Characteristic Subsets of TPTP Benchmarks*

Karel Chvalovský and Jan Jakubův

Czech Technical University in Prague, Prague, Czech Republic

1 Motivation: ATP Evaluation over Large Benchmarks

When developing an Automated Theorem Prover (ATP), like E [21], Vampire [16], or Prover9 [18], one often needs to evaluate the prover over a large set of benchmark problems. This is typically done to empirically evaluate a new implementation when a new feature or a new proof search strategy are implemented in the prover.

There are several standardized and well-established libraries of first-order problems maintained with the goal to help developers to evaluate the generality of their provers. First of all, there is the TPTP library of problems in the TPTP language [22], which became widely adopted by the ATP community. Next, a large number of problems are being translated from large mathematical libraries of interactive theorem provers like Mizar [10], Coq [7], and Isabelle [19], by dedicated systems [4] like MPTP [24], CoqHammer [8], and SledgeHammer [5]. Moreover, an annual automated theorem prover competition (CADE) [23] introduces special tracks with additional interesting problems, like problems coming from the AIM project [15].

Evaluating a single prover strategy over all first-order problems in TPTP with a standard time limit (like 5 minutes) can easily take several hours, even with the help of massive parallelization. The situation gets even worse when more than one strategy, or a parametric strategy with many arguments, needs to be evaluated. This is usually approached by restricting the evaluation time limit or by selecting a random benchmark subset to obtain the evaluation results in a reasonable time.

On the other hand, within large problem libraries, many of the problems can be syntactically similar or even duplicate. Since similar prover performance can be expected with similar problems, the identification of similar problems can help us to speed up the evaluation. Here we propose a method to construct the *benchmark characteristic subset* by employing standard machine learning methods for clustering [20]. The desired property of this characteristic subset is that it faithfully characterizes all benchmark problems. That is, that any development, like parameter tuning or scheduler construction, performed on this subset yields similar results as the same development performed on all benchmark problems, but faster.

2 Benchmark Clustering and Characteristic Subsets

Our proposed method to construct the *benchmark characteristic subset* is to employ clustering algorithms. Clustering algorithms can partition a set of entities into classes, called *clusters*, such that similar entities appear in the same cluster. As the benchmark characteristic subset should not contain similar problems, we propose to cluster all benchmark problems and to construct the characteristic subset by selecting one or a few problems from each cluster.

*Supported by the ERC Project *SMART* Starting Grant no. 714034, the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902, and the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/0000466 and the European Regional Development Fund.

Problem Features. To employ clustering algorithms, such as the k-means [17] algorithm, benchmark problems need to be represented by numeric *feature vectors*. We propose two methods to compute feature vectors.

(1) To use the ENIGMA features [12, 13, 6, 11] based on symbol-independent clause syntax. The ENIGMA features are designed to represent first-order clauses by encapsulating various syntactic properties, like clause length, variable count, and various encodings of clause syntax trees. We plan to represent each problem by accumulating ENIGMA features of the problem clauses.

(2) To compute *performance features* obtained from the statistics of short probing prover runs. We perform short resource-limited runs with several E Prover strategies, and we extract several runtime statistics, like the number of processed/generated clauses, the count of performed rewriting steps, and so on.

Clustering. We employ several clustering algorithms, such as the k-means [17] algorithm and the density-based DBSCAN [9]. From each cluster, we select one or a few problem representants, and we add them to the benchmark characteristic subset. Thus, the desired size of the characteristic subset can be controlled by the count of clusters. Hence, we can compute characteristic subsets of various sizes.

Benchmarks. We propose two separate experiments. The first is on the first-order problems from TPTP [22]. The second is on Mizar40 [14] problems coming from translation of the Mizar Mathematical Library [10] to first-order logic.

