Premise Selection for Mathematics by Corpus Analysis and Kernel Methods

Jesse Alama[*] · **Tom Heskes**[**] · **Daniel Kühlwein**[**] · **Evgeni Tsivtsivadze**[**] · **Josef Urban**[**]

Abstract Smart premise selection is essential when using automated reasoning as a tool for large-theory formal proof development. A good method for premise selection in complex mathematical libraries is the application of machine learning to large corpora of proofs.

This work develops learning-based premise selection in two ways. First, a newly available minimal dependency analysis of existing high-level formal mathematical proofs is used to build a large knowledge base of proof dependencies, providing precise data for ATP-based re-verification and for training premise selection algorithms. Second, a new machine learning algorithm for premise selection based on kernel methods is proposed and implemented. To evaluate the impact of both techniques, a benchmark consisting of 2078 large-theory mathematical problems is constructed, extending the older MPTP Challenge benchmark. The combined effect of the techniques results in a 50% improvement on the benchmark over the Vampire/SInE state-of-the-art system for automated reasoning in large theories.

1 Introduction

In this paper we significantly improve theorem proving in large formal mathematical libraries by using a two-phase approach combining precise proof analysis with machine learning of premise selection.

The first phase makes the first practical use of the newly available *minimal dependency analysis* of the proofs in the large Mizar Mathematical Li-

brary (MML)[1]. This analysis allows us to construct precise problems for ATP-based re-verification of the Mizar proofs. More importantly, the precise dependency data can be used as a large repository of previous problem-solving knowledge from which premise selection can be efficiently automatically learned by machine learning algorithms.

In the second phase, a complementary improvement is achieved by using a new kernel-based machine learning algorithm, which outperforms existing methods for premise selection. This means that based on the large number of previously solved mathematical problems, we can more accurately estimate which premises will be useful for proving a new conjecture.

Such learned knowledge considerably helps automated proving of new formally expressed mathematical problems by recommending the most relevant previous theorems and definitions from the very large existing libraries, and thus shielding the existing ATP methods from considering thousands of irrelevant axioms. The better such symbiosis of formal mathematics and learning-assisted automated reasoning gets, the better for both parties: improved automated reasoning increases the efficiency of formal mathematicians, and lowers the cost of producing formal mathematics. This in turn leads to larger corpora of previously solved nontrivial problems from which the learning-assisted ATP can extract additional problem-solving knowledge covering larger and larger parts of mathematics.

The rest of the paper is organized as follows. Section 2 describes recent developments in large-theory automated reasoning and motivates our problem. Section 3 summarizes the recent implementation of precise dependency analysis over the large MML, and its use for ATP-based cross-verification and training premise selection. Section 4 describes the general machine learning approach to premise selection and an efficient kernel-based multi-output ranking algorithm for premise selection. In Section 5 a new large-theory benchmark of 2078 related MML problems is defined, extending the older and smaller MPTP Challenge benchmark, and our techniques are evaluated on this benchmark in Section 6. Section 7 concludes and discusses future work and extensions.

2 Automated Reasoning in Large Theories (ARLT)

In recent years, large formal libraries of re-usable knowledge expressed in rich formalisms have been built with interactive proof assistants, such as Mizar [11], Isabelle [17], Coq [6], HOL (light) [13], and others. Formal approaches are also being used increasingly in non-mathematical fields such as software and hardware verification and common-sense reasoning about real-world knowledge. Such trends lead to growth of formal knowledge bases in these fields.

One important development is that a number of these formal knowledge bases and core logics have been translated to first-order formats suitable for ATPs [16,32,19], and first-order ATP is today routinely used for proof assistance in systems like Isabelle [18,7], Mizar [35,34], and HOL [12]. These

---

[1] http://www.mizar.org

first-order translations give rise to large, semantically rich corpora that present significant new challenges for the field of automated reasoning. The techniques developed so far for ATP in large theories can be broadly divided into two categories:

1. Heuristic symbolic analysis of the formulas appearing in problems, and
2. Analysis of previous proofs.

In the first category, the SInE preprocessor by K. Hoder [14, 33] has so far been the most successful. SInE is particularly strong in domains with many hierarchical definitions such as those in common-sense ontologies. In the second category, machine learning of premise selection, as done e.g. by the MaLARea [36] system, is an effective method in hard mathematical domains, where the knowledge bases contain proportionally many more nontrivial lemmas and theorems than simple definitions, and previous verified proofs can be used for learning proof guidance.

