E-MaLeS 1.1

Daniel Kühlwein[1], Stephan Schulz[2], and Josef Urban[1]

[1] Radboud University Nijmegen, Nijmegen, Netherlands*,
[2] Technische Universität München, München, Germany

Abstract. Picking the right search strategy is important for the success of automatic theorem provers. E-MaLeS is a meta-system that uses machine learning and strategy scheduling to optimize the performance of the first-order theorem prover E. E-MaLeS applies a kernel-based learning method to predict the run-time of a strategy on a given problem and dynamically constructs a schedule of multiple promising strategies that are tried in sequence on the problem. This approach has significantly improved the performance of E 1.6, resulting in the second place of E-MaLeS 1.1 in the FOF divisions of CASC-J6 and CASC@Turing.

1 Introduction

Automatic theorem provers (ATPs) for first-order logic search for proofs of a conjecture in a potentially infinite space of derivations. Experience has shown that no single search strategy can be expected to perform well over very diverse proof problems. Thus, most theorem provers provide dozens or even hundreds of parameters. For systems based on modern equational calculi, parameters include clause selection schemes, term orderings, inference and reduction rules used, etc.

The theorem prover E [8] uses a language for describing useful combinations of parameters as *strategies*. Such strategies can be automatically evaluated over large problem sets and compiled directly into C source code. Over 200 strategies have been named and evaluated for E 1.6, and the best of them are included in the E source code.

This large number of strategies with associated performance data suggests the use of data-driven methods to estimate how to solve new problems. E is one of the the first ATPs that have applied machine learning to strategy selection. Below, we first describe how the *automatic mode* of E is generated using simple analogy-based learning. We then introduce an alternative method based on kernel-based learning that automatically determines a schedule of E strategies. The resulting strategy-scheduling meta-system, E-MaLeS 1.1, outperforms the E's original single-strategy automated mode.

2 Learning of Strategy Selection

Characterizing ATP problems is a hard problem. An ambitious formulation could be for example:

* The authors were supported by the NWO projects "MathWiki a Web-based Collaborative Authoring Environment for Formal Proofs" and "Learning2Reason".

Given a set of ATP strategies and a large class of ATP problems problems, find a (typically finite) set of efficiently computable problem features *that can be used to efficiently partition the set of problems into subclasses that share a common best strategy.*

The set of features is in practice suggested by the intuition of the ATP implementer or other domain expert (for example, a mathematician who might know what features could be important for distinguishing different classes of problems in his domain). Some of the features can correspond to precise knowledge about logical calculi, for example, if the problem is Horn, or effectively propositional. Others can express hunches, for example, if there are many or few clauses, symbols, terms, etc. If the set of problems uses symbols consistently (this is the case for recent large-theory corpora created from the Mizar and Isabelle libraries, and for the SUMO and Cyc common-sense ontologies), then the (combinations of) symbols (and derived structures, like terms) are often also relevant for proof search.

This paper presents two learning methods that have been developed to estimate the best strategy for a new problem. Both are based on the results of strategies run on the TPTP problems [12]. The traditional one by Stephan Schulz is an instance of the *case-based reasoning* method. The newer alternative uses *kernel-based* learning. Both methods rely on the TPTP being a sufficiently representative set of ATP problems of different kinds, however the methods can be applied for any sufficiently big corpus of problems (e.g. the MPTP [15]). Both methods require a relevant *feature characterization* of problems and a record of the performance of different strategies for training. Once trained, both provide suggestions for strategies to use on new problems, either a single strategy for the original method, or a schedule of strategies for the newer method.

2.1 E's Feature Characterization

The set of features used in E for problem characterization is listed in Table 1. All features apply to the clausal form of a problem. A clause is called *negative* if it only has negative literals. It is called *positive* if it only has positive literals. A ground clause is a clause that contains no variables. In this setting, we refer to all negative clauses as "goals", and to all other clauses as "axioms". Clauses can be *unit* (having only a single literal), *Horn* (having at most one positive literal), or *general* (no constraints on the form). All unit clauses are Horn, and all Horn clauses are general. Goals have no positive literals and are hence always at least Horn.

2.2 E's automatic mode

E supports an automatic mode that analyzes the problem and determines all major search parameters (literal selection function, clause selection heuristics, term ordering, and a number of mostly discrete options controlling optional simplifications and preprocessing steps). It has been conservatively extended from the very first implementation in E 0.3 *Castleton*, released in 1999.

