SpySMAC: Automated Configuration and Performance Analysis of SAT Solvers

Stefan Falkner, Marius Lindauer, and Frank Hutter

University of Freiburg
{sfalkner,lindauer,fh}@cs.uni-freiburg.de

Abstract. Most modern SAT solvers expose a range of parameters to allow some customization for improving performance on specific types of instances. Performing this customization manually can be challenging and time-consuming, and as a consequence several automated algorithm configuration methods have been developed for this purpose. Although automatic algorithm configuration has already been applied successfully to many different SAT solvers, a comprehensive analysis of the configuration process is usually not readily available to users. Here, we present SpySMAC to address this gap by providing a lightweight and easy-to-use toolbox for (i) automatic configuration of SAT solvers in different settings, (ii) a thorough performance analysis comparing the best found configuration to the default one, and (iii) an assessment of each parameter's importance using the fANOVA framework. To showcase our tool, we apply it to Lingeling and probSAT, two state-of-the-art solvers with very different characteristics.

1 Introduction

Over the last decade, modern SAT solvers have become more and more sophisticated. With this sophistication, usually the number of parameters inside the algorithm increases, and the performance may crucially depend on the setting of these parameters. For example, in the case of the prominent competition-winning solver Lingeling [3], there are 323 parameters which give rise to approximately 10^{1341} possible settings. Exploring these parameter spaces manually is tedious and time-consuming at best. Consequently, automated methods for solving this so-called algorithm configuration problem have been developed to find parameter settings with good performance on a given class of instances [13, 1, 17, 11].

Despite several success stories of automated configuration of SAT solvers [10, 16, 19], the reasons why a configuration system chose a certain parameter setting often remain unclear to SAT solver developers. Especially, information about the importance of specific parameter settings is usually not provided. To give more insights into the configuration process of SAT solvers, we present SpySMAC, a lightweight toolbox that combines: (i) the state-of-the-art algorithm configuration system SMAC [11], (ii) automatic evaluation comparing the performance of the default and the optimized configuration across training and test

instances, and (iii) an automatic method to quantify the importance of parameters, `fANOVA` [12]. In the end, `SpySMAC` generates a report with relevant tables and figures that summarize the results and reveal details about the configuration process. The standardized input and output of SAT solvers allowed us to design `SpySMAC` to be very easy to use for both developers and users of SAT solvers.

2 Algorithm Configuration and Analysis

The general task of algorithm configuration consists of determining a well-performing parameter configuration for a given instance set and a performance metric (e.g., runtime). To this end, an algorithm configuration system, or configurator for short, iteratively evaluates different configurations trying to improve the overall performance. After a given time budget is exhausted, the configuration process ends, and the configurator returns the best parameter setting found. The configurator can typically only explore a small fraction of the space of all possible configurations since that space is exponential in the number of parameters and evaluating a single configuration requires running it on multiple instances.

Several different approaches have been taken towards efficiently searching through the configuration space, among others: iterated local search (`ParamILS` [13]), genetic algorithms (`GGA` [1]), iterated racing procedures (`irace` [17]), and model-based Bayesian optimization (`SMAC` [11]). The Configurable SAT Solver Challenge (CSSC) [15] recently evaluated these configurators (except `irace`) and achieved significant speed-ups for various solvers and benchmarks. For example, in the CSSC 2014, the PAR10 score[1] of `Lingeling` [3], `clasp` [9], and `probSAT` [2] improved by up to a factor of 5, 108 and 1500, respectively. Across a wide range of solvers on a broad collection of benchmarks, `SMAC` consistently achieved the largest speedups in the challenge; therefore, we decided to use it in our tool.

`SMAC` is a sequential model-based algorithm configuration system: it models the performance metric based on finished runs (as a function of the parameter configuration used in each run and characteristics of the instance used), and uses this model to determine the next promising configuration to evaluate. It uses random forests as the underlying model [4], methods from Bayesian optimization [5], and applies mechanisms to evaluate poor configurations only on few instances terminating long runs early [11, 13].