Automated reasoning in large mathematical corpora is an interesting new field in several respects. Large theories permit data-driven approaches [27] to constructing ATP algorithms; indeed, the sheer size of such libraries actually necessitates such methods. It turns out that purely deductive, brute-force search methods can be improved significantly by heuristic and inductive[2] methods, thus allowing experimental research into combinations [36] of inductive and deductive methods. Large-theory benchmarks like the MPTP Challenge[3], and its extended version developed here in Section 5, can serve for rigorous evaluation of such novel Artificial Intelligence (AI) methods over thousands of real-world mathematical problems.[4] Apart from the *novel AI* aspect, and the obvious *proof assistance* aspect, automated reasoning over large formal mathematical corpora can also become a new tool in the established field of *reverse mathematics* [28]. This line of research has been already started, for example by Solovay's analysis [29] of the connection between Tarski's axiom [30] and the axiom of choice, and by Alama's analysis of the Euler's polyhedron formula [1], both conducted over the MML.

3 Computing Minimal Dependencies in Mizar

In the world of automated theorem proving, proofs contain essentially all logical steps, even very small ones (such as the steps taken in a resolution proof). In the world of interactive theorem proving, one of the goals is to allow the users to express themselves with minimal verbosity. Towards that end, interactive theorem proving (ITP) systems often come with mechanisms for suppressing some steps of an argument. By design, an ITP can suppress logical and mathematical steps that might be necessary for a complete analysis of

---

[2] The word *inductive* denotes here *inductive reasoning*, as opposed to *deductive reasoning*.

[3] http://www.tptp.org/MPTPChallenge

[4] We do not evaluate on the CASC LTB datasets, because they are too small to allow machine learning techniques. Our goal is to help mathematicians who work with and re-use large amounts of previously established complex proofs and theorems.

what a particular proof depends upon. In this section we summarize a recently developed solution to this problem for the MML. The basis of the solution is refactoring of the articles of the MML into one-item micro-articles, and computing their minimal dependencies by a brute-force minimization algorithm. For a more detailed discussion of Mizar, see [15, 11]; for a more detailed discussion of refactoring and minimization algorithms, see [4].

As an example of how inferences in ITP-assisted formal mathematical proofs can be suppressed, consider a theorem of the form

$$\forall x : \tau[\psi\,(g(x))],$$

where "$x : \tau$" means that the variable x has type τ, and g is a unary function symbol that accepts arguments of type τ'. Suppose further that, prior to the assertion of this theorem, it is proved that τ is a subtype of τ'. The well-formedness of the theorem depends on this subtyping relationship. Moreover, the proof of the theorem may not mention this fact; the subtyping relationship between τ and τ' may very well not be an outright theorem. In such a situation, the fact

$$\forall x(x : \tau \to x : \tau')$$

is suppressed. We can see that by not requiring the author of a formal proof to supply such subtyping relationships, we permit him to focus more on the heart of the matter of his proof, rather than repeating the obvious. But if we are interested in giving a complete answer to the question of what a formalized proof depends upon, we must expose suppressed facts and inferences. Having the complete answer is important for a number of applications, see [4] for examples. The particular importance for the work described here is that when efficient first-order ATPs are used to assist high-level formal proof assistants like Mizar, the difference between the *implicitly* used facts and the *explicitly* used facts disappears. The ATPs need to explicitly know all the facts that are necessary for finding the proofs. (If we were to omit the subtyping axiom, for example, an ATP might find that the problem is countersatisfiable.)