Feature	Description
`axioms`	Most specific class (unit, Horn, general) describing all axioms
`goals`	Most specific class (unit, Horn) describing all goals
`equality`	Problem has no equational literals, some equational literals, or only equational literals
`non_ground_units`	Number of unit axioms that are not ground
`ground_goals`	Are all goals ground?
`clauses`	Number of clauses
`literals`	Number of literals
`term_cells`	Number of terms (including subterms)
`ground_positive_axioms`	Number of positive axioms that are ground
`max_fun_arity`	Maximal arity of a function or predicate symbol
`avg_fun_arity`	Average arity of symbols in the problem
`sum_fun_arity`	Sum of arities of symbols in the problem
`clause_max_depth`	Maximal clause depth

Table 1. Problem features used by strategy selection in E

The automatic mode is based on a static partitioning of the set of all CNF problems into disjoint classes. It is generated in two steps. First, the set of all training examples (typically the set of all current TPTP problems) is classified into disjoint classes using the features listed in Table 1. For the numeric features, threshold values have originally been selected to split the TPTP into 3 or 4 approximately equal subsets on each feature. Over time, these have been manually adapted using trial and error.

Once the classification is fixed, a Python program reads the different classes and a set of test protocols describing the performance of different strategies on all problems from the test set. It assigns to each class one of the strategies that solves the most examples in this class. For *large* classes (arbitrarily defined as having more than 200 problems), it picks the strategy that also is fastest on that class. For small classes, it picks the globally best strategy among those that solve the maximum number of problems. A class with zero solutions by all strategies is assigned the overall best strategy.

2.3 Strategy scheduling

Strategy learning is currently being used by E's automatic mode to predict the best strategy for a problem. The predicted strategy is then run for the full time.

An alternative approach (known mainly from Gandalf [14], E-SETHEO [11], and Vampire [7]) is strategy scheduling. The idea is to run different strategies for fractions of the overall time. This can help when the strategies are sufficiently orthogonal (solve different problems), and the problem classification is not good enough to clearly point to one best strategy.

In the simplest setting, the suitable set of strategies is considered independent of the problem features. In that case, given a database of results of many strategies on a large set of problems (TPTP, MPTP, etc.), the goal is to cover the set of solvable problems by as few strategies as possible. This is an NP-complete problem, which however seems to be quite easy (for our instances) for systems like MiniSat++ [10]. For example, for an experiment with covering randomly chosen 400 problems from the MPTP2078 benchmark by 280 E strategies, MiniSat++ can find the minimal cover (9 strategies) in 0.09 s.

We present a learning based method that depending on the problem features predicts the time each strategy needs to solve the problem. The predictions are used to schedule in which order and for how long the strategies should be run.

2.4 E-MaLeS 1.1

E-MaLeS (E Machine Learning of Strategies) uses the kernel-based MOR algorithm, see [1] for details. E-MaLeS is freely available at `http://cs.ru.nl/~kuehlwein/`. E-MaLeS learns a function that predicts the performance for each strategy on each problem. Given the features of a problem defined in Table 1, E-MaLeS predicts for each strategy s how long E running s will need to solve the problem. A similar approach has successfully been used in the SAT community [17].

The MOR algorithm is an instance of kernel-based learning. Kernel-based learning is a machine learning approach that finds (typically non-linear in the features) approximations of the training data by minimizing a *loss function* describing the difference between learned approximation and training data. By mapping the data in a higher-dimensional vector space, kernel methods combine the expressiveness of a high-dimensional function space with the simplicity of linear regression. Intuitively, kernels can be seen as a similarity measure between different data points (in our case problems). Kernel-based methods are among the most successful algorithms applied to various problems from bioinformatics to information retrieval to computer vision [9].

E-MaLeS uses a Gaussian kernel. The similarity measure induced by this kernel can be imagined as a Gaussian distribution around each data point with the width σ of the distribution being an adjustable parameter. To ensure that the learned functions generalize well, a regularization parameter λ is used. Regularization adds an additional term based on the complexity of the learned function to the loss function. The more complex a function, the bigger the penalty. The intuition behind this is that complex function are more likely to overfit. The value of λ determines the weight of the regularization term, with $\lambda = 0$ being equivalent to no regularization.

