Greybox Automated Algorithm Configuration

Marie Anastacio
Leiden Institute of Advanced Computer Science, Leiden University, The Netherlands
m.i.a.anastacio@liacs.leidenuniv.nl

Abstract

The performance of state-of-the-art algorithms is highly dependent on their parameter values, and choosing the right configuration can make the difference between solving a problem in a few minutes or hours. Automated algorithm configurators have shown their efficiency on a wide range of applications. However, they still encounter limitations when confronted to a large number of parameters to tune or long algorithm running time.

We believe that there is untapped knowledge that can be gathered from the elements of the configuration problem, such as the default value in the configuration space, the source code of the algorithm, and the distribution of the problem instances at hand. We aim at utilising this knowledge to improve algorithm configurators.

1 Introduction

The internal behaviour of many algorithms depends on their parameter settings. Choosing the right parameter values can make the difference between poor and state-of-the-art performance [Hutter *et al.*, 2009]. To select the right values of those parameters and to optimise the algorithm performance on a specific set of problem instances is known as algorithm configuration or hyper-parameter tuning. General-purpose automatic algorithm configurators, such as irace [Birattari *et al.*, 2010; López-Ibáñez *et al.*, 2016], GGA++ [Ansótegui *et al.*, 2015], SMAC [Hutter *et al.*, 2011], and more recently GPS [Pushak and Hoos, 2020], are widely applied in domains such as Machine Learning (see, *e.g.*, [Kotthoff *et al.*, 2017]) and NP-hard problem solving (see, *e.g.*, [Hutter *et al.*, 2017; Hutter *et al.*, 2010]).

The algorithm configuration problem is defined as follows (see, *e.g.*, [Hoos, 2012]): Given a target algorithm A; a configuration space C containing all valid combinations of parameter values of A; a set of problem instances I; and a performance metric m that measures the performance of a configuration in C of target algorithm A on an instance of I; find $c^* \in C$ that optimises the performance of A on instance set I, according to metric m. Each parameter p_j have domain D_j of possible values that can be of different types: categorical, ordinal or numerical. Categorical parameters have an unordered finite set of possible values and are often used to select between several heuristic components or mechanisms. Ordinal parameters have an ordered finite set of possible values. Numerical parameters are real- or integer-valued and often control aspects of an algorithm or heuristic. Parameters can also conditionally depend on each other, meaning that they are active only when another parameter takes a specific value; as a simple example, consider a Boolean parameter that activates a mechanism, whose behaviour is adjusted using a numerical parameter.

Algorithm configurators need to handle two main challenges. First, the time required to evaluate the performance of each configuration can vary from seconds to hours, and it is critical to reduce the number of evaluation for the algorithm. To do so, a possible approach is to rely on an empirical performance model that predicts how well a configuration will perform without running the target algorithm (SMAC and GGA++). Second, the possibility to configure their algorithm afterwards leads programmers to expose more parameters. To handle this large and complex search space, a critical mechanism is the sampling of new configurations to be evaluated. They can be generated based on known good configurations (irace and GGA) or improved according to predictions obtained from a performance model (SMAC and GGA++).

Despite sophisticated approaches, configurators still encounter limitations when the search space grows or the running time is too long. We aim to improve them further and tackle their limitations by exploiting information unused by current configurators.

2 Contribution

To improve current configuration methods, we want to use information already available to them but not yet used or only partially.

2.1 Approach

We look more closely at the target algorithm A, the configuration space C and the instance set I to extract knowledge that can be used by the configurator. In particular, we note the following: current configuration approaches are considering A as a black-box algorithm while in many cases its source code is accessible; the search space C is considered as if all values had equal probabilities to be good and all parameters had the same importance for performance, while this is known to

be untrue [Fawcett and Hoos, 2016]; and problem instances from I are considered equally when deciding on which instance to evaluate the target-algorithm, while the time needed to solve them can be predicted thanks to a set a characteristic features to avoid wasteful evaluations.

2.2 Results and Ongoing Work

In most cases, C contains a default value for each parameter, which is provided by the developers or based on prior researches, and thus reveals a deeper understanding of the mechanism behind the parameter or extensive testing on relevant problem instances. We exploited this prior knowledge from experts by focusing the sampling around those default values with a truncated normal distribution. We achieved a significant performance improvement over a set of well-studied configuration scenarios [Anastacio and Hoos, 2020]. We think that the default values are only a small part of the expert knowledge that can be gathered from the users. Hence we are exploring more sophisticated ways of collecting this information and integrating it into the configuration process.