The first step in the computation of fine-grained dependencies in Mizar is to break up each article in the MML into a sequence of Mizar texts, each consisting of a single top-level item (e.g., theorem, definition). Each of these texts can—with suitable preprocessing—be regarded as a complete, valid Mizar article in its own right. The decomposition of a whole article from the MML into such smaller articles typically requires a number of nontrivial refactoring steps, comparable, e.g., to automated splitting and re-factoring of large programs written in programming languages with complicated syntactic mechanisms.

In Mizar, every article begins with a so-called *environment* specifying the background knowledge (theorems, notations, etc.) that is used to verify the article. The actual Mizar content that is imported, given an environment, is, in general, a rather conservative overestimate of the items that the article actually needs. That is why we apply a greedy minimization process to the environment to compute a minimal set of items that are sufficient to verify

each "micro-article". This produces a minimal set of dependencies[5] for each Mizar item, both *syntactic* (e.g., notational macros), and *semantic* (e.g., theorems, typings, etc.). The drawback of this minimization process is that the greedy approach to minimization[6] of certain kinds of dependencies can be time consuming.[7] The advantage is that (unlike in any other proof assistant) the computed set of dependencies is truly minimal (with respect to the power of the proof checker), and does not include redundant dependencies which are typically drawn in by overly powerful proof checking algorithms (like congruence closure over sets of all available equalities, etc.) when the dependency tracking is implemented internally inside a proof assistant. The dependency minimization is particularly important for the ATP and premise-selection applications that are explained in this paper: a day more of routine computation of the minimal dependencies is a very good time investment if it can provide better guidance for the fast-growing search space explored by ATPs. Another advantage of this approach is that it also provides *syntactic* dependencies, which are needed for real-world recompilation of the particular item as written in the article. This functionality is important for fast fine-grained recompilation in formal wikis [2], however for semantic applications like ATP we are only considering the truly semantic dependencies, i.e., those dependencies that result in a formula when translated by the MPTP system [32] to first-order logic.

Table 1 provides a summary of the fine-grained dependency data for the set of 33 Mizar articles coming from the MPTP2078 benchmark developed in Section 5, and used for the experiments in Section 6. For each theorem in the sequence of the 33 Mizar articles (ordered from first to last by their order in the MML) we show how many explicit dependencies are involved (on average) in their proofs and how many implicit dependencies (on average) it contains. The table also shows how much of an improvement the exact dependency calculation is, compared to a simple safe fixed-point MPTP construction of an over-approximation of what is truly used in the MML proof.

4 Premise Selection in Large Theories by Machine Learning

When reasoning over a large theory (like the MML), thousands of premises are available. In the presence of such large numbers of premises, the performance of most ATP systems degrades considerably [33]. Yet typically only a fraction of the available premises are actually needed to construct a proof. Estimating which premises are likely to be useful for constructing a proof is our research problem:

---

[5] Precisely, the minimality means that removing any dependence will cause the verification to fail.

[6] The basic greedy minimization proceeds by checking if an article still compiles after removing increasingly larger parts of the environment.

[7] This can be improved by heuristics for guessing the needed dependencies, analogous to those used for ATP premise selection.

Table 1 Effectiveness of fine-grained dependencies on the 33 MPTP2078 articles ordered from top to bottom by their order in the MML.

