Universiteit Leiden
The Netherlands

Sampling strategies in automated algorithm configuration

Anastacio, M.I.A.

7

Conclusions

In this thesis, we conducted a detailed investigation of the sampling strategies in the context of general-purpose algorithm configurators for running time optimisation tasks (as defined in Chapter 2).

7.1 The work achieved so far

Before developing our own methods, it was essential to gain a deeper understanding of the current state of the art. We compiled a set of widely used AAC scenarios for running time optimisation and used them to evaluate state-of-the-art algorithm configurators. Current configuration approaches are based on various search algorithms such as racing (Birattari et al., 2010), genetic algorithm (Ansótegui et al., 2009), Bayesian optimisation (Hutter et al., 2011b) and golden search (Pushak and Hoos, 2020). Each of those methods demonstrated its own strengths in a set of scenarios when introduced to the research community. However, we are not aware of any extensive comparison between them for running time scenarios. In Chapters 2 and 3, we addressed this gap (through RQ1, RQ2 and RQ3). We ran a carefully selected set of configurators – ROAR, SMAC2 and 3, irace, GGA++, GPS and paramILS – on a set of 20 scenarios spanning four different NP-hard problems, eight target algorithms and ten datasets of problem instances. Additionally, in Chapter 2, we identified 16 general characteristics of the configuration scenarios, highlighting in the process that the commonly used benchmark AClib lacks variety in some of those characteristics. Specifically, it is

limited to one default configuration, whereas some configurators can handle multiple of them, and it contains mostly non-deterministic algorithms and randomly generated datasets. In Chapter 3, we analysed the performance of the configurators in general and with regard to the scenario characteristics specifically. This allowed us to draw high-level conclusions regarding their strength and weaknesses. Specifically, we found that, despite learning on a discretised search space, paramILS is the best performing configurator on almost half of the considered scenarios. We further described the average ranking of the configurators, showing that SMAC2 has the best average ranking, statistically tied with paramILS, SMAC3 and ROAR. We exposed good complementarity between the top performing configurators using Shapley values and attempted to find which characteristics of a scenario impact the performance of the configurators the most. However, it remains challenging to understand what makes some scenarios more difficult than others or which configurator should be used for specific scenarios.

We observed that in any configurator, there are two key elements to sample: new target algorithm configurations to evaluate and instances on which to run them. In the subsequent chapters of this thesis, we explored whether there are simple, yet effective ways to refine the sampling strategies at those points.

First, we examined the sampling strategies for new configurations. Each configurator has its own approach to generating new configurations based on insights gained during the configuration run. However, most overlook a prior typically given with an algorithm: the default configuration. This configuration is typically set based on the algorithm designers' insight regarding their method, coupled with prior experiments results. In Chapter 4, we hypothesised that the default contains meaningful information. To test this, we first explored the impact and usage of these default parameter values in current configurators (RQ4). We compared the performance of configurators depending on the given default configuration and showed that it has a large impact on irace but a limited one on SMAC. We then proposed approaches to either reduce the search space to keep only configurations close to the given default or sample following a truncated normal distribution centred on the default value (RQ5). We tested these approaches in a state-of-the-art configurator on 20 configuration scenarios, showing that they improve performance for a majority of the studied scenarios.

Second, we examined how configurators decide on which instances to run the next configuration during its evaluation. Most configurators simply select an instance uniformly at random. We hypothesised that two algorithms or two configurations thereof could be accurately and efficiently compared on a smaller set of well-chosen instances. Chapter 5 provided evidence in support of that intuition in the context of the com-

parison of algorithms (RQ6). We evaluated 5 selection methods including random sampling, and two statistical tests to stop the comparison of two algorithms. We showed that, by selecting instances with either a high variance in their running time or a high discrimination power, statistical tests can decide which algorithm is best after collecting algorithm performance data for less than 15% of the CPU time required for a complete comparison. Then, Chapter 6 translated the previously developed methods for the comparison of two configurations of a single algorithm to study how they could help in the context of automated algorithm configuration. First, we designed experiments to evaluate how efficiently two configurations of the same algorithm can be compared and how quickly we can abandon bad configurations using the two best performing methods for instance selection in the context of algorithm comparison (RQ7) and two methods inspired from the active learning literature. We tested the selection methods on randomly generated configurations, and obtained significant speed-ups, from 5 to 3000 folds, which encouraged us towards the next step. We implemented the best-performing selection methods – based on variance and discrimination power – within the state-of-the-art model-based configurator SMAC and evaluated their impact on the configuration process (RQ8). Although there is no clear guideline allowing us to determine which selection method to apply when, we found that including these mechanisms into algorithm configurators could unlock previously unseen performance. In particular for EAX on `rue-1000-3000`, one of the five studied scenarios, our modified version of SMAC3 nearly doubled the improvement of vanilla SMAC3 and performed better than the best performing method from Chapter 3.

