

Ranking Model Checking Backends for Automated Selection via Classification and Regression Learning

Jannik Dunkelau, Leo Baldus

Heinrich-Heine-Universität Düsseldorf, Universitätsstr. 1, 40225 Düsseldorf, Germany

Abstract

With classification and regression learning, we employ ranking approaches for the constraint solving backends of the PROB model checker to assess the most suited backend to use for a given constraint. In case the top ranked backend fails to solve the constraint, the predicted ranking yields a clear order of which backend to utilise in a second attempt. Our trained predictors achieve an overall runtime overhead of only 7 % while a static order of backends results in an overhead of 8.5 %. Thus, we are confident in our approach as it is more dynamic and offers potential to increase performance in the future.

Keywords

Model checking, constraint solving, automated backend selection, ranking algorithms, machine learning

1. Introduction

In model checking, we exhaustively search for every reachable state of a program to show that no erroneous states are ever reachable. Starting with an initial state, we check for each consecutively reachable state whether it violates an invariant which specifies the desired behaviour. This leads to a number of constraints implied by the state space which need to be validated. However, solving these constraints is not always straightforward, simple, or even possible with different tools being more suitable for certain instances than others as outlined by the *no free lunch theorems* [1, 2] and the *algorithm selection problem* [3].

In the following, we concern ourselves with a selection problem over three different constraint solving backends of PROB [4, 5], a model checker for the B Method [6]. Its core is written in SICStus Prolog [7] and uses SICStus' CLP(FD) library [8] in its default constraint solving backend. PROB further provides bindings to the SAT-solving based Kodkod backend [9, 10] and to the DPLL(T) based SMT backend [11] which utilises Z3 [12]. Each of these three backends was already shown to work best in different subdomains of the problem space [13]. We present a ranking approach for the constraint solving backends of the PROB model checker to assess for a given constraint solving instance the most suited backend to use, defined as the backend yielding a definite answer ("constraint is valid or invalid") the fastest. Ranking the backends instead of only classifying the single most suited is beneficial in case of mispredictions: if the


OVERLAY'21: Workshop on Artificial Intelligence and fOrmal VERification, Logic, Automata, and sYnthesis, 22nd September, 2021, Padova, Italy

✉ jannik.dunkelau@hhu.de (J. Dunkelau); leo.baldus@hhu.de (L. Baldus)

ORCID 0000-0003-0819-5554 (J. Dunkelau)



© 2021 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

 CEUR Workshop Proceedings (CEUR-WS.org)

predicted backend fails to satisfy or invalidate the constraint, a ranking clearly defines which backend to use for solving the constraint next.

In the following, we briefly discuss related work in Section 2. Our training data, used machine learning algorithms, and experimental results are summarised in Sections 3 and 4. The paper concludes in Section 5. The code for our experiments and our full results are available on Github: <https://github.com/jdnklau/ranked-runtime-predictions-for-probs-backends>.

2. Related Work

Previous work investigated classification approaches [13, 14, 15] in which a classifier predicts only the best suited backend for a given constraint. While initially done with neural networks [14], follow-up work focused on decision trees and random forests [13, 15]. The conducted ranking via regression method is inspired by the works of Healy et al. [16] in which an algorithm portfolio was created for the Why3 platform [17]. A more recent approach was done by Scott et al. in the form of MachSMT [18]. Internally it uses AdaBoost [19] over 200 decision trees to achieve a ranking over the specified solvers. Due to its only recent publication, we were unable to compare our approach with MachSMT in time.

3. The Training Data and Feature Set

We utilise the set of 597,134 training constraints collected over the ProB public examples¹ by Dunkelau et al. [13] as well as their presented set of 109 features over the B language extended by another feature accounting for the use of cardinality constraints for sets. Thus we arrive at 110 features, measuring frequencies of operators and language subdomains in the constraints.

Given a possible backend response $r \in \{\text{valid}, \text{invalid}, \text{unknown}, \text{timeout}, \text{error}\}$ and a measured runtime t over each backend with a timeout of δ (in our application 25 s), each constraint was labelled with one cost value per backend given by the cost function by Healy et al. [16] in Equation (1). 25 % of the data were withheld for performance tests and not used during training.

$$\text{cost}(r, t) = \begin{cases} t & \text{if } r \in \{\text{valid}, \text{invalid}\} \\ t + \delta & \text{if } r = \text{unknown} \\ t + 2\delta & \text{if } r \in \{\text{timeout}, \text{error}\} \end{cases} \quad (1)$$

4. Predicting Runtime Rankings

As we want to favour faster backends over slower backends yet also definitive responses over unknown, timeout, or error responses in our ranking, we utilise the cost function from Equation (1). This reduces our goal to predicting the backends in ascending order of their costs. We can make use of three different approaches.

Multi-output regression is the approach followed by Healy et al. [16]. One singular predictor is trained to predict the three target variables (the backends' cost values) simultaneously. The

¹https://www3.hhu.de/stups/downloads/prob/source/ProB_public_examples.tgz

advantage lies in only having to train and maintain one regressor which is able to share internal inferences across the three output dimensions.

