SMAC and XGBoost Your Theorem Prover

Edvard K. Holden and Konstantin Korovin

The University of Manchester, U.K.

Abstract

Heuristic selection for automated theorem provers [6, 7, 11] has received considerable attention in recent years [1, 4, 5, 8, 9, 10, 12]. Various heuristics for proof searching yield dramatically different solving times for different problems. In this paper we introduce methods for learning good heuristics for sets of diverse problems via parameter optimisation and dynamic clustering. We also propose a method for predicting the optimal heuristic and estimating the solving time of a given problem. We divide the system into two phases: the heuristic learning phase (HLP), and the heuristic mapping phase (HMP).

Heuristic Learning Phase: HLP

The goal of this phase, presented in Algorithm 1, is to learn diverse heuristics using SMAC [3] and at the same time cluster problems based on heuristics performance. SMAC (Sequential Model-based Algorithm Configuration) is a hyperparameter optimiser which can optimise both numerical and categorical parameters. It builds a model for selecting promising configurations, by creating an ensemble of regression trees over the space of the parameter options.

We can run SMAC over a collection of problems with the goal of optimising the number of problems solved. However, problem sets are usually diverse: different problems require different heuristics, and learning a single heuristic which is globally optimal does not necessarily result in a useful heuristic for specific problems. We propose to cluster similar problems and optimise heuristics for each cluster separately. To create collections of problems with some similarity, we cluster the problems based on the syntactic problem features and in addition we use dynamic features representing the heuristics performance on the problems. These features are normalised, weighted and combined to create a feature vector which is used to produce K problem clusters by applying the K-means algorithm.

A syntactic feature is a feature representing syntactic properties of the problem. Such features can be the number of EPR or Horn clauses. A selection of syntactic features constructs $problem\_feature\_vector$. The $heuristic\_evaluation\_vector$ consists of the solving times for the problem when ran over the set of heuristics. A special value is used for time outs. We combine $problem\_feature\_vector$ and $heuristic\_evaluation\_vector$ to cluster problems and run the SMAC heuristic optimisation over these clusters separately (the inner for loop in Algorithm 1). After optimising the heuristics over the clusters, we evaluate the best local heuristics over the whole problem set and re-cluster problems based on the new heuristics performance.

Heuristic Mapping Phase: HMP

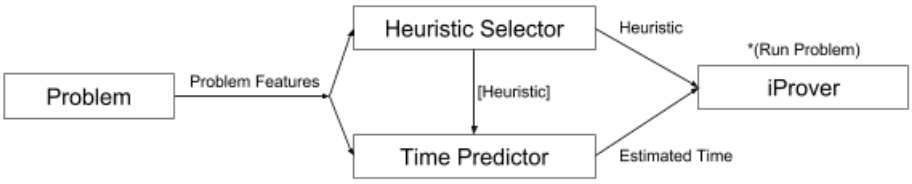
The second phase consists of building an automatic heuristic selector which selects the optimal heuristic concerning the solving time from a set of heuristics. This heuristic set is the result of the HLP phase as it has discovered optimal heuristics for subsets of problems. Based on this data we construct the dataset $D = [(x_1, y_1), ..., (x_n, y_n)]$ where x is the $problem\_feature\_vector$ and y is the optimal heuristic for the problem. Hence, we can represent the heuristic mapping by the function $f : x \to y$. We can approximate this function by utilising supervised machine learning methods. In particular, we can use XGBoost [2] which is an implementation of Gradient

Algorithm 1 Heuristic Learning Phase

```
best_heuristics ← get_initial_heuristics()
problem_feature_vector ← get_problem_features()
repeat
    heuristic_evaluation_vector ← compute_proving_times(best_heuristics)
    problem_clusters ← kmeans(problem_feature_vector, heuristic_evaluation_vector)
    new_heuristics ← ∅
    for cluster ∈ problem_clusters do
        problem_sample ← get_problem_sample(cluster)
        run_SMAC(best_heuristic(cluster), problem_sample)
        new_heuristics ← new_heuristics ∪ get_optimal_heuristics_from_SMAC_run()
    end for
    new_heuristics ← get_top_heuristics(problem_clusters)
    best_heuristics ← best_heuristics ∪ new_heuristics
until Timeout
```

Boosting Machines. This model is known to be quite fast, have state-of-the-art performance and to control overfitting better than most alternatives. It is therefore reasonably to assume that the model will perform well on this dataset, even though it is likely to be imbalanced.