Article	Theorems	Expl. Refs.	Uniq. Expl. Refs.	Fine Deps.	MPTP Deps.
xboole_0	7	4	2.7	11.57	12.62
xboole_1	117	5.34	2.3	15.27	17.86
enumset1	87	3.26	2.7	10.67	10.82
zfmisc_1	129	4.74	2.9	16.59	21.08
subset_1	43	4.62	2.4	22.30	28.15
setfam_1	48	6.56	2.4	25.62	37.93
relat_1	184	5.66	2.2	19.97	27.31
funct_1	107	7.69	3.4	22.94	42.05
ordinal1	37	7.81	3.9	26	61.92
wellord1	53	11.7	5.6	30.45	55.5
relset_1	32	4.71	2.6	27	55.59
mcart_1	92	5.71	2.8	21.25	29.77
wellord2	24	14.2	6.9	36.41	75.2
funct_2	124	4.14	2.5	30.77	92.45
finset_1	15	8.66	3.9	23.93	72.22
pre_topc	36	6.47	3.2	35.58	53.51
orders_2	56	10.6	5.3	46.28	79.77
lattices	27	5.59	3.1	43	72.96
tops_1	71	6.67	4.2	43.46	66.42
tops_2	65	7.36	4.0	37.41	103.4
compts_1	23	17.7	8.6	48.86	102.9
connsp_2	29	9.86	6.4	39.51	96.5
filter_1	61	14.6	5.3	52.27	122.8
lattice3	55	8.6	4.2	47.07	92.25
yellow_0	70	6.75	3.2	26.55	46.12
yellow_1	28	9.03	5.1	53.17	128.3
waybel_0	76	9.34	4.1	35.03	82.34
tmap_1	141	8.78	4.5	47.04	140.0
tex_2	74	7.66	5.0	37.31	155.6
yellow_6	44	11.2	6.4	50.31	173.5
waybel_7	46	13.0	7.7	57.10	140.3
waybel_9	41	9	5.2	51.56	156.5
yellow19	36	8.44	5.3	53.02	137.2

Article: Mizar Article relevant to the MPTP2078 benchmark.

Theorems: Total number of theorems in the article.

Expl. Refs.: Average number of (non-unique) explicit references (to theorems, definitional theorems, and schemes) per theorem in the article.

Uniq. Expl. Refs.: Average number of unique explicit references per theorem.

Fine Deps.: Average number of all (both explicitly and implicitly) used items (explicitly referred to theorems, together with implicitly used items) per theorem as computed by dependency analysis.

MPTP Deps.: Average number of items per theorem as approximated by the MPTP fixpoint algorithm.

Definition 1 (Premise selection problem)
Given a large number of premises $\mathcal{P}$ and a new conjecture c, predict those premises from $\mathcal{P}$ that are likely to be useful for automatically constructing a proof of c.

Knowledge of previous proofs and problem-solving techniques is used by mathematicians to guide their thinking about new problems. The detailed MML proof analysis described above provides a large computer-understandable corpus of dependencies of mathematical proofs. In this section we present the machine learning setting and algorithms that are used to train premise selection on such corpora. Our goal is to begin emulating the training of human mathematicians.

When the translation [32] from Mizar to ATP formats is applied, the Mizar theorems and their proof dependencies (definitions, other theorems, etc.) translate to first-order formulas, used in the corresponding ATP problems as *conjectures* and their *premises* (axioms). For further presentation here we identify each MML formula with its first-order translation.[8] We will work in the following setting, which is tailored to the MML, but can easily be translated to other large datasets. Let Γ be the set of first order formulas that appear in the MML.

Definition 2 (Proof matrix) Using the fine-grained Mizar proof analysis - which says for each pair of formulas $p, c \in \Gamma$ whether p is used in the MML proof of c - define the function $\mu : \Gamma \times \Gamma \to \{0, 1\}$ by

$$\mu(c, p) := \begin{cases} 1 & \text{if } p \text{ is used to prove } c, \\ 0 & \text{otherwise.} \end{cases}$$

In other words, μ is the adjacency matrix of the graph of the direct MML proof dependencies. This proof matrix, together with suitably chosen formula features, will be used for training machine learning algorithms.

Note that in the MML, there is always exactly one (typically a textbook) proof of a particular theorem c, and hence exactly one set of premises $usedPremises(c) := \{p \mid \mu(c, p) = 1\}$ used in the proof of c. This corresponds to the mathematical textbook practice where typically only one proof is given for a particular theorem. It is however obvious that for example any expansion of the proof dependencies can lead to an alternative proof.