To give some insights into the configuration process, different complementary techniques have been developed towards identifying parameter importance. These include forward selection of parameters based on an empirical performance model [14], ablation paths between the default and optimized configuration to identify the important parameter value flips [8], and functional ANOVA (`fANOVA`) to quantify the importance of parameters in the entire configuration space based on random forests as empirical performance models [12].

[1] PAR10 is the penalized average runtime where timeouts are accounted for 10 time the runtime cutoff.

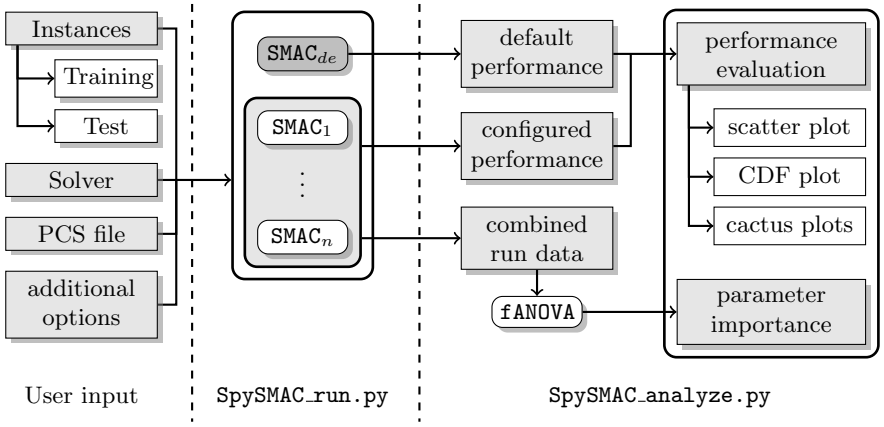

Fig. 1. Schematic of SpySMAC's workflow. The only significant user input (left) is needed to start the configuration phase via `SpySMAC_run.py`. The n independent SMAC runs search for a better configuration while SMAC$_{de}$ only evaluates the default performance. After all runs have finished, `SpySMAC_analyze.py` is called to prepare a report showing details about the configuration process and the final configuration.

Each of these methods has some advantages and disadvantages. The forward selection approach is the only of the three that can detect patterns relating instance characteristics to well-performing configurations, but forward selection can be computationally very demanding as it requires the fitting of hundreds of machine learning models. The ablation path is the only one that directly quantifies the performance difference of each changed parameter based on new experiments, but the drawback is that for long ablation paths these experiments can take even longer than the configuration step. Finally, the functional ANOVA approach is computationally efficient (it only requires fitting a single machine learning model and does not require any new algorithm runs) and does not only quantify which parameters are important but also how well each of the parameters' values perform. While we are ultimately planning to support all of these methods, SpySMAC's first version focuses on `fANOVA` to keep the computational cost of the analysis step low.

3 SpySMAC's Framework

The workflow in SpySMAC is as follows. First, the user provides information about the solver, its parameters and the instance set. Based on that, the configuration phase (running SMAC) and analysis phase (evaluating performance and parameter importance) are conducted. Figure 1 shows a schematic workflow.

The solver specifics provided by the user include the solver binary, and a specification of its parameters and their possible ranges. We use SMAC's *parameter configuration space* (PCS) file format. This format allows the declaration of

real, integer, ordinal and categorical parameters, as well as conditional parameters that are only active dependent on other (so-called parent) parameters (e.g., subparameters of a heuristic h are only active if h is selected by another parent parameter). Complex dependencies can be expressed as hierarchies of conditionalities, as well as forbidden partial assignments of parameters (e.g., if one choice for a data structure is not compatible with a certain type of preprocessing). For a detailed introduction, please refer to `SpySMAC`'s documentation. For the solvers that competed in the CSSC, these PCS files are already available, which provides many examples for writing new PCS files.

The user also needs to provide a set of benchmark instances to use for the configuration step and for the subsequent validation. It is possible to either specify the training and the test set directly, or to specify a single instance set that `SpySMAC` will split into disjoint training and test sets. Splitting the instances into two sets is necessary to get a unbiased performance estimate on unseen, new instances, to avoid over-tuning effects.

