MaLeCoP
Machine Learning Connection Prover

Josef Urban[1,⋆], Jiří Vyskočil[2,⋆⋆], and Petr Štěpánek[3,⋆⋆⋆]

[1] Radboud University Nijmegen, The Netherlands
[2] Czech Technical University
[3] Charles University, Czech Republic

Abstract. Probabilistic guidance based on learned knowledge is added to the connection tableau calculus and implemented on top of the leanCoP theorem prover, linking it to an external advisor system. In the typical mathematical setting of solving many problems in a large complex theory, learning from successful solutions is then used for guiding theorem proving attempts in the spirit of the MaLARea system. While in MaLARea learning-based axiom selection is done outside unmodified theorem provers, in MaLeCoP the learning-based selection is done inside the prover, and the interaction between learning of knowledge and its application can be much finer. This brings interesting possibilities for further construction and training of self-learning AI mathematical experts on large mathematical libraries, some of which are discussed. The initial implementation is evaluated on the MPTP Challenge large theory benchmark.

1 Introduction

This paper describes addition of machine learning and probabilistic guidance to connection tableau calculus, and the initial implementation in the leanCoP system using an interface to an external advice system. The paper is organized as follows:[1] Section 1 describes the recent developments in large-theory automated reasoning and motivation for the research described here. Section 2 describes the machine learning (data-driven) paradigm and its use in guiding automated reasoning. Section 3 shortly summarizes the existing leanCoP theorem prover based on connection tableaux. Section 4 explains the general architecture for combining external machine learning guidance with a tableau prover. Section 5 describes our experimental implementation. Section 6 describes some experiments done with the initial implementation. Section 7 concludes and discusses future work and extensions.

⋆ Supported by The Netherlands Organization for Scientific Research (NWO) grants *Learning2Reason* and *MathWiki*.
⋆⋆ Supported by the Czech institutional grant MSM 6840770038.
⋆⋆⋆ Supported by the Grant Agency of Charles University, grant 9828/2009.
[1] We would like to thank the anonymous referees for helping to significantly improve the presentation of this work.

1.1 Large-Theory Automated Reasoning

In the recent years, increasing amount of mathematics and knowledge in general is being expressed formally, in computer-understandable and computer-processable form. Large formal libraries of re-usable knowledge are built with interactive proof assistants, like Mizar, Isabelle, Coq, and HOL (Light). For example, the large Mizar Mathematical Library (MML) contains now (February 2011) over 1100 formal articles from various fields, covering substantial part of undergraduate mathematics. At the same time, the use of the formal approach is also increasing in non-mathematical fields, for example in software and hardware verification and in common-sense reasoning about real-world knowledge. This again leads to growth of formal knowledge bases in these fields.

Large formal theories are a recent challenge for the field of automated reasoning. The ability of ATP systems to reason inside large theories has started to improve after 2005, when first-order ATP translations of the particular formalisms used e.g. by Mizar [16], Isabelle [5], SUMO, and Cyc started to appear, and large theory benchmarks and competitions like MPTP Challenge and CASC LTB were introduced. The automated reasoning techniques developed so far for large theories can be broadly divided into two categories:

1. Techniques based purely on heuristic symbolic analysis of formulas available in problems.
2. Techniques taking into account also previous proofs.

The SInE preprocessor by Kryštof Hoder [4,17] seems to be so far the most successful heuristic in the first category. In domains like common-sense reasoning that typically lack large number of previous nontrivial and verified proofs and lemmas, and mostly consist of hierarchic definitions, such heuristics can sometimes even provide complete strategies for these domains.[2] MaLARea [18] is an example of a system from the second category. It is strong in hard mathematical domains, where the knowledge bases contain much less definitions than nontrivial lemmas and theorems, and previous verified proofs can be used for learning proof guidance. This approach is described in the next section, giving motivation for the work described in this paper.

2 Machine Learning in Large Theory ATP

The data-driven [12] approaches to constructing algorithms have been recently successful in AI domains like web search, consumer choice prediction, autonomous vehicle control, and chess. In contrast to purely theory-driven approaches, when whole algorithms are constructed explicitly by humans, the data-driven approaches rely on deriving substantial parts of algorithms from large amounts of data. In the ATP domain, the use of machine learning started to be explored by the Munich

[2] For example, a Prolog-based premise selection preprocessor was used by Vampire in the CYC category of the 2008 CASC LTB competition to solve all problems.

group [3]. The E prover [11] by Stephan Schulz contains several hooks where algorithms can be optimized based on machine learning. The most advanced technique there being probably matching of abstracted previous proof traces for guiding the inference (given clause loop) process. Simpler techniques like optimization of strategy selection or scheduling are used not only by E prover, but also, e.g., by the Vampire system [10]. In 2007 the Machine Learner for Automated Reasoning (MaLARea [18]) started to be developed, triggered by the translation of the Mizar library to first-order ATP format and the need to provide efficient reasoning over the ca 50000 theorems and definitions in it. We explain here the basic idea of learning from proofs, which is in several modified forms used also in MaLeCoP.