In separate single-output regression only one independent regressor is trained per backend. This allows for easier maintenance in the long run as optimizing performance for a single backend does not impact the other regressors. Adding a new backend into the portfolio also only consists of adding a new, optimised regressor instead of retraining the singular regressor over all backends.

A non-regression approach is ranking classification. Each possible ranking is defined as its own class, leading to 6 classes in total. This however is even less extendable than the multi-output regression as adding a new backend not only leads to retraining of the whole classifier but also increases the amount of classes by a factorial factor, losing feasibility by increasing number of backends. We also lose the information of predicted runtime cost per backend.

4.1. Machine Learning Algorithms

We compared the performances of multiple machine learning algorithms, employing a total of six different algorithms: linear regression (LR) [20], ridge regression (RG) [21], k nearest neighbors (KNN) [22], support vector machines (SVM) [23], decision trees (DT) [24], and random forests (RF) [25]. For each learning algorithm, we conducted an extensive hyperparameter search via grid search and evaluated performance via 5-fold cross validation. As random forests yielded the best results in our experiments (cf. Table 1), we omit details on the other algorithms due to space reasons. Random forests are an ensemble approach in which multiple decision trees are trained. Decision tree training constructs predictors of a tree like structure in which each inner node splits the training data along a chosen feature dimension into more pure subsets thus reducing variance in the labels. For random forests, each decision tree in the ensemble is trained on random subsets of both the training data and the feature set.

4.2. Measuring Performance

To quantify the performance of our ranking predictors we utilise the following two metrics. The first is the *double normalised discounted cumulative gain* (dnDCG) [26], a ranking evaluation metric that penalises misplacement of higher ranked items more strictly as it deems these more impactful. It produces a value between 0 and 1, with 1 being the best possible performance. The formula for the dnDCG is stepwise calculated by

$$\text{DCG}_p = \sum_{i=1}^p \frac{2^{\text{rel}_i} - 1}{\log_2(i + 1)}, \quad \text{nDCG}_p^* = \frac{\text{DCG}_p}{\text{DCG}_p^*}, \quad \text{dnDCG}_p = \frac{\text{nDCG}_p - \text{nDCG}_p^\omega}{1 - \text{nDCG}_p^\omega} \quad (2)$$

for a ranking of length p , where rel_i is the relevance of the i th backend, DCG_p^* is the DCG value for the ideal ranking, and nDCG_p^ω is the nDCG value for the worst ranking. The second metric is the *usage duration comparison value* (UDC) which we defined ourselves to assess the runtime overhead introduced by predicting a non-ideal ranking. This is motivated by observing that, e.g., picking the second best backend first might only add a couple nano seconds to the final runtime and thus introduces no significant overhead. If the firstly ranked backend fails to return a definite answer we would query the remaining backends in order of their ranking, until either

Table 1

Performance of each algorithm per ranking approach. Highlighted are best results per approach. From the 216 combinations for single-output regression, we only present the best (3×RF) due to space reasons.

Approach	dnDCG						UDC					
	DT	RF	KNN	SVM	LR	RG	DT	RF	KNN	SVM	LR	RG
SO reg.	-	0.79	-	-	-	-	-	1.08	-	-	-	-
MO reg.	0.80	0.80	0.80	0.74	0.69	0.69	1.08	1.07	1.07	1.10	1.28	1.28
Class.	0.80	0.81	0.80	0.74	-	0.76	1.09	1.07	1.08	1.14	-	1.11

one gives a definite answer or we run out of backends. Let the time used up by this query be \hat{t}_i for a constraint i . Let the query time over the ideal ranking be t_i^* . We define the UDC as

$$\text{UDC} = \frac{\sum_{i=1}^n \hat{t}_i}{\sum_{i=1}^n t_i^*} \in [1, \infty). \quad (3)$$

4.3. Results

Table 1 shows the best performing models per approach. While we investigated multiple machine learning algorithms, it stands out that random forests took part in all the best achieved performances. This is inline with the results, observations, and decisions from related work [13, 15, 16]. The single-output regression performs slightly worse than multi-output regression and ranking classification, however not by any significant amount. The best UDC was achieved by multi-output regression with a value of 1.07, standing for 7 % longer runtime over all constraints in the test set compared to always using the ideal ranking.

In comparison, a static algorithm that always uses the ranking CLP(FD) \succ Z3 \succ Kodkod already yields a UDC of 1.085, competing with our predictors. However, our approach is more dynamic and allows for improvement due to further tuning or a more refined feature set.

5. Conclusion

We trained multiple predictors for ranking the PROB backends by ascending, weighted, expected runtime. Our results show not much of a difference between the approaches of single-output regression, multi-output regression, and ranking classification, while the single-output regression still represents the most flexible approach and should be focus of research going forward. Finally, we achieved performance values of a double normalised discounted cumulative gain of up to 0.81 and an usage difference comparison value of up to 1.07. While a dummy approach which always prefers a single, static ranking competes with our learned predictors we are confident that our method proves more valuable going forward as it allows for further tuning.