In general, given a mathematical theory, there can be a variety of more or less related alternative proofs of a particular theorem. This variety however typically is not the *explicit* textbook data on which mathematicians study. Such variety is only formed (in different measure) in their minds, after studying (training on) the textbook proofs, which typically are chosen for some nice properties (simplicity, beauty, clarity, educational value, etc.). Hence it is also plausible to use the set of MML proofs for training algorithms that attempt to emulate human proof learning. Below we will often refer to the set of premises

---

[8] Mizar is "nearly first-order", so the correspondence is "nearly" one-to-one, and in particular it is possible to construct the set of ATP premises from the exact Mizar dependencies for each Mizar theorem.

used in the (unique) MML proof of a theorem c as *the* set of premises of c. This concept is not to be read as the *only* set of premises of c, but rather as *the particular set of premises that is used in human training*, and therefore is also likely to be useful in training computers. It would not be difficult to relax this approach, if the corpora from which we learn contained a number of good alternative proofs. This is however so far not the case with the current MML, on which we conduct these experiments.

Also note that although our training set consists of formal proofs, these proofs have been authored by humans, and not found fully automatically by ATPs. But the evaluation conducted here (Section 6) is done by running ATPs on the recommended premises. It could be the case (depending on the ATP implementation) that a fully automatically found proof would provide a better training example than the human proof from the MML. A major obstacle for such training is however the relative weakness of existing ATPs in finding more involved proofs of MML theorems (see [33]), and thus their failure to provide the training examples for a large part of MML. Still, a comparison of the power of training on MML and ATP proofs could be interesting future work.

Definition 3 (Feature matrix) We characterize MML formulas by the symbols and (sub)terms appearing in them. We use de Bruijn indices for variables, and term equality is then just string equality. Let $T := \{t_1, \ldots, t_m\}$ be a fixed enumeration of the set of all symbols and (sub)terms that appear in all formulas from Γ. We define $\Phi : \Gamma \times \{1, \ldots, m\} \to \{0, 1\}$ by

$$\Phi(c, i) := \begin{cases} 1 & \text{if } t_i \text{ appears in } c, \\ 0 & \text{otherwise.} \end{cases}$$

This matrix gives rise to the *feature function* $\varphi : \Gamma \to \{0, 1\}^m$ which for $c \in \Gamma$ is the vector φ^c with entries in $\{0, 1\}$ satisfying

$$\varphi_i^c = 1 \iff \Phi(c, i) = 1.$$

The *expressed features of a formula* are denoted by the value of the function $e : \Gamma \to \mathcal{P}(T)$ that maps c to $\{t_i \mid \Phi(c, i) = 1\}$.

Note that our choice of feature characterization is quite arbitrary. We could try to use only symbols, or only (sub)terms, or some totally different features. The better the features correspond to the concepts that are relevant when choosing theorems for solving a particular problem, the more successful the machine learning of premise selection can be. As with the case of using alternative proofs for training, we just note that finding suitable feature characterizations is a very interesting problem in this area, and that our current choice seems to perform already quite reasonably in the experiments. For the particular heuristic justification of using formula (sub)terms, see [36].

The premise selection problem can be treated as a ranking problem, or as a classification problem. In the ranking approach, we for a given a conjecture c rank the available premises by their predicted usefulness for an automated proof of c, and use some number n of premises with the highest ranking (denoted here as $advisedPremises(c, n)$). In the classification approach,

we are looking for each premise $p \in \Gamma$ for a real-valued *classifier function* $C_p(\cdot) : \Gamma \to \mathbb{R}$ which, given a conjecture c, estimates how useful p is for proving c. In standard classification, a premise p would then be used if $C_p(c)$ is above certain threshold. A common approach to ranking is to use classification, and to combine the real-valued classifiers [21]: the premises for a conjecture c are ranked by the values of $C_p(c)$, and we choose a certain number of the best ones. This is the approach that we use in this paper.

Given a training corpus, machine learning algorithms can automatically *learn* classifier functions. The main difference between learning algorithms is the function space in which they search for the classifiers and the measure they use to evaluate how good a classifier is. In our prior work on the applications of machine learning techniques to the premise selection problem [36] we used the SNoW implementation [9] of a multiclass naive Bayes learning method because of its efficiency. In this work, we experiment with state-of-the-art *kernel-based learning methods* for premise selection. We present both methods and show the benefits of using kernels.

