Opening the Black Box: Automatically Characterizing Software for Algorithm Selection (Student Abstract)

Damir Pulatov, Lars Kotthoff
Department of Computer Science
University of Wyoming
{dpulatov, larsko}@uwyo.edu

Abstract

Meta-algorithmics, the field of leveraging machine learning to use algorithms more efficiently, has achieved impressive performance improvements in many areas of AI. It treats the algorithms to improve on as black boxes – nothing is known about their inner workings. This allows meta-algorithmic techniques to be deployed in many applications, but leaves potential performance improvements untapped by ignoring information that the algorithms could provide. In this paper, we open the black box without sacrificing the universal applicability of meta-algorithmic techniques by automatically analyzing the source code of the algorithms under consideration and show how to use it to improve algorithm selection performance. We demonstrate improvements of up to 82% on the standard ASlib benchmark library.

Introduction

The past decade has seen the rise of meta-algorithmic techniques that enable the more efficient and intelligent use of existing algorithms, providing an alternative to the laborious task of developing new algorithms. The first prominent proponent of this was the SATzilla system (Xu et al. 2008), which implements an algorithm selector that decides which of an existing SAT solver to run to solve a particular SAT problem instance. After the success of SATzilla, there have been other examples of utilizing meta-algorithmic techniques in practice. For example, 3S (Kadioglu et al. 2011) creates a schedule of algorithms to run on problem instances.

The above examples employ various techniques and approaches but have a unifying idea – algorithms are treated as black boxes. That is, the only property of algorithms that is known is their performance. This idea allows practitioners to ignore the underlying specifics of each algorithm. While this approach has achieved great results, there is a potential for improvement that can be gained by taking into the account the algorithms.

In this paper, we argue that ignoring the information that the algorithm can provide in this way leaves useful knowledge untapped that can improve the performance of meta-algorithmic approaches. We open the black box without compromising the general applicability of meta-algorithmic techniques by automatically analyzing the source code of the algorithms under consideration and demonstrate the performance improvements that can be achieved this way – up to 82% on MCP compared to leaving the black box closed. We leverage the standard ASlib benchmark library (Bischl et al. 2016) and a standard software analysis tool.

Related Work

The algorithm selection problem is simply defined as selecting the most appropriate algorithm for a specific instance of a problem and was first introduced by Rice in 1976 (Rice 1976). An important idea in algorithm selection is the construction of algorithm portfolios where the goal is to collect instances of the same type of problems and algorithms that solve them into one portfolio. The purpose of this portfolio is to minimize the risk of selecting a sub-optimal algorithm for a given problem instance.

The first famous algorithm selection system was SATzilla (Xu et al. 2008), which built a portfolio of SAT solvers and utilized them to solve instances of SAT. This idea was later replicated in other domains of AI, such as ASP (Hoos, Lindauer, and Schaub 2014), TSP (Kerschke et al. 2018), 3S (Kadioglu et al. 2011).

A common approach to algorithm selection is to build empirical performance models for each algorithm in a portfolio. These performance models are machine learning models trained on features of problem instances. Then, given a new problem instance, an algorithm selector chooses an algorithm with the most optimal predicted performance. To the best of our knowledge, there are no approaches that analyze the source code of an algorithm to extract information for algorithm selection.

Methodology and Results

For this project, we have combined algorithmic and instance features to build one performance model per portfolio. That is, there is only one model that predicts the performance of all algorithms for a given scenario. This is in contrast to the standard approach of building performance models for each algorithm in a scenario.

So far, we have performed experiments on six ASlib scenarios (Bischl et al. 2016) – TSP-LION2015, SAT11-RAND, SAT11-HAND, SAT11-INDU, SAT03-16_INDU and OPENML-WEKA-2017. We have also created a SAT18-EXP scenario based on the SAT-2018 competition results (Heule, Järvisalo, and Suda 2018).

We analyzed the algorithms in the above scenarios with the open-source Metrix++ tool (Metrix++ 2009) and collected the following features:

- cyclomatic complexity, the number of independent execution paths (McCabe 1976),
- maxindent complexity, the maximum level of indentation as a proxy for code complexity (Tornhill 2018),
- number of lines of code,
- size in bytes, and
- number of files.