The configuration phase consists of multiple, independent `SMAC` runs (which should take place on the same type of hardware to yield comparable runtimes). Since the configuration of algorithms is a stochastic process and many local minima in the configuration space exist, multiple runs of `SMAC` can be used to improve the performance of the final configuration found. In principle, one very long run of `SMAC` would have the same effect, but multiple runs can be effectively parallelized on multi-core systems or compute clusters. We emphasize that we determine the best performing configurations among all `SMAC` runs based on the training set, not on the test set. This avoids over-tuning effects, again.

After all configuration runs have finished, the separate evaluation step can commence. The user simply executes the analyze script, `SpySMAC_analyze.py`, to automatically generate a report summarizing the results. This report includes a performance evaluation of the default configuration and the found configuration on the test and training instances[2], scatter plots to visualize the performance on each instance, as well as cumulative distribution function (CDF) and cactus plots to visualize the runtime distributions.

The analysis step can also run `fANOVA` based on the performance data collected during the configuration to compute parameter importance, producing a table with quantitative results for each parameter and plots to visualize the effect of different parameter values. The `fANOVA` step is based on a machine learning model fitted on the combined performance data of all solver runs performed in the configuration phase. For many solver runs (i.e., hundreds of thousands runs), even `fANOVA`'s computations can take up to several hours and require several GB memory; therefore, `fANOVA` is an optional part of the analyzing step.

[2] Showing the training and test performance helps to identify over-tuning effects, i.e., the performance improved on the training set but not on test set.

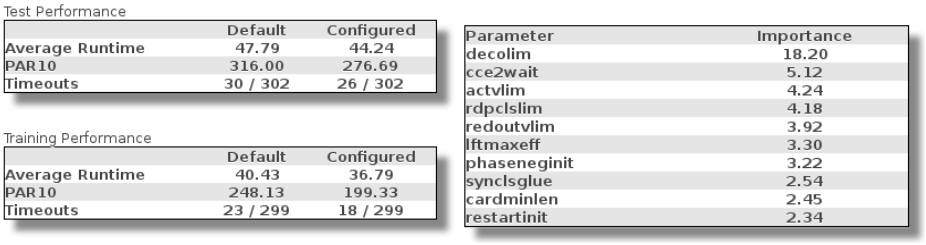

Fig. 2. Performance overview for test and training data (left), and the parameter importance determined by `fANOVA` for `Lingeling` on `CircuitFuzz` (right).

4 Spying on `Lingeling` and `probSAT`

In this section, we apply `SpySMAC` to two solvers and three different benchmarks from the Configurable SAT Solver Challenge: `Lingeling` [3] on an instance set from circuit-based CNF fuzzing (`CircuitFuzz` [6]), and `probSAT` [2] on two collections of random satisfiable CNF formulas (7-SAT instances with 90 clauses, `7SAT90-SAT`, and 3-SAT instances with 1000 clauses, `3SAT1k-SAT`, see [19]).

Figure 2 shows tables generated for the report for the `Lingeling` example. By comparing the test and training performance, one can see that `SMAC` found a configuration improving over the standard parameter setting. Even though `Lingeling`'s default already performed very well on this instance set, `SMAC` was still able to lower the average runtime further, and to reduce the number of timeouts. The table on the right shows the ten most important parameters of `Lingeling` on this set. The importance score quantifies the effect of varying a parameter across all instantiations of all other parameters. A high value corresponds to large variations in the performance meaning it is important to set this parameter to a specific value (see [12] for more detail).

As an example for `probSAT`, we used two other scenarios from the CSSC to show the differences our tool can reveal about the configuration on different instance sets. Figure 3 displays the kind of performance plots generated for the report for one of the sets. It clearly shows that configuration successfully improved the performance, reducing mean runtimes for training and test instances by more than a factor of four.