4.1 A Naive Bayes Classifier

Naive Bayes is a statistical learning method based on Bayes's theorem about conditional probabilities[9] with a strong (read: naive) independence assumptions. In the naive Bayes setting, the value $C_p(c)$ of the classifier function of a premise p at a conjecture c is the probability that $\mu(c,p) = 1$ given the expressed features $e(c)$.

To understand the difference between the naive Bayes and the kernel-based learning algorithm we need to take a closer look at the naive Bayes classifier. Let θ denote the statement that $\mu(c,p) = 1$ and for each feature $t_i \in T$ let $\bar{t}_i$ denote that $\Phi(c,i) = 1$. Furthermore, let $e(c) = \{s_1, \ldots, s_l\} \subseteq T$ be the expressed features of c (with corresponding $\bar{s}_1, \ldots, \bar{s}_l$). We have

---

[9] In its simplest form, Bayes's theorem asserts for a probability function P and random variables X and Y that

$$P(X|Y) = \frac{P(Y|X)P(X)}{P(Y)},$$

where $P(X|Y)$ is understood as the conditional probability of X given Y.

$$P(\theta \mid \bar{s}_1, \dots, \bar{s}_l) \propto \ln \frac{P(\theta \mid \bar{s}_1, \dots, \bar{s}_l)}{P(\neg\theta \mid \bar{s}_1, \dots, \bar{s}_l)} \tag{1}$$

$$= \ln \frac{P(\bar{s}_1, \dots, \bar{s}_l \mid \theta)P(\theta)}{P(\bar{s}_1, \dots, \bar{s}_l \mid \neg\theta)P(\neg\theta)} \tag{2}$$

$$= \ln \frac{P(\bar{s}_1, \dots, \bar{s}_l \mid \theta)}{P(\bar{s}_1, \dots, \bar{s}_l \mid \neg\theta)} + \ln \frac{P(\theta)}{P(\neg\theta)} \tag{3}$$

$$= \ln \prod_{i=1}^{l} \frac{P(\bar{s}_i \mid \theta)}{P(\bar{s}_i \mid \neg\theta)} + \ln \frac{P(\theta)}{P(\neg\theta)} \text{ by independence} \tag{4}$$

$$= \sum_{i=1}^{m} \varphi_i^c \ln \left(\frac{P(\bar{t}_i \mid \theta)}{P(\bar{t}_i \mid \neg\theta)} \right) + \ln \frac{P(\theta)}{P(\neg\theta)} \tag{5}$$

$$= \mathbf{w}^T \boldsymbol{\varphi}^c + \ln \frac{P(\theta)}{P(\neg\theta)} \tag{6}$$

where

$$w_i := \ln \left(\frac{P(\bar{t}_i \mid \theta)}{P(\bar{t}_i) \mid \neg\theta} \right) \tag{7}$$

Line (6) shows that the naive-Bayes classifier is "essentially" (after the monotonic transformation) a linear function of the features of the conjecture. The feature weights $\mathbf{w}$ are here computed using formula (7).

4.2 Kernel-based Learning

We saw that the naive Bayes algorithm gives rise to a linear classifier. This leads to several questions: 'Are there better parameters?' and 'Can one get better performance with non-linear functions?'. Kernel-based learning provides a framework for investigating such questions. In this subsection we give a simplified, brief description of kernel-based learning that is tailored to our present problem; for further information, see [5, 25, 27].

4.2.1 Are there better parameters?

To answer this question we must first define what 'better' means. Using the number of problems solved as measure is not feasible because we cannot practically run an ATP for every possible parameter combination. Instead, we measure how *good* a classifier approximates our training data. We would like to have that

$$\forall x \in \Gamma : C_p(x) = \mu(x, p).$$

However, this will almost never be the case. To compare how well a classifier approximates the data, we use loss functions and the notion of expected loss that they provide, which we now define.

Definition 4 (Loss function and Expected Loss) A *loss function* is any function $l : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^+$.