The experimental results are summarized in Figure 1. MCP stands for misclassification penalty, which is the additional cost incurred because a sub-optimal algorithm was chosen. The single best solver is the individual solver that performs best on average across all problem instances. The virtual best solver is a solver that always chooses the best solver from a portfolio for a given instance. Together, the single and virtual best solvers provide bounds on the performance of an algorithm selection system – it cannot be better than the virtual best solver and should not be worse than the single best solver. We normalized the performances of an algorithm selection system to the fraction of the gap closed between single and virtual best, following (Bischl et al. 2016), which allows us to directly compare performance across different scenarios.

These results show that adding software features improves the performance of algorithm selection in most of cases (four out of seven). While adding algorithm features does not help for all scenarios, there is a significant performance improvements in some cases.

On average, there is a 34.07% relative improvement in terms of MCP across the four scenarios where our approach improves algorithm selection performance. Notably, TSP-LION2015 scenario sees an a relative improvement of 82% for MCP metric. For the remaining three scenarios, the relative decrease in performance is 5% in terms of MCP. These results demonstrate the promise of the new approach.

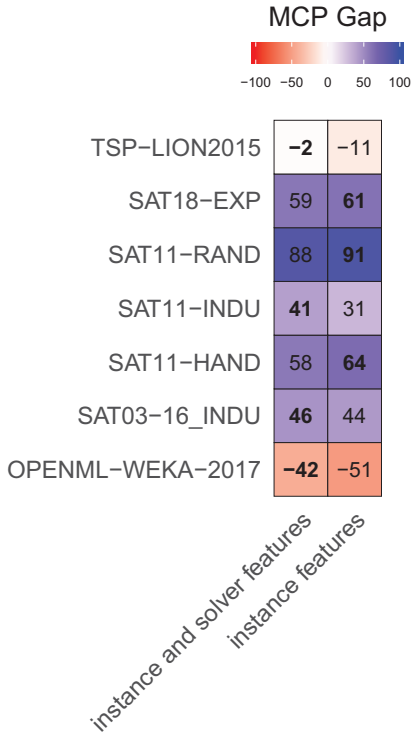

Figure 1: Percent gap closed between single best (0) and virtual best solver (100) in percent in terms of MCP. Numbers less than 0 denote performance worse than the single best solver. Values were rounded to integers. The best value for a particular scenario is shown in **bold**.

References

Bischl, B.; Kerschke, P.; Kotthoff, L.; Lindauer, M.; Malitsky, Y.; Fréchette, A.; Hoos, H. H.; Hutter, F.; Leyton-Brown, K.; Tierney, K.; and Vanschoren, J. 2016. ASlib: A Benchmark Library for Algorithm Selection. *Artificial Intelligence Journal (AIJ)* (237):41–58.

Heule, M.; Järvisalo, M.; and Suda, M., eds. 2018. *Proceedings of SAT Competition 2018*, volume B-2018-1 of *Department of Computer Science Series of Publications B*. Department of Computer Science, University of Helsinki.

Hoos, H.; Lindauer, M.; and Schaub, T. 2014. claspfolio 2: Advances in Algorithm Selection for Answer Set Programming. *TPLP* 14(4-5):569–585.

Kadioglu, S.; Malitsky, Y.; Sabharwal, A.; Samulowitz, H.; and Sellmann, M. 2011. Algorithm Selection and Scheduling. In *17th International Conference on Principles and Practice of Constraint Programming*, 454–469.

Kerschke, P.; Kotthoff, L.; Bossek, J.; Hoos, H. H.; and Trautmann, H. 2018. Leveraging TSP Solver Complementarity through Machine Learning. *Evolutionary Computation* 26(4):597–620.

McCabe, T. J. 1976. A complexity measure. *IEEE Trans. Softw. Eng.* 2(4):308–320.

Metrix++. 2009. Software metrics analysis tool. https://metrixplusplus.github.io/home.html.

Rice, J. R. 1976. The Algorithm Selection Problem. *Advances in Computers* 15:65–118.

Tornhill, A. 2018. *Software Design X-Rays: Fix Technical Debt with Behavioral Code Analysis*.

Xu, L.; Hutter, F.; Hoos, H. H.; and Leyton-Brown, K. 2008. SATzilla: Portfolio-based Algorithm Selection for SAT. *J. Artif. Intell. Res.* 32:565–606.