7.2 Future work

The continuation of this line of research could involve implementing our methods in other configurators such as paramILS and SMAC2 – the top performing according to Chapter 3 – and evaluating them according to the framework used in Chapter 3 to assess in more detail which kinds of scenarios benefit from using them and which are not. For the latter, identifying the reasons for such a discrepancy could lead to a better understanding of what makes a scenario challenging and how to handle those challenging scenarios more effectively. For example, the instance selection methods from Chapters 5 and 6 would likely have a larger impact on datasets containing problem instances with large variations in running time, whilst the sampling strategies from Chapter 4 would benefit scenarios with a large search space and well-thought default values based on long-standing research.

The work done in Chapter 3 could also be extended to more types of algorithm configuration problems, such as multi-objective configuration or configuration for other performance metrics besides running time optimisation (see *e.g.* Blot et al., 2016). It also opens the door to the development of a configurator selector, which would predict for a specific scenario which configurator should be applied based on its characteristics. However, for this last point, we conducted preliminary experiments to learn a multi-class random forest model on the data collected in Chapter 3 and did not reach any promising results. We might need more expressive characteristics than those listed in Chapter 2 or design scenarios with more variations in those characteristics to allow the model to learn more from them.

The work conducted in Chapter 5 opens up ways to make the development and evaluation of solvers for hard problems more sustainable. For example, instance selection methods would allow a developer to receive quick feedback on new ideas without running their solver on too many problem instances. These methods could also reduce the amount of computation needed to declare the winners of a competition by only running the algorithms on more instances if there is a probability for them to land in the top. Since the less performing algorithms are also the ones consuming the most resources, this would have a large impact on the required computing time, *e.g.* in 2020 the top 10 algorithms of the SAT competition ran for about a tenth of the time required for the 49 others. Compared to the 2-round design used in the early instalment of the competition (see *e.g.* Simon et al., 2005), the choice of benchmarks would be based on statistics rather than be hand-picked by the organisers.

Following up on our work regarding instance selection in Chapter 6, similar mechanisms could be designed in a machine learning context to focus the learning effort on relevant data rather than processing as much data as possible. The challenge here would be to decide what is applicable in that context. Further work is already ongoing to apply similar techniques for automated algorithm selection (Kuş et al., 2024). The key difference between our methods and the ones found in active learning is that the active learning community often assumes that acquiring a data point has a fixed cost which is the same for each data point. On the other hand, we account for the price of each point on top of how informative they are. This problem has also been studied with cost-aware active learning (see *e.g.* Tomanek and Hahn, 2010; Guillory and Bilmes, 2009) and more work can be done at the intersection of our fields.

Another element that we did not explore is the target algorithm itself. It could be interesting, following our work on algorithm selection (Pulatov et al., 2022), to define algorithm features for algorithm configuration that would inform the model used by

configurator about the meaning and impact of a specific parameter value on the inner working of the target algorithm. The challenge in this would be to describe features that represent the different configurations and are dependent on the configuration. Such feature could for example represent the amount and complexity of code activated by a Boolean parameter. There might also be a way, through source code inspection (such as *e.g.* program slicing Gallagher and Kozaitis, 2025), to find out relationships between parameters or predict how much they impact the performance. However, the differences in programming languages and code style might have a significant impact on this kind of research, for example, the syntax trees we built for the features analysed in our prior work (Pulatov et al., 2022) were largely different between the solvers in C and the ones in Java.

7.3 Final word

In this thesis, we addressed key challenges in the design of general-purpose automated algorithm configurators for running time optimisation, focusing on their sampling strategies. We provided a deeper understanding of the strength and weaknesses of current configurators through empirical analysis and developed methods to improve them. Our contributions show the potential of simple refinement to substantially improve the state of the art in AAC.

This work paves the way towards more flexibility in algorithm configuration, where the right approach would be applied to each configuration scenario. The methods we developed, in particular the ones related to instance selection, also offer the opportunity to be applied to other domains with costly evaluations and a focus on sustainability.

Our hope is that this thesis encourages further research into the underlying question: *How to design flexible systems that adapt to the problem at hand and learn more from less?*