The basic idea of the machine learning approach is to learn an association from features (in the machine learning terminology) of the conjectures (or even of the whole problems when speaking generally) to proving methods that are successful when the features are present. In MaLARea, this general setting is instantiated in the following way: The features characterizing a conjecture are the symbols appearing in them, and the proving method is an ordering of all the axioms according to their expected relevance to proving the conjecture. One might think of this as the particular set of symbols determining a particular sublanguage (and thus also a subtheory) of a large theory, and the corresponding ordering of all the available axioms as, e.g., a frequency of their usage in a book written about that particular subtheory. Once a sufficient body of proofs is known, a machine learning system (SNoW [2] is used by MaLARea in naive Bayes mode) is trained on them, linking conjecture symbols with the axioms that were useful for proving the conjectures. For learning and evaluation in SNoW, all symbols and axiom names are disjointly translated to integers. The integers corresponding to symbols are the input features, and those corresponding to axioms are the output features of the learning process.

MaLARea can work with arbitrary ATP backends (E and SPASS by default), however, the communication between learning and the ATP systems is high-level: The learned relevance is used to try to solve problems with varied limited numbers of the most relevant axioms. Successful runs provide additional data for learning (useful for solving related problems), while unsuccessful runs can yield countermodels, which can be in MaLARea re-used for semantic pre-selection and as additional input features for learning.

An advantage of the high-level approach is that it gives a generic inductive (learning)/deductive (ATP) metasystem to which any ATP can be easily plugged as a blackbox. Its disadvantage is that it does not attempt to use the learned knowledge for guiding the ATP search process once the axioms are selected. Hence the logical next step described in this paper: We try to suggest how to use the learned knowledge for guiding proof search *inside* a theorem prover. We choose the leanCoP theorem prover for the experiments, both because of its simplicity and easiness of modification, and for a number of interesting properties described below that make it suitable for interaction with learning. The next section summarizes leanCoP.

3 leanCoP: Lean Connection-Based Theorem Prover

3.1 Why leanCoP

leanCoP is an economically written connection-based theorem prover. The main theorem prover can be written on a couple of lines in Prolog, while its performance is surprisingly good, especially when goal-directness is important. The reasons for choosing leanCoP for our experiments can be summarized in the following points:

– leanCoP already has good performance on the MPTP Challenge benchmark [7]. This guarantees sufficient amount of proofs to learn from.
– The implementation is simple, high-level, and Prolog-based, making it easy to experiment with. Our experience with modifying C-written ATPs (even very nicely written like the E prover as in [15]) is that it always involves a lot of low-level implementation work.
– The tableau calculus seems to be quite suitable for the kind of additions that we want to experiment with. It has a transparent notion of proof state (branch that needs to be closed, open goals) to which advising operations can be applied. This contrasts with resolution ATPs that have just several large piles of clauses describing the proof state.
– The integration of learning and ranking and its use can be very tight, allowing implementation of techniques similar to just-in-time compilation. This means keeping track of frequent requests – especially in the many-problems/large-theory setting – and providing (possibly asynchronous) advice for them.
– We hope that the simple Prolog setting should allow easy additional experiments. This could include simple addition of other kinds of external advisors (e.g. for computer algebra), experiments with online learning from closed branches, and experiments with probabilistic finding of decision procedures (expressed just as sets of Prolog clauses), with the possibility of Prolog techniques for program transformation of the found algorithms.

3.2 The Basic leanCoP Procedure and its Parametrization

For further understanding, it is good to summarize the main features of leanCoP [6,8]. leanCoP is an automated theorem prover for classical first-order logic with equality. It uses an optimized structure-preserving transformation into clausal form (DNF, see also below), to which connected tableau search (with iterative deepening to guarantee completeness) is then applied. The reduction rule of the connection calculus is applied before the extension rule, and open branches are selected in a depth-first way. Additional inference rules and strategies are regularity, lemmata, and restricted backtracking. leanCoP has several parameters influencing its work [8][3], which can also be used for defining various strategies and scheduling over them:

– if the option *def* is used, new definitions are introduced systematically during the clausification to shorten the resulting clause set

[3] This description is also relevant for leanCoP 2.1.