Evaluation. Once the benchmark characteristic subsets of different sizes are computed, we need to evaluate their quality. The first evaluation method is based on the construction of the *best cover*. Suppose we evaluate the set of prover strategies S on all benchmark problems P. We can then construct the strategy subset $S_0 \subseteq S$ (of a given size) which maximizes the performance on a given benchmark characteristic subset $P_0 \subseteq P$. The subset S_0 is called the *best cover* on P_0. We can compare the performance of S_0 on all problems P with the best cover S' computed on all problems P and measure their relative error. Moreover, we can compute the relative error of the best covers constructed on the random benchmark subsets. Then we can verify that our benchmark characteristic subsets provide a better benchmark characterization than random subsets of the same size.

Another method of evaluation is based on a parameter search performed on a problem subset. For example, we can perform a parameter grid search on the parameters of a parametrized prover strategy. Then we can compare the performance of the best parameters found on a benchmark characteristic subset with the best parameters found on a random subset of the same size.

Related Work. A similar approach to construct a benchmark characteristic subset, limited to the use of performance features and to the evaluation on the best cover construction, has been recently[1] successfully experimented with using the SMT-LIB library [1, 2] with the help of the CVC4 [3] solver to compute performance features. Here we shift our attention to the TPTP problems, we experiment with syntactic problem features, and we provide additional clustering and evaluation methods.

[1] Currently under review.

References

[1] Clark Barrett, Pascal Fontaine, and Cesare Tinelli. The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org`, 2016.

[2] Clark Barrett, Aaron Stump, and Cesare Tinelli. The SMT-LIB Standard: Version 2.0. In A. Gupta and D. Kroening, editors, *Proceedings of the 8th International Workshop on Satisfiability Modulo Theories (Edinburgh, UK)*, 2010.

[3] Clark W. Barrett, Christopher L. Conway, Morgan Deters, Liana Hadarean, Dejan Jovanovic, Tim King, Andrew Reynolds, and Cesare Tinelli. CVC4. In *CAV*, volume 6806 of *Lecture Notes in Computer Science*, pages 171–177. Springer, 2011.

[4] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.

[5] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In Jürgen Giesel and Reiner Hähnle, editors, *Proc. of the 5th IJCAR, Edinburgh*, volume 6173 of *LNAI*, pages 107–121. Springer, 2010.

[6] Karel Chvalovský, Jan Jakubův, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In *CADE*, volume 11716 of *Lecture Notes in Computer Science*, pages 197–215. Springer, 2019.

[7] The Coq Proof Assistant. `http://coq.inria.fr`.

[8] Lukasz Czajka, Burak Ekici, and Cezary Kaliszyk. Concrete semantics with coq and coqhammer. In *CICM*, volume 11006 of *Lecture Notes in Computer Science*, pages 53–59. Springer, 2018.

[9] Martin Ester, Hans-Peter Kriegel, Jörg Sander, and Xiaowei Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. In *KDD*, pages 226–231. AAAI Press, 1996.

[10] Adam Grabowski, Artur Korniłowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.

[11] Jan Jakubův, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020.

[12] Jan Jakubův 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, July 17-21, 2017, Proceedings*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017.

[13] Jan Jakubův and Josef Urban. Enhancing ENIGMA given clause guidance. In *CICM*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.

[14] Cezary Kaliszyk and Josef Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.

[15] Michael K. Kinyon, Robert Veroff, and Petr Vojtechovský. Loops with abelian inner mapping groups: An application of automated deduction. In Maria Paola Bonacina and Mark E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 151–164. Springer, 2013.

[16] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.

[17] Stuart P. Lloyd. Least squares quantization in PCM. *IEEE Trans. Inf. Theory*, 28(2):129–136, 1982.

[18] William McCune. Prover9 and Mace4. `http://www.cs.unm.edu/~mccune/prover9/`, 2005–2010.

[19] Tobias Nipkow, Lawrence C. Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.

[20] Mahamed Omran, Andries Engelbrecht, and Ayed Salman. An overview of clustering methods. *Intell. Data Anal.*, 11:583–605, 11 2007.

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

[22] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.

[23] Geoff Sutcliffe. The CADE-27 automated theorem proving system competition - CASC-27. *AI Commun.*, 32(5-6):373–389, 2019.

[24] Josef Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.