Acknowledgments

Computational support and infrastructure was provided by the “Centre for Information and Media Technology” (ZIM) at the University of Düsseldorf (Germany).

References

- [1] D. H. Wolpert, W. G. Macready, et al., No free lunch theorems for search, Technical Report, Technical Report SFI-TR-95-02-010, Santa Fe Institute, 1995.
- [2] D. H. Wolpert, W. G. Macready, No free lunch theorems for optimization, *IEEE transactions on evolutionary computation* 1 (1997) 67–82.
- [3] J. R. Rice, The algorithm selection problem, *Advances in computers* 15 (1976) 65–118.
- [4] M. Leuschel, M. Butler, ProB: A model checker for B, in: *FME 2003: Formal Methods*, volume 2805, Springer, Berlin, Heidelberg, 2003, pp. 855–874.
- [5] M. Leuschel, M. Butler, ProB: An automated analysis toolset for the B method, *International Journal on Software Tools for Technology Transfer* 10 (2008) 185–203.
- [6] J.-R. Abrial, *The B-Book: Assigning Programs to Meanings*, Cambridge University Press, New York, NY, USA, 1996.
- [7] M. Carlsson, J. Widen, J. Andersson, S. Andersson, K. Boortz, H. Nilsson, T. Sjöland, *SICStus Prolog user’s manual*, volume 3, Swedish Institute of Computer Science Kista, Sweden, 1988.
- [8] M. Carlsson, G. Ottosson, B. Carlson, An open-ended finite domain constraint solver, in: *Programming Languages: Implementations, Logics, and Programs*, volume 1292, Springer, 1997, pp. 191–206.
- [9] E. Torlak, D. Jackson, Kodkod: A relational model finder, in: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2007, pp. 632–647.
- [10] D. Plagge, M. Leuschel, Validating B, Z and TLA+ using prob and kodkod, in: *International Symposium on Formal Methods*, Springer, 2012, pp. 372–386.
- [11] S. Krings, M. Leuschel, SMT solvers for validation of B and Event-B models, in: *International Conference on Integrated Formal Methods*, Springer, 2016, pp. 361–375.
- [12] L. De Moura, N. Bjørner, Z3: An efficient SMT solver, *Tools and Algorithms for the Construction and Analysis of Systems* (2008) 337–340.
- [13] J. Dunkelau, J. Schmidt, M. Leuschel, Analysing ProB’s constraint solving backends, in: *International Conference on Rigorous State-Based Methods*, Springer, 2020, pp. 107–123. doi:10.1007/978-3-030-48077-6_8.
- [14] J. Dunkelau, S. Krings, J. Schmidt, Automated backend selection for ProB using deep learning, in: *NASA Formal Methods*, volume 11460 of *LNCS*, Springer, 2019, pp. 130–147. doi:10.1007/978-3-030-20652-9_9.
- [15] J. Petrasch, *The Decision Does Not Fall Far from the Tree: Automatic Configuration of Predicate Solving*, Master’s thesis, Heinrich Heine Universität Düsseldorf, Universitätsstraße 1, 40225 Düsseldorf, 2018.
- [16] A. Healy, R. Monahan, J. F. Power, Predicting SMT solver performance for software verification, in: *Proceedings of the Third Workshop on Formal Integrated Development Environment*, volume 240 of *EPTCS*, 2017, pp. 20–37. doi:10.4204/EPTCS.240.2.
- [17] J.-C. Filliâtre, A. Paskevich, Why3—where programs meet provers, in: *European symposium on programming*, Springer, 2013, pp. 125–128.
- [18] J. Scott, A. Niemetz, M. Preiner, S. Nejati, V. Ganesh, MachSMT: A machine learning-based algorithm selector for SMT solvers, in: *Tools and Algorithms for the Construction and*

- Analysis of Systems, volume 12652 of *LNCS*, Springer, 2021, pp. 303–325. doi:10.1007/978-3-030-72013-1_16.
- [19] Y. Freund, R. E. Schapire, A decision-theoretic generalization of on-line learning and an application to boosting, *Journal of Computer and System Sciences* 55 (1997) 119–139. doi:10.1006/jcss.1997.1504.
- [20] G. D. Hucheson, Ordinary least-squares regression, L. Moutinho and GD Hucheson, *The SAGE dictionary of quantitative management research* (2011) 224–228.
- [21] A. E. Hoerl, R. W. Kennard, Ridge regression: Biased estimation for nonorthogonal problems, *Technometrics* 12 (1970) 55–67.
- [22] T. Cover, P. Hart, Nearest neighbor pattern classification, *IEEE transactions on information theory* 13 (1967) 21–27.
- [23] C. Cortes, V. Vapnik, Support-vector networks, *Machine learning* 20 (1995) 273–297.
- [24] L. Breiman, J. Friedman, C. J. Stone, R. A. Olshen, *Classification and regression trees*, CRC press, 1984.
- [25] L. Breiman, Random forests, *Machine learning* 45 (2001) 5–32.
- [26] K. Järvelin, J. Kekäläinen, Cumulated gain-based evaluation of IR techniques, *ACM Transactions on Information Systems (TOIS)* 20 (2002) 422–446.