- if the option *nodef* is used, no new definitions are introduced in clausification
- if none of the options *def, nodef* is used, and the formula has the form $X \to Y$, then X is processed as in the *nodef* case, while Y as in the *def* case
- if option *reo(N)* is used, the set of clauses is shuffled using N as a parameter
- if option *cut* is used, there is no backtracking after successfully closed branches
- if option *scut* is used, backtracking is restricted for alternative start clauses
- if option *conj* is used, a special literal is added to the conjecture clauses in order to mark them as the only possible start clauses.

4 The General Architecture

As mentioned in the introduction, our goal is to experiment with smart (external) proof search guidance used inside theorem provers' internal mechanisms, not just outside them for premise pruning as MaLARea does. Our goal is an AI architecture that is closer to human thinking in that it does not blindly try every deductive possibility, but chooses the best course of action based on its knowledge of the world and previous experiences. The architecture should be able to learn both from successes and from mistakes, and update its decision mechanisms when such new knowledge is available. We also want our architecture to be not just a theoretical toy, but a system that actually proves theorems, and does that (at least often) more efficiently than the unguided systems thanks to the smart guidance. This set of requirements seems to be quite an ambitious program, below we explain our general approach, problems encountered, and the initial implementation.

4.1 The Concerns

The particular task that we initially consider is advising clause selection in the extension steps of the tableau proving process. The obvious intuition that this is the core source of possible speedups (once smart guidance is provided) is demonstrated below in the Evaluation section.

Several concerns like speed, generality, and extendability influence the general design. Measurements show that leanCoP can do an order of several hundred thousands basic inferences (extension steps) per minute on recent hardware.[4] In large mathematical theories, typically thousands of different symbols appear, and thousands of theorems are available for proving. If a smart (external) mechanism for formula/clause selection in such large theories took a minute for its recommendation, and we were using the mechanism for each inference, the inference speed would drop from hundreds of thousands to one per minute. With sufficiently "complicated AI" implementations of the external advice such (and much higher) times are conceivable. Even worse, one can still argue that it might be the right thing to do when solving hard problems, because the raw inference

[4] All measurements are done on the server of the Foundations group at Radboud University Nijmegen (RU), which is eight-core Intel Xeon E5520 2.27GHz with 8GB RAM and 8MB CPU cache.

speed matters very little when we traverse superexponential search space. Obviously, doing experimental research in such setting would be very costly. That's why we want to have reasonable speed of the guiding mechanism for experiments, and possibly use it only in the most critical choices.

Several options can be considered for implementing the guidance mechanism:

1. Using a raw external learning/advising system like SNoW in MaLARea, via socket communication with the theorem proving process.
2. Implementing the learning/advising system directly as a part of the prover.
3. Compiling/linking an external system directly with the theorem prover's binary (to avoid communication overhead).
4. Using an interface layer (directly in the prover or as a separate tool) that talks to the external tools, organizes their work, and talks to the prover.
5. Combinations of above.

Generality and extendability requirements tell us to avoid the second option, at least in the experimental phase, because we want to be able to easily plug in different external advice systems. For example, the SNoW system itself provides several learning mechanisms (winnow, perceptron, naive bayes) and a number of options to them. Kernel-based learning has been also recently experimented with in the context of premise selection [14], improving the guidance precision. The general design suggested below and instantiated in our prototype[5] uses the fourth option from the above list.

4.2 The Design

The general design that we propose is as follows (see also Figure 1): The theorem prover (P) should have a sufficiently fast communication channel to a general advisor (A) that accepts queries (proof state descriptions) and training data (characterization of the proof state[6] together with solutions[7] and failures) from the prover, processes them, and replies to the prover (advising, e.g., which clauses to choose). The advisor A also talks to external system(s) (E). A translates the queries and information produced by P to the formalism used by a particular E, and translates E's guidance back to the formalism used by P. At suitable time, A also hands over the (suitably transformed) training data to E, so that E can update its knowledge of the world on which its advice is based. A is free to spawn/query as many instances/versions of Es as necessary, and A is responsible for managing the guidance provided by them. Particular instances of Es that we have in mind are learning systems, however we believe that the tableau setting is also suitable for linking of SMT solvers, computer algebra systems, and all kinds of other AI systems, probably in a more straightforward way than for the resolution-based systems [13,9].