To demonstrate what insights can be gained from the analysis, Figure 4 shows the parameter importance plots for the parameter `cb1`, the constant `probSAT` uses to weight the break score in its scoring function. The `fANOVA` procedure reveals this parameter to be the most important one in both scenarios, but the values differ for the two sets: it should be set high for 7-SAT and low for 3-SAT. We note that this automatically-derived insight is aligned with expert practice for setting `probSAT`'s `cb1` parameter. By doing thorough sets of experiments, developers can use our tool to understand the impact of their parameters better, and to try to find ways to adapt parameters based on prior knowledge, such as the clause length in our example here.

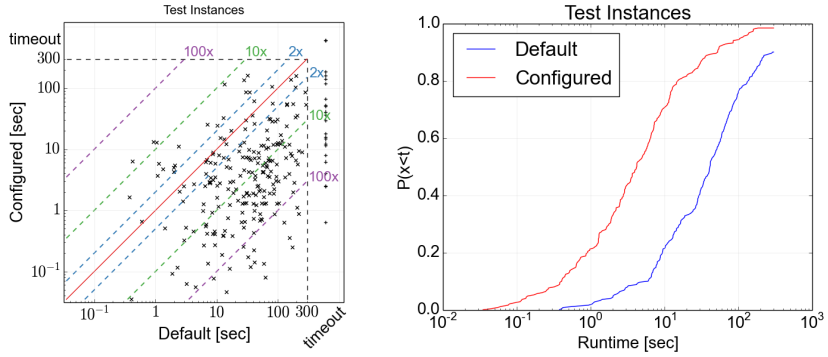

Fig. 3. Example scatter plot (left) and CDF plot (right) from applying `SpySMAC` to `probSAT` on `7SAT90-SAT`. Both show that parameter tuning significantly improves the overall performance across the whole range of runtimes.

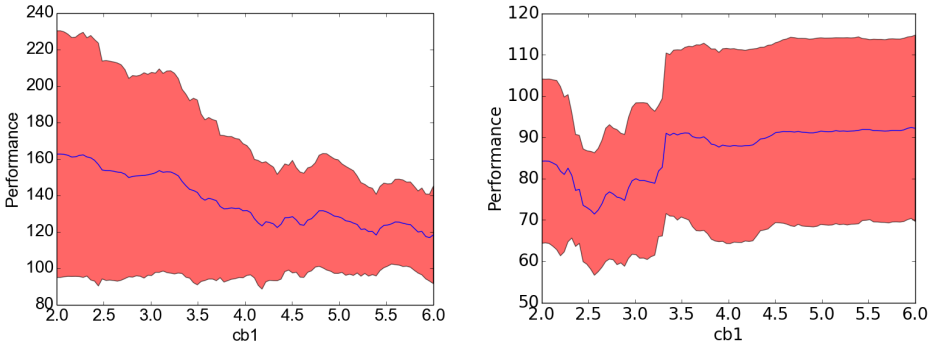

Fig. 4. Parameter importance plot for the most important parameter `cb1` on `7SAT90-SAT` (left) and `3SAT1k-SAT` (right). The plots show the mean performance (blue line) with confidence intervals (red area) as a function of `cb1`, marginalized over all other parameters. The best found configurations set it to 4.35 and 2.86 respectively.

5 Conclusion

We have presented `SpySMAC`: a tool for automatic SAT solver configuration using `SMAC` combined with extensive analysis allowing the user to "spy" into the configuration process of a solver on a given instance set. The report `SpySMAC` generates offers some insight into performance improvements and also quantifies parameter importance by applying `fANOVA`. We have shown for three examples how the framework works, re-running and analyzing three CSSC scenarios effortlessly. For the future, we plan to integrate more methods to evaluate parameter importance, including ablation [8] and forward selection to identify key parameters [14]. `SpySMAC` is available at `www.ml4aad.org/spysmac` under AGPL license with a long list of examples.