The values for σ and λ are determined via a 10-fold cross-validation. First, we define logarithmically scaled grids of potential values. The training dataset is then shuffled and divided into two parts using a 70/30 split. For each parameter pair the algorithm trains a function based on the data points in the larger part and evaluates it (i.e. compares the predicted run times with the actual run times) on the data points in the smaller part. This process is repeated 10 times. The parameter pair with the best average performance (i.e. the minimum average squared difference between the predicted run times and the actual run times) on the smaller set is then used for the final learning. The goal of cross-validation is to estimate the performance of the learned function on unseen data points.

During the learning phase, the input to E-MaLeS is a list of strategies and a list of problems with their features, together with the performances of the strategies on the problems within a fixed time limit (e.g. 300 seconds). The features are first normalized to values between 0 and 1. Ideally, we would have the exact time needed until a proof is found for each problem-strategy pair. Unfortunately, real world limitations restrict us to finite run times – in our case 300 seconds. Problems that were not solved within this time limit are ignored. Note that this leads to different training data for different strategies and a bias towards lower times. Thus, the learned prediction functions are likely to predict a time that is lower than the actual needed time. An alternative to simply ignoring unsolved problems would be to use a large fixed time for each (e.g. 600 seconds). However, in initial experiments this did not show any improvement.

When trying to solve a new problem, E-MaLeS employs a combination of E's automatic mode and strategy scheduling. First, the automatic mode is run for 60 seconds.[3] If the automatic mode fails to find a proof, the features of the problem are computed and normalized.[4] For each strategy the time needed to solve the problem is predicted using the prediction function learned during the setup. Since E's timeout parameter expects seconds, the predicted times are rounded up to next full second. The strategies are then run for their predicted time, starting with the strategy with the smallest predicted time.[5] If the sum of the rounded predicted times is less than the total time given, the remaining free time is spread equally over all strategies.

3 Results

Both E-MaLeS 1.1 and E 1.6 competed at CASC@Turing and CASC-J6. In both competition, E-MaLeS 1.1 won the second place in the FOF division, solving more problems than E 1.6. The results are shown in Tables 2 and 3

In the FOF division of CASC@Turing, E-MaLeS solved 4.6% more problems than E. If we only compare the results of the new problems, E-MaLeS solved 14.5% more problems. The results for FOF division of CASC-J6 are similar, with

[3] Running the auto mode first allows us to reduce the number of training examples of the machine learning algorithm. We only learn from problems that cannot be solved by the automatic mode within 60 seconds. The reason for this is that the learning time of the algorithm is cubic in the number of training examples which makes learning from all examples (the whole TPTP library) infeasible.

[4] Since the normalization function is defined during setup, the normalized features of new problems may fall out of the $[0, 1]$ interval.

[5] A possible improvement would be to take not only the predicted times, but also the orthogonality (difference in problems solved between strategies) of strategies into account. First experiments with this approach were promising.

System	All Problems	New Problems
E 1.6	378/500	73/97
E-MaLeS 1.1	401/500	87/97

Table 2. Results for the CASC@Turing problems

System	All Problems	New Problems
E 1.6	359/450	50/68
E-MaLeS 1.1	377/450	59/68

Table 3. Results for the CASC-J6 problems

E-MaLeS solving 4% more problems than E. On the new problems, E-MaLeS solved 13.2% more problems than E.

A possible explanation of the discrepancy between old and new problems solved is that E 1.6 is overspecialized for the old problems. Cross-validation and regularization helps E-MaLeS to combat such overfitting.

E-MaLeS also competed in the LTB (Large Theory Batch) division of CASC-J6 and placed fourth with 83 problems solved after E 1.6 with 87 solved problems. The LTB division contains problems from large theories, namely problems exported from Isabelle, Mizar and SUMO. Unlike in the FOF division, the ATPs may use all cores of the machine. The LTB version of E-MaLeS uses the extra time and all available cores to run more strategies but is in no other way optimized. E 1.6, on the other hand, makes use of the batch structure to avoid repeated parsing of the large background theories, and also makes better use of the various SInE strategies that heuristically select relevant premises from the large theories.

4 Future Work

There are several ways to improve the current algorithm. Extracting features based on the FOF instead of the CNF representation of a formula could lead to better learning performance. Using different learning algorithms might allow us to run E-MaLeS without relying on E's automatic mode.