Figure 1: Heuristic Mapping and Regression Overview



There may be the case that a heuristic cannot solve a problem. This occurs either if the model predicted the wrong heuristic or the problem is previously unseen, in which case a heuristic which can solve the problem does not exist in the set. These situations can waste an unnecessary amount of resources, as the prover will not be able to solve the problem within a global time constraint. To reduce this resource waste, we propose to create a regression model which estimates the solving time of a problem. The problems can be represented by the $problem\_feature\_vector$. As the runtime is dependent on the heuristic, we can encode the heuristic as a one-hot vector to get a proper representation of the input to the prover. This regression model can be constructed by non-linear regression using XGBoost or neural networks.

Conclusion

The HLP phase is implemented as a Python wrapper around the SMAC framework which runs iProver as its target function. SMAC has an option for sharing the model between multiple SMAC instances which allows us to optimise the same model over several server nodes. All the experiment data is stored in a database. The HMP phase interacts with the database and the machine learning models, both for training and evaluation.

Experimental results over the CASC'18 FOF data set show that the HLP-prediction phase manages to increase the number of solved problems by 24% in 30 hours, starting with the

default iProver heuristic. The HMP phase reduces the average solving time over jointly solved problems by 40%.

One of the related methods is BlistrTune [5] which interleaves a search for high-level parameters with fine-tuning for the heuristic invention. Our approach differs as it does not interleave during the search but utilises a model based configuration algorithm, as well as it optimises over subsets of problems from dynamically generated clusters.

In [1] we see automatic heuristic selection using Support Vector Machines and Gaussian Processes on a general set of heuristics, whereas we utilise a tree boosting algorithm on a specialised set of heuristics. The system E-MaLeS [8] uses a Gaussian Kernel to perform heuristic selection by selecting the best time estimate of a problem for each candidate heuristic. Our approach differs by treating heuristic selection as a classification task and time estimation as a regression problem. The separation of concerns may lead to better performing components which can significantly improve the overall system.

References

[1] James P. Bridge, Sean B. Holden, and Lawrence C. Paulson. Machine learning for first-order theorem proving. *Journal of Automated Reasoning*, 53(2):141–172, Aug 2014.

[2] Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. *CoRR*, abs/1603.02754, 2016.

[3] F. Hutter, H. H. Hoos, and K. Leyton-Brown. Parallel algorithm configuration. In *Proc. of LION-6*, pages 55–70, 2012.

[4] Jan Jakubuv and Josef Urban. Blistrtune: hierarchical invention of theorem proving strategies. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France*, pages 43–52. ACM, 2017.

[5] Jan Jakubuv and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK. Proceedings*, volume 10383 of *LNCS*, pages 292–302. Springer, 2017.

[6] Konstantin Korovin. iProver - an instantiation-based theorem prover for first-order logic (system description). In *IJCAR 2008. Proceedings*, pages 292–298, 2008.

[7] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV. Proceedings*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[8] Daniel Kühlwein, Stephan Schulz, and Josef Urban. E-males 1.1. In Maria Paola Bonacina, editor, *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, 2013. Proceedings*, volume 7898 of *LNCS*, pages 407–413. Springer, 2013.

[9] Giles Reger, Martin Suda, and Andrei Voronkov. New Techniques in Clausal Form Generation. In Christoph Benzmüller, Geoff Sutcliffe, and Raul Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016.

[10] Simon Schäfer and Stephan Schulz. Breeding theorem proving heuristics with genetic algorithms. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *GCAI 2015. Global Conference on Artificial Intelligence*, volume 36 of *EPiC Series in Computing*, pages 263–274. EasyChair, 2015.

[11] Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR-19*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[12] Josef Urban. Blistr: The blind strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, 2015*, volume 36 of *EPiC Series in Computing*, pages 312–319. EasyChair, 2015.