Research* 126: 106–130.

Helsgaun, K. 2009. General *k*-opt Submoves for the Lin-Kernighan TSP Heuristic. *Mathematical Programming Computation* 1(2–3): 119–163.

Helsgaun, K. 2018. Efficient Recombination in the Lin-Kernighan-Helsgaun Traveling Salesman Heuristic. In Auger, A.; Fonseca, C. M.; Lourenço, N.; Machado, P.; Paquete, L.; and Whitley, D., eds., *Parallel Problem Solving from Nature - PPSN XV*, volume 11101 of *Lecture Notes in Computer Science*, 95–107. Springer, Cham. doi:10.1007/978-3-319-99253-2_8.

Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2011. Sequential Model-Based Optimization for General Algorithm Configuration. In Coello Coello, C. A., ed., *Learning and Intelligent Optimization, 5th International Conference, LION 5*, volume 6683 of *Lecture Notes in Computer Science*, 507–523. Springer, Heidelberg, Germany. doi:10.1007/978-3-642-25566-3_40.

Hutter, F.; Hoos, H. H.; Leyton-Brown, K.; and Stützle, T. 2009. ParamILS: An Automatic Algorithm Configuration Framework. *Journal of Artificial Intelligence Research* 36: 267–306. doi:10.1613/jair.2861.

Lewis, R. M. R. 2016. *A Guide to Graph Colouring: Algorithms and Applications*. Springer, Cham. doi:10.1007/978-3-319-25730-3.

Lin, S.; and Kernighan, B. W. 1973. An Effective Heuristic Algorithm for the Traveling Salesman Problem. *Operations Research* 21(2): 498–516.

Lodi, A.; Martello, S.; and Vigo, D. 1999. Heuristic and metaheuristic approaches for a class of two-dimensional bin packing problems. *INFORMS Journal on Computing* 11(4): 345–357. doi:10.1287/ijoc.11.4.345.

Lodi, A.; Martello, S.; and Vigo, D. 2004. TSpack: a unified tabu search code for multi-dimensional bin packing problems. *Annals of Operations Research* 131(1-4): 203–213. doi:10.1023/B:ANOR.0000039519.03572.08.

López-Ibáñez, M.; Dubois-Lacoste, J.; Pérez Cáceres, L.; Stützle, T.; and Birattari, M. 2016. The irace package: Iterated Racing for Automatic Algorithm Configuration. *Operations Research Perspectives* 3: 43–58.

Pérez Cáceres, L.; López-Ibáñez, M.; Hoos, H. H.; and Stützle, T. 2017. An experimental study of adaptive capping in irace. In Battiti, R.; Kvasov, D. E.; and Sergeyev, Y. D., eds., *Learning and Intelligent Optimization, 11th International Conference, LION 11*, volume 10556 of *Lecture Notes in Computer Science*, 235–250. Cham, Switzerland: Springer.