[5] `http://mws.cs.ru.nl/~urban/malecop/`

[6] Instantiated, e.g., as the set of literals/symbols on the current branch.

[7] Instantiated, e.g., as the description of clauses used at particular proof states.

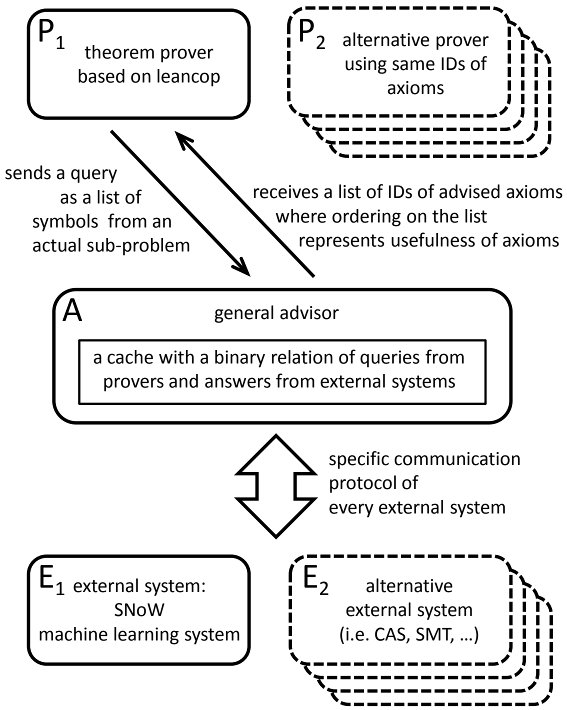
P1 theorem prover based on leancop
P2 alternative prover using same IDs of axioms
sends a query as a list of symbols from an actual sub-problem
receives a list of IDs of advised axioms where ordering on the list represents usefulness of axioms
A general advisor
a cache with a binary relation of queries from provers and answers from external systems
specific communication protocol of every external system
E1 external system: SNoW machine learning system
E2 alternative external system (i.e. CAS, SMT, ...)

Fig. 1. The General Architecture

5 MaLeCoP: Machine Learning Connection Prover

The above general ideas have been initially implemented by using a modified version of leanCoP as P, the SNoW machine learner/advisor as E, and writing a general managing library in Perl for A. These components and their linking are explained below.

5.1 The Theorem Prover

The theorem prover we choose is leanCoP (version 2.1), however a number of modifications have to be made so that it fits into the general architecture. Here we mention some of them.

Consistent Clausification. The original leanCoP interleaves various clausification strategies with the tableau search. This means that sometimes significant changes can be done by changing the clausification (see, e.g., the options *def, nodef, reo* in Section 3). Changing the clausification however means that different skolem symbols and new definitions are produced, and clauses have different names and shapes. Additionally, the original leanCoP works in a one-problem-at-a-time setting, always inventing new names for skolem symbols and new definitions for each problem, despite the fact that the formulas/clauses are shared among many problems. This would not work in our setting, because the guidance is trained on symbol names and clause names. In other words, it is necessary to change leanCoP in such a way so that its various settings always work with the same clause normal form, and so that the new symbols are introduced consistently (with the same name) in all problems that share a particular formula. This is done by splitting the work of leanCoP on a large number of problems into two phases:

1. Common clausification done for all problems together, with consistent introduction of new symbols.
2. The separate tableau search, starting already with the clausified problem.

The 252 large MPTP Challenge problems contain 1485 unique formulas with a lot of repetition across the problems (this is the large consistent theory aspect), yielding 102331 (nonunique) formulas in all 252 problems all together. These 1485 formulas are consistently clausified in the first phase to 6969 unique clauses, appearing all together 396927 times in the jointly clausified problems, and containing 2171 unique symbols. This level of consistency and sharing on the symbol and clause level should ensure good transfer of knowledge during the learning and advice phases. As a byproduct of the clausification we also produce a mapping of clauses to symbols contained in them, a mapping of clauses to hashes of all terms contained in them, and a listing of all symbol names and clause names. These are later used by the general advisor (A) and external system(s) E.[8]

Strategies for Guidance. While we try to make the access to the advice as fast as possible (by caching in the advisor, etc.), it turns out that on average it takes about 0.2 second to the SNoW system to produce advice. This is certainly a bottleneck we need to work on, on the other hand, evaluation of a Bayes net on nearly 7000 targets, and their sorting according to the activation weight might really justify these times. In that case further work on faster external systems will be needed. As it is now, one external advice costs an order of one thousand leanCoP inferences. That is why we need to define strategies trading in a reasonable way the external advice for internal inferencing. The current set of strategies is as follows:

1. `original_leancop`: This mode works exactly as the original core leanCoP prover, however using the consistent clausification (affecting some options) as described above.
2. `naive`: From all the literals on the current branch symbols are extracted, and sent to the advisor. The advisor replies with an ordered list of recommended clauses for the current inference. If none of these clauses succeeds, the conjecture clauses are tried. This mode can obviously be incomplete if

[8] The term hashes are not used yet for learning/advice, and we also do not provide semantic (model) features as in MaLARea SG1 [18].

the advice is bad and a non-conjecture clause is needed. It is also demands advice for every inference, making it currently very slow.