References

1. Ansótegui, C., Sellmann, M., Tierney, K.: A gender-based genetic algorithm for the automatic configuration of algorithms. In: Gent, I. (ed.) Proceedings of the Fifteenth International Conference on Principles and Practice of Constraint Programming (CP'09). Lecture Notes in Computer Science, vol. 5732, pp. 142–157. Springer-Verlag (2009)
2. Balint, A., Schöning, U.: Choosing probability distributions for stochastic local search and the role of make versus break. In: Cimatti and Sebastiani [7], pp. 16–29
3. Biere, A.: Yet another local search solver and lingeling and friends entering the SAT competition 2014. In: Belov, A., Diepold, D., Heule, M., Järvisalo, M. (eds.) Proceedings of SAT Competition 2014: Solver and Benchmark Descriptions. Department of Computer Science Series of Publications B, vol. B-2014-2, pp. 39–40. University of Helsinki (2014)
4. Breimann, L.: Random forests. Machine Learning Journal 45, 5–32 (2001)
5. Brochu, E., Cora, V., de Freitas, N.: A tutorial on Bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. Computing Research Repository (CoRR) abs/1012.2599 (2010)
6. Brummayer, R., Lonsing, F., Biere, A.: Automated testing and debugging of SAT and QBF solvers. In: Cimatti and Sebastiani [7], pp. 44–57
7. Cimatti, A., Sebastiani, R. (eds.): Proceedings of the Fifteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'12), Lecture Notes in Computer Science, vol. 7317. Springer-Verlag (2012)
8. Fawcett, C., Hoos, H.: Analysing differences between algorithm configurations through ablation. Journal of Heuristics pp. 1–28 (2015)
9. Gebser, M., Kaufmann, B., Schaub, T.: Conflict-driven answer set solving: From theory to practice. Artificial Intelligence 187-188, 52–89 (2012)
10. Hutter, F., Babić, D., Hoos, H., Hu, A.: Boosting verification by automatic tuning of decision procedures. In: O'Conner, L. (ed.) Formal Methods in Computer Aided Design (FMCAD'07). pp. 27–34. IEEE Computer Society Press (2007)
11. Hutter, F., Hoos, H., Leyton-Brown, K.: Sequential model-based optimization for general algorithm configuration. In: Coello, C. (ed.) Proceedings of the Fifth International Conference on Learning and Intelligent Optimization (LION'11). Lecture Notes in Computer Science, vol. 6683, pp. 507–523. Springer-Verlag (2011)
12. Hutter, F., Hoos, H., Leyton-Brown, K.: An efficient approach for assessing hyperparameter importance. In: Xing, E., Jebara, T. (eds.) Proceedings of the 31th International Conference on Machine Learning, (ICML'14). vol. 32, pp. 754–762. Omnipress (2014)
13. Hutter, F., Hoos, H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)
14. Hutter, F., Hoos, H.H., Leyton-Brown, K.: Identifying key algorithm parameters and instance features using forward selection. In: Pardalos, P., Nicosia, G. (eds.) Proceedings of the Seventh International Conference on Learning and Intelligent Optimization (LION'13), Lecture Notes in Computer Science, vol. 7997, pp. 364–381. Springer-Verlag (2013)
15. Hutter, F., Lindauer, M., Balint, A., Bayless, S., Hoos, H.H., Leyton-Brown, K.: The Configurable SAT Solver Challenge. Computing Research Repository (CoRR) (2015), `http://arxiv.org/abs/1505.01221`

16. KhudaBukhsh, A., Xu, L., Hoos, H., Leyton-Brown, K.: SATenstein: Automatically building local search SAT solvers from components. In: Boutilier, C. (ed.) Proceedings of the 22th International Joint Conference on Artificial Intelligence (IJCAI'09). pp. 517–524 (2009)
17. López-Ibáñez, M., Dubois-Lacoste, J., Stützle, T., Birattari, M.: The irace package, iterated race for automatic algorithm configuration. Tech. rep., IRIDIA, Université Libre de Bruxelles, Belgium (2011), http://iridia.ulb.ac.be/IridiaTrSeries/IridiaTr2011-004.pdf
18. Sakallah, K., Simon, L. (eds.): Proceedings of the Fourteenth International Conference on Theory and Applications of Satisfiability Testing (SAT'11), Lecture Notes in Computer Science, vol. 6695. Springer (2011)
19. Tompkins, D., Balint, A., Hoos, H.: Captain jack: New variable selection heuristics in local search for SAT. In: Sakallah and Simon [18], pp. 302–316