Extracting the information contained in the source-code of A can be done through metrics and features. Those algorithm features further improve empirical performance models such as the ones contained in SMAC or GGA++. A first paper exploring this avenue for Algorithm Selection is currently under review in Artificial Intelligence Journal. Transforming our algorithm features such that they describe a specific set of parameter values is not trivial, but software engineering methods, such as partial program evaluation or code slicing, offer promising possibilities.

The instances contained in I have various running time and often come with instance features that are used for example by SMAC's model to predict how long is needed to solve them. However, when current configurators choose on which instances to evaluate A, they sample them uniformly. We are investigating methods to reduce the time spent on running the algorithm by prioritizing instances that have an expected low running time and high variance between configurations, such that we can easily discriminate between configurations without spending too much time on algorithm evaluations.

3 Conclusion

This research aims at improving current algorithm configurators and is based on the key idea that the different elements given to current configurators contain knowledge that so far remains untapped. Our work on using the knowledge contained in the default-value to bias the search toward a promising area of the configuration space has already led to significant improvements [Anastacio and Hoos, 2020], and we are now exploring possibilities to use source-code features for the algorithm and chose more wisely the instances on which to run the target algorithm.

Based on this research, we will develop a new configurator that leverages this new extracted knowledge. Moreover, existing configurators could implement our methods to improve their own performance.

References

[Anastacio and Hoos, 2020] Marie Anastacio and Holger Hoos. Model-based algorithm configuration with default-guided probabilistic sampling. In *Proceedings of the 16th International Conference on Parallel Problem Solving from Nature (PPSN-20)*, 2020.

[Ansótegui *et al.*, 2015] C. Ansótegui, Y. Malitsky, H. Samulowitz, M. Sellmann, and K. Tierney. Model-based genetic algorithms for algorithm configuration. In *Proc. IJCAI 2015*, pages 733–739, 2015.

[Birattari *et al.*, 2010] M. Birattari, Z. Yuan, P. Balaprakash, and T. Stützle. F-Race and iterated F-Race: An overview. In *Experimental Methods for the Analysis of Optimization Algorithms*, pages 311–336. 2010.

[Fawcett and Hoos, 2016] C. Fawcett and H. H. Hoos. Analysing differences between algorithm configurations through ablation. *Journal of Heuristics*, 22(4):431–458, 2016.

[Hoos, 2012] H. H. Hoos. Automated algorithm configuration and parameter tuning. In Y. Hamadi, E. Monfroy, and F. Saubion, editors, *Autonomous Search*, pages 37–71. Springer, 2012.

[Hutter *et al.*, 2009] F. Hutter, H. H. Hoos, K. Leyton-Brown, and T. Stützle. ParamILS: An automatic algorithm configuration framework. *J. Artif. Intell. Res.*, 36:267–306, 2009.

[Hutter *et al.*, 2010] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Automated configuration of mixed integer programming solvers. In *Proc. CPAIOR 2010*, pages 186–202, 2010.

[Hutter *et al.*, 2011] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Sequential model-based optimization for general algorithm configuration. In *Proc. LION 5*, pages 507–523, 2011.

[Hutter *et al.*, 2017] F. Hutter, M. Lindauer, A. Balint, S. Bayless, H. H. Hoos, and K. Leyton-Brown. The configurable SAT solver challenge (CSSC). *Artif. Intell.*, 243:1–25, 2017.

[Kotthoff *et al.*, 2017] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown. Auto-WEKA 2.0: Automatic model selection and hyperparameter optimization in WEKA. *Journal of Machine Learning Research*, 18:25:1–25:5, 2017.

[López-Ibáñez *et al.*, 2016] M. López-Ibáñez, J. Dubois-Lacoste, L. Pérez Cáceres, T. Stützle, and M. Birattari. The irace package: Iterated racing for automatic algorithm configuration. *Operations Research Perspectives*, 3:43–58, 2016.

[Pushak and Hoos, 2020] Yasha Pushak and Holger H. Hoos. Golden parameter search: exploiting structure to quickly configure parameters in parallel. In Carlos Artemio Coello Coello, editor, *GECCO '20: Genetic and Evolutionary Computation Conference*, pages 245–253. ACM, 2020.