3. `naive_and_complete`: As `naive`, but if no advised clause succeeds, all remaining clauses from the problem are tried in their original order in the problem. This makes this strategy complete, yet still very slow.
4. `full_caching_and_complete`: Produces same results as `naive_and_complete`, however it applies internal caching in Prolog to cut down the number of slow external queries. For every external query the advised clauses are cached in the advised order in a new clause database, and if the query is repeated this clause database is used instead of asking the external advisor again. This obviously increases the memory consumption of the Prolog process.
5. `smart_caching_and_complete`: Works in the same way as `naive_and_complete`, using a similar method as `full_caching_and_complete`. However new clauses databases are created only for the advised clauses (not for all clauses as in `full_caching_and_complete`). If the new database is not successful, then the original leanCoP database is used.
6. `smart_caching`: Works as `smart_caching_and_complete`, however it caches only the advised clauses. The original clause database is never used, making this strategy incomplete.
7. `original_leancop_with_first_advice`: At the start of the proof search one query is made to the advisor sending all conjecture symbols. A new Prolog database is created containing the advised clauses in the advised order, followed by the rest of the clauses in the original order. This database is then used in the rest of proof search without further queries to the advisor. This is obviously very fast and complete, but the use of external advice is very limited.
8. `leancop_ala_malarea`: As `original_leancop_with_first_advice`, however only the advised clauses and conjecture clauses are asserted into the new database. This results in an incomplete search limited in the same way as in MaLARea.
9. `limited_smart_with_first_advice(Depth)`: This uses `smart_caching` until the Depth in tableau is reached, then it proceeds as `original_leancop_with_first_advice`. The first stages thus cause incompleteness. External queries are limited to Depth, which allows us to be flexible with trading speed for precision.
10. `limited_smart_and_complete_with_first_advice(Depth)`: Uses `smart_caching_and_complete` until Depth, then again `original_leancop_with_first_advice`.
11. `scalable_with_first_advice(Limit,Mode_After_Limit)`: This is a metastrategy. Its basis is `original_leancop_with_first_advice`. If the clause search does not succeed for Limit-th clause (in the original database) of a particular literal (extension step), the strategy `Mode_After_Limit` is used for the next clause search. This again is used to limit the uses of the (typically expensive) `Mode_After_Limit` strategy to “justified cases”, when the original proof search seems to be bad.

12. `scalable_with_first_advice(Limit,Query_Limit,Mode_After_Limit)`: This is an extension of the previous metastrategy. When the Limit-th clause search fails, the branching factor is computed (by counting all clauses to which a connection can be made at the point). If it is greater than Query_Limit, the strategy `Mode_After_Limit` is used. Otherwise the original database (`original_leancop_with_first_advice`) is used. This strategy provides even finer language for limiting the number of external queries to the most important branchings.

Learning and Other Options. The basic learning in MaLARea is used to associate conjecture symbols with premises used in the conjecture's proof.[9] This learning mode corresponds well to the `original_leancop_with_first_advice` strategy above, and various metastrategies re-using it. For learning clause selection on branches we can further use another information supplied by the prover: successful clause choices done for particular paths in the proof. If the `MACHINE_LEARNING_OF_SUBTREES` option is set, the prover generates for each proved subtree a list of symbols present on the current path (input features for learning), together with the successfully used clause (output feature for learning). However, the notion of "success" is relative: a success in a subtree does not imply a success in the whole proof. That is why we only collect this information from successful proofs, after re-running them just with the necessary clauses. This should avoid learning from any "pseudo successes". The information extracted from subtrees also contains the cost (again in terms of inference numbers) of finishing the subtree. We do not use this information yet in learning, however we plan to use learning on this data for gradually overcoming the most costly bad clause choices. This could be used to attack hard unsolved problems, by interleaving the learning to avoid such traps with re-newed proof attempts.