Given a loss function l we can then define the *expected loss* $E(\cdot)$ of a classifier C_p as

$$E(C_p) = \sum_{x \in \Gamma} l(C_p(x), \mu(x, p))$$

One might add additional properties such as $l(x, x) = 0$, but this is not necessary. Typical examples of a loss function $l(x, y)$ are the square loss $(y - x)^2$ or the 0-1 loss defined by $I(x = y)$.

We can compare two different classifiers via their expected loss. If the expected loss of classifier C_p is less than the expected loss of a classifier C_q then C_p is the better classifier. It should be noted that a lower expected loss on a particular training set (like the MML proofs) need not necessarily lead to more solved problems by an ATP. One could imagine that the training set contains proofs that are very different from the way a particular ATP would proceed most easily. Also, what happens if the classifier is not able to predict all MML premises, but just a large part of them? These are questions about alternative proofs, and about the robustness of the ATP and prediction methods. An experimental answer is provided in Section 6.3.

4.2.2 Nonlinear Classifiers

It seems straightforward that more complex functions would lead to a lower expected loss and are hence desirable. However, parameter optimization becomes tedious once we leave the linear case. Kernels provide a way to use the machinery of linear optimization on non-linear functions.

Definition 5 (Kernel) A *kernel* is is a function $k : \Gamma \times \Gamma \to \mathbb{R}$ satisfying

$$k(x, y) = \langle \phi(x), \phi(y) \rangle$$

where $\phi : \Gamma \to F$ is a mapping from Γ to an inner product space F with inner product $\langle \cdot, \cdot \rangle$. A kernel can be understood as a *similarity measure* between two entities.

Example 1 A simple kernel for our setting is the linear kernel:

$$k_{\text{lin}}(x, y) := \langle \boldsymbol{\varphi}^x, \boldsymbol{\varphi}^y \rangle$$

with $\langle \cdot, \cdot \rangle$ being the normal dot product in $\mathbb{R}^m$. Here, $\boldsymbol{\varphi}^f$ denotes the features of a formula f (see definition 3), and the inner product space F is $\mathbb{R}^m$. A nontrivial example is the Gaussian kernel with parameter σ:

$$k_{\text{gauss}}(x, y) := \exp \left(-\frac{\langle \boldsymbol{\varphi}^x, \boldsymbol{\varphi}^x \rangle - 2\langle \boldsymbol{\varphi}^x, \boldsymbol{\varphi}^y \rangle + \langle \boldsymbol{\varphi}^y, \boldsymbol{\varphi}^y \rangle}{\sigma^2} \right)$$

We can now define our kernel function space in which we will search for classification functions.

Definition 6 (Kernel Function Space) Given a kernel k, we define

$$\mathcal{F}_k := \left\{ f \in \mathbb{R}^\Gamma \mid f(x) = \sum_{v \in \Gamma} \alpha_v k(x, v), \alpha_v \in \mathbb{R}, \|f\| < \infty \right\}.$$

as our *kernel function space*, where

$$\left\| \sum_{v \in \Gamma} \alpha_v k(x, v) \right\| = \sum_{u,v \in \Gamma} \alpha_u \alpha_v k(u, v)$$

Essentially, every function in $\mathcal{F}_k$ compares the input x with formulas in Γ using the kernel, and the weights α determine how important each comparison is[10].

The kernel function space $\mathcal{F}_k$ naturally depends on the kernel k. It can be shown that when we use k_{lin}, $\mathcal{F}_{k_{\text{lin}}}$ consists of linear functions of the MML features T. In contrast, the Gaussian kernel k_{gauss} gives rise to a very nonlinear kernel function space.