Strategies themselves are just combinations of many ATP parameters. An interesting application of machine learning to ATP is to develop new strategies by searching for such good parameter combinations systematically (by methods like hill-climbing) on a large corpus of problems. A related task is to produce a set of strategies that are highly orthogonal, i.e., that solve very different problems, so that their collective coverage is high.

We could also learn strategies based on conjecture features (like symbols) in consistently-named corpora like MPTP, instead of just using abstract features like on the TPTP problems.

The methods that we develop can probably be used for any ATP, and it would be interesting to see how such learning and optimization works for systems like

Vampire [7], Z3 [6], and iProver [5]. Meta-systems like Isabelle/Sledgehammer [4,3] and MaLARea [16] could then use this deeper knowledge about ATPs to attack new conjectures with the strongest possible combinations of systems and strategies.

References

1. Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze & Josef Urban (2011): *Premise Selection for Mathematics by Corpus Analysis and Kernel Methods*. *CoRR* abs/1108.3446. Accepted to JAR.
2. Alessandro Armando, Peter Baumgartner & Gilles Dowek, editors (2008): *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*. *LNCS* 5195, Springer.
3. Jasmin Christian Blanchette, Sascha Böhme & Lawrence C. Paulson (2011): *Extending Sledgehammer with SMT Solvers*. In Nikolaj Bjørner & Viorica Sofronie-Stokkermans, editors: *CADE*, *LNCS* 6803, Springer, pp. 116–130.
4. Jasmin Christian Blanchette, Lukas Bulwahn & Tobias Nipkow (2011): *Automatic Proof and Disproof in Isabelle/HOL*. In Cesare Tinelli & Viorica Sofronie-Stokkermans, editors: *FroCoS*, *LNCS* 6989, Springer, pp. 12–27.
5. Konstantin Korovin (2008): *iProver - An Instantiation-Based Theorem Prover for First-Order Logic (System Description)*. In Armando et al. [2], pp. 292–298.
6. Leonardo Mendonça de Moura & Nikolaj Bjørner (2008): *Z3: An Efficient SMT Solver*. In C. R. Ramakrishnan & Jakob Rehof, editors: *TACAS*, *LNCS* 4963, Springer, pp. 337–340.
7. Alexandre Riazanov & Andrei Voronkov (2002): *The design and implementation of VAMPIRE*. *AI Commun.* 15(2-3), pp. 91–110.
8. Stephan Schulz (2002): *E - A Brainiac Theorem Prover*. *AI Commun.* 15(2-3), pp. 111–126.
9. John Shawe-Taylor & Nello Cristianini (2004): *Kernel Methods for Pattern Analysis*. Cambridge University Press, New York, NY, USA.
10. Niklas Sörensson & Niklas Eén (2008): *MiniSat 2.1 and MiniSat++ 1.0 SAT Race 2008 Editions*. Technical Report, Chalmers University of Technology, Sweden.
11. G. Stenz & A. Wolf (2000): *E-SETHEO: An Automated^3 Theorem Prover – System Abstract*. In R. Dyckhoff, editor: *Proc. of the TABLEAUX'2000*, *LNAI* 1847, Springer, pp. 436–440.
12. G. Sutcliffe (2009): *The TPTP Problem Library and Associated Infrastructure: The FOF and CNF Parts, v3.5.0*. *Journal of Automated Reasoning* 43(4), pp. 337–362.
13. Geoff Sutcliffe, Stephan Schulz, Koen Claessen & Allen Van Gelder (2006): *Using the TPTP Language for Writing Derivations and Finite Interpretations* . In Ulrich Fuhrbach & Natarajan Shankar, editors: *Proc. of the 3rd IJCAR, Seattle*, *LNAI* 4130, Springer, 4130, pp. 67–81.
14. Tanel Tammet (1997): *Gandalf*. *Journal of Automated Reasoning* 18, pp. 199–204.
15. Josef Urban (2006): *MPTP 0.2: Design, Implementation, and Initial Experiments*. *J. Autom. Reasoning* 37(1-2), pp. 21–43.
16. Josef Urban, Geoff Sutcliffe, Petr Pudlák & Jirí Vyskocil (2008): *MaLARea SG1-Machine Learner for Automated Reasoning with Semantic Guidance*. In Armando et al. [2], pp. 441–456.
17. Lin Xu, Frank Hutter, Holger H. Hoos & Kevin Leyton-Brown (2008): *SATzilla: portfolio-based algorithm selection for SAT*. *J. Artif. Int. Res.* 32(1), pp. 565–606.