The original options *cut, scut, comp(L), reo(N), conj* work (after possible re-implementation) also in the modified leanCoP, i.e., it still makes sense to pass them to the clausal prover. Options *def, nodef* however have to be fixed during the clausification stage, and cannot be changed later.

5.2 The General Advisor and the External System

The general advisor is a simple layer of functions written in Perl providing communication with leanCoP via a TCP socket, and talking to SNoW either via a TCP socket or via a UNIX FIFO (named pipe). The advisor takes queries in the forms of symbol lists from the prover, and translates the symbols into the numeric representation used by SNoW, feeding it to SNoW afterward and translating the SNoW output (again a series of numbers) back into the clause names. The ordered list of clause names is then handed over to leanCoP. The advisor starts by loading the symbol and clause tables produced by the clausification phase. The library obviously also implements management of the training examples produced by the proof runs, it manages the prover runs and the external system's

[9] In machine learning terminology, the conjecture symbols are the input features, and the premises are the output features (targets).

training and querying. In addition to that, the advisor also implements a simple fast cache of queries (just a Perl hash) which considerably speeds up the external advice on previously seen queries. While the SNoW system averages to about five answered queries per second, the advisor cache can answer more than one thousand queries per second, which makes it comparable to the leanCoP inference speed. Thus a sufficiently big pre-computed cache (typically from many problems) speeds up proof attempts relying on external advice considerably. The cache size for the experiments described below – which issued ca hundred thousand queries to SNoW in total – is about 500MB, which is quite manageable.

As already mentioned SNoW is used as the external learning/advice system, to which the prover talks via the advisor. The biggest concern is its raw speed in the advice mode. The overall SNoW CPU time used for about 120 thousand queries is seven hours. Future work could include experiments with using several SNoW instances using just a limited number of targets relevant for each problem, and improvement of SNoW's speed when evaluating a large number of targets.

6 Evaluation

6.1 Dataset

The evaluation is done on the MPTP Challenge[10] data (specifically on its harder – Chainy – division), and does not use the CASC LTB[11] data. This (together with our machine learning systems not competing in recent CASCs) has recently raised questions. The explanation follows.

The MPTP Challenge is a benchmark specifically created to allow comparison of learning and non-learning ATP approaches. While still reasonably small for experiments, it seems to provide sufficient amount of data for testing the data-driven approaches to ATP. Although the MPTP Challenge design (by the first author) was borrowed by the first CASC LTB in 2008, the main motivation, i.e., providing suitable benchmark for both learning and non-learning ATP methods, has been practically abandoned by CASC LTB in 2009. The number of problems in CASC LTB (and specifically hard problems solvable only by learning) and the learning options have been reduced, and the original "AI" competition mechanism was largely changed back towards the old-style one-problem-at-a-time CASC rules. In short, machine learning makes little sense when done only on a few examples, which is what CASC LTB currently allows. To help to remedy this, in addition to the MPTP Challenge, several mathematical large-theory benchmarks have been recently defined in [17] and used for ATP evaluation in real mathematical setting.

6.2 Results in Proof Shortening

The first test conducted is evaluation of the learning's ability to guide the search for proofs in problems that were already solved. This is a sanity check telling us

[10] `http://www.tptp.org/MPTPChallenge/`

[11] The CADE ATP System Competition Large Theory Batch division.

Table 1. Comparison of number of inferences for the 73 problems solved by original leanCoP, and by leanCoP using guidance trained on the 73 solutions

problem	orig. inferences	guided inferences	problem	orig. inferences	guided inferences
t69_enumset1	676	177	t12_xboole_1	314	291
t13_finset_1	397	99	t17_xboole_1	263	81
t15_finset_1	16	26	t19_xboole_1	1533	757
l82_funct_1	748	1106	t1_xboole_1	305	225
t35_funct_1	813	148	t26_xboole_1	55723	18209
t70_funct_1	1631	669	t28_xboole_1	320	327
t8_funct_1	388	664	t2_xboole_1	22	16
t7_mcart_1	15863	39	t36_xboole_1	477	113
t10_ordinal1	42729	645	t37_xboole_1	27	63
t12_pre_topc	29	26	t39_xboole_1	12452	68164
t116_relat_1	6751	162	t3_xboole_1	35	78
t117_relat_1	14191	2588	t45_xboole_1	3434	520
t118_relat_1	516	293	t48_xboole_1	108205	3863
t119_relat_1	32721	1431	t60_xboole_1	131	96
t144_relat_1	117908	1577	t63_xboole_1	2733	479
t146_relat_1	33580	1370	t7_xboole_1	211	89
t167_relat_1	156202	1629	t83_xboole_1	1885	326
t20_relat_1	1359	405	t8_xboole_1	4018	2612
t30_relat_1	754	583	t44_yellow_0	1533	989
t56_relat_1	3793	181	t6_yellow_0	3605	138
t60_relat_1	6251	148	l1_zfmisc_1	22281	233
t64_relat_1	43674	1491	l23_zfmisc_1	230	126
t88_relat_1	10285	1749	l25_zfmisc_1	4495	799
t90_relat_1	27169	875	l28_zfmisc_1	59233	6095
t99_relat_1	478	124	l50_zfmisc_1	3182	200
t16_relset_1	1931	130	t106_zfmisc_1	92	131
l3_subset_1	12295	5052	t10_zfmisc_1	2055	2115
t50_subset_1	46702	2071	t119_zfmisc_1	16954	199
t54_subset_1	1064	217	t1_zfmisc_1	13471	843
l1_wellord1	29925	4580	t37_zfmisc_1	46	63
l29_wellord1	1059	180	t39_zfmisc_1	45	116
t20_wellord1	60844	1821	t46_zfmisc_1	17	26
t32_wellord1	35573	3607	t65_zfmisc_1	23503	1966
t7_wellord2	107	63	t6_zfmisc_1	1650	112
t3_xboole_0	696	609	t8_zfmisc_1	71884	1321
t4_xboole_0	47	150	t92_zfmisc_1	19	26
l32_xboole_1	19088	589			
Averages:	15678	2042	Avrg. ratio:	19.80	

how much is the overall architecture working as expected. We want to know if it provides the right advice on the problems that it has already seen and been trained on.

The evaluation is done as follows. The original leanCoP (the `original_leancop` strategy, see 5.1) is run with 20s timelimit on the 252 large problems from the MPTP Challenge, solving 73 of them. The guidance system is then trained on the 73 proofs, together with 630 path/clause choices corresponding to the proofs,[12] 703

[12] See the option `MACHINE_LEARNING_OF_SUBTREES` described in 5.1.

training examples in total. Then we try to solve the 73 problems again, this time using the trained guidance in the `limited_smart_with_first_advice(4)` mode (see above for detailed description). Again, 20s timelimit (excluding the guidance) is used, and all 73 problems are solved with the guidance. For comparison of the proof search we use the number of extension steps done by leanCoP. This seems to be a suitable metric which abstracts from the communication overhead and the overhead for running the guidance system(s). The following Table 1 shows the results of this comparison. The guidance helps in this case to shorten the proof search on average by a factor of 20, and in some cases (t167_relat_1) by nearly a factor of 100. This seems to be sufficiently convincing as a sanity check for the guidance architecture. Several other strategies were evaluated in this mode too, however we do not show their results here for lack of space.

6.3 Solving New Problems

The main test of any learning system is its ability to generalize over the provided data, and give good advice for new queries. This is measured by attempting to solve the remaining MPTP Challenge problems by using the guidance system trained on the 73 problems solved by original leanCoP. We do not (yet) iterate the learning as in MaLARea, and just evaluate the performance and behavior of the overall architecture with various settings using the initially trained guidance.

As mentioned above, some settings now require a lot of CPU time from the trained advisor, so we only evaluate seven interesting advanced strategies that make the experiments feasible in a couple of hours. The seven strategies (numbered as follows) solve all together 15 problems unsolved in the first run (each

Table 2. Comparison of number of inferences for the 15 problems solved all together by the seven strategies (empty entries were not solved within the time limit)

problem	1	2	3	4	5	6	7
t26_finset_1						4033	
t72_funct_1	1310						
t143_relat_1		28458	59302				61660
t166_relat_1		17586			4067	5263	
t65_relat_1					79756	36217	
t43_subset_1		82610					
t16_wellord1					37148		
t18_wellord1			3517	2689			2524
t33_xboole_1	3659	16456				16902	17925
t40_xboole_1		16488			15702		28404
t30_yellow_0	24277						
l2_zfmisc_1							85086
l3_zfmisc_1							79786
l4_zfmisc_1		17074			9584	14299	30273
t9_zfmisc_1			80684				77532

again uses the 20s timelimit, excluding the guidance time). The comparison of the successes and inference numbers are shown in Table 2.

1. leancop_ala_malarea
2. limited_smart_with_first_advice(3)
3. scalable_with_first_advice(40,limited_smart_and_complete_with_first_advice(3))
4. scalable_with_first_advice(3,20,original_leancop)
5. limited_smart_with_first_advice(4)
6. scalable_with_first_advice(3,20,limited_smart_with_first_advice(5))
7. limited_smart_and_complete_with_first_advice(3)

7 Discussion, Future Work

A number of future directions are already discussed above. The slow external advice is currently a clear bottleneck, necessitating further tuning of strategies that can use the advice only in critical places. Combination of complete and incomplete strategies is an interesting topic for research, and when looking at Table 2, there does not seem to be any clear choice. Learning has been so far done only on symbols extracted from the clauses, while in MaLARea the term structure and (counter) models are used too. This is probably a straightforward addition, which will however again raise the number of features used by SNoW, possibly making SNoW even slower. We have so far not run the full inductive/deductive loop as in MaLARea, which will be enriched by the training data extracted from successful subtrees. An interesting addition is also gradual learning of important choices from unsuccessful proof attempts, which could lead to quite intelligently behaving proving systems. Another option is learning of sequences of clauses that lead to success for particular classes of inputs. Such sequences are sufficient for defining algorithms in Prolog, and if it is possible to detect terminating behavior, they could be called decision procedures. A nice feature of such futuristic scenarios is that the input classes together with the algorithms defined for them could be tested for theoremhood, just by adding them as new conjectures to the whole large theory we work in. Such data-driven methods might produce a large number of heuristics that are easier to acquire automatically on a large number of problems than both the existing manual research in finding of suitable simplification orderings for small domains, and research in manual crafting of decision procedures for particular classes of problems and adding them to ATPs.

A probably much simpler (but less “AI”) way how to add decision procedures in our setting is just by querying external computer algebra systems and other solvers for the literals on the current path. As mentioned above, the tableau setting seems to be quite suitable for such extensions, and probably more suitable than the resolution setting. Quite surprisingly, it seems that this is the first time a tableau system is being linked to external advice mechanisms.

References

1. Armando, A., Baumgartner, P., Dowek, G. (eds.): IJCAR 2008. LNCS (LNAI), vol. 5195. Springer, Heidelberg (2008)
2. Carlson, A., Cumby, C., Rosen, J., Roth, D.: SNoW User's Guide. Technical Report UIUC-DCS-R-99-210, University of Illinois at Urbana-Champaign (1999)
3. Denzinger, J., Fuchs, M., Goller, C., Schulz, S.: Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München (1999)
4. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: CADE 11 (2011) (To appear)
5. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reasoning 40(1), 35–60 (2008)
6. Otten, J., Bibel, W.: leanCoP: Lean Connection-Based Theorem Proving. Journal of Symbolic Computation 36(1-2), 139–161 (2003)
7. Otten, J.: leanCoP 2.0 and ileanCoP 1.2: High performance lean theorem proving in classical and intuitionistic logic (system descriptions). In: Armando, A., et al. (eds.) [1], pp. 283–291
8. Otten, J.: Restricting backtracking in connection calculi. AI Commun. 23(2-3), 159–182 (2010)
9. Prevosto, V., Waldmann, U.: SPASS+T. In: Sutcliffe, G., Schmidt, R., Schulz, S. (eds.) ESCoR 2006. CEUR, vol. 192, pp. 18–33 (2006)
10. Riazanov, A., Voronkov, A.: The Design and Implementation of Vampire. AI Communications 15(2-3), 91–110 (2002)
11. Schulz, S.: E: A Brainiac Theorem Prover. AI Communications 15(2-3), 111–126 (2002)
12. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, Cambridge (2004)
13. Suda, M., Sutcliffe, G., Wischnewski, P., Lamotte-Schubert, M., de Melo, G.: External Sources of Axioms in Automated Theorem Proving. In: Mertsching, B., Hund, M., Aziz, Z. (eds.) KI 2009. LNCS, vol. 5803, pp. 281–288. Springer, Heidelberg (2009)
14. Tsivtsivadze, E., Urban, J., Geuvers, H., Heskes, T.: Semantic graph kernels for automated reasoning. In: SDM 2011 (to appear, 2011)
15. Urban, J.: MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. International Journal on Artificial Intelligence Tools 15(1), 109–130 (2006)
16. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1-2), 21–43 (2006)
17. Urban, J., Hoder, K., Voronkov, A.: Evaluation of automated theorem proving on the Mizar Mathematical Library. In: ICMS, pp. 155–166 (2010)
18. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLARea SG1- machine learner for automated reasoning with semantic guidance. In: Armando, et al. (eds.) [1], pp. 441–456