4.2.3 Putting it all together

Having defined loss functions, kernels and kernel function spaces we can now define how kernel-based learning algorithms learn classifier functions. Given a kernel k and a loss function l, recall that we measure how good a classifier C_p is with the expected loss $E(C_p)$. With all our definitions it seems reasonable to define C_p as

$$C_p := \arg\min_{f \in \mathcal{F}_k} E(f) \tag{8}$$

However, this is not what a kernel based learning algorithm does. There are two reasons for this. First, the minimum might not exist. Second, in particular when using complex kernel functions, such an approach might lead to overfitting: C_p might perform very well on our training data, but bad on data that was not seen before. To handle both problems, a *regularization parameter* $\lambda > 0$ is introduced to penalize complex functions (assuming that high complexity implies a high norm). This regularization parameter allows us to place a bound on possible solution which together with the fact that $\mathcal{F}_k$ is a Hilbert space ensures the existence of C_p. Hence we define

$$C_p = \arg\min_{f \in \mathcal{F}_k} E(f) + \lambda \|f\|^2 \tag{9}$$

Recall from the definition of $\mathcal{F}_k$ that C_p has the form

$$C_p(x) = \sum_{v \in \Gamma} \alpha_v k(x, v), \tag{10}$$

with $\alpha_v \in \mathbb{R}$. Hence, for any fixed λ, we only need to compute the weights α_v for all $v \in \Gamma$ in order to find C_p. In section 4.3 we show how to solve this optimization problem in our setting.

---

[10] A more general approach to kernel spaces is available; see [24].

4.2.4 Naive Bayes vs Kernel-based Learning

Kernel-based methods typically outperform the naive Bayes algorithm. There are several reasons for this. Firstly and most importantly, while naive Bayes is essentially a linear classifier, kernel based methods can learn non-linear dependencies when an appropriate non-linear (e.g. Gaussian) kernel function is used. This advantage in expressiveness usually leads to significantly better generalization[11] performance of the algorithm given properly estimated hyperparameters (e.g., the kernel width for Gaussian functions). Secondly, kernel-based methods are formulated within the regularization framework that provides mechanism to control the errors on the training set and the complexity ("expressiveness") of the prediction function. Such setting prevents overfitting of the algorithm and leads to notably better results compared to unregularized methods. Thirdly, some of the kernel-based methods (depending on the loss function) can use very efficient procedures for hyperparameter estimation (e.g. fast leave-one-out cross-validation [22]) and therefore result in a close to optimal model for the classification/regression task. For such reasons kernel-based methods are among the most successful algorithms applied to various problems from bioinformatics to information retrieval to computer vision [27]. A general advantage of naive Bayes over kernel-based algorithms is the computational efficiency, particularly when taking into account the fact that computing the kernel matrix is generally quadratic in the number of training data points. However, recent advances in large scale learning have led to extensions of various kernel-based methods such as SVMs, with sublinear complexity, provably fast convergence rate, and the generalization performance that cannot be matched by most of the methods in the field [26].

4.3 MOR Experimental Setup

For our experiments, we will now define a kernel-based multi-output ranking (MOR) algorithm that is a relatively straightforward extension of our preference learning algorithm presented in [31]. MOR is also based on the regularized least-squares algorithm presented in [22].

Let $\Gamma = \{x_1, \ldots, x_n\}$. Then formula (10) becomes

$$C_p(x) = \sum_{i=1}^{n} \alpha_i k(x, x_i)$$

Using this and the square-loss $l(x, y) = (x - y)^2$ function, solving equation (9) is equivalent to finding weights α_i that minimize

$$\min_{\alpha_1, \ldots, \alpha_n} \left[\sum_{i=1}^{n} \left(\sum_{j=1}^{n} \alpha_j k(x_i, x_j) - \mu(x_i, p) \right)^2 + \lambda \sum_{i,j=1}^{n} \alpha_i \alpha_j k(x_i, x_j) \right] \quad (11)$$

---

[11] *Generalization* is the ability of a machine learning algorithm to perform accurately on new, unseen examples after training on a finite data set.

Recall that C_p is the classifier for a single premise. Since we eventually want to rank all premises, we need to train a classifier for each premise. So we need to find weights $\alpha_{i,p}$ for each premise p. This does seem to complicate the problem quite a bit. However, we can use the fact that for each premise p, C_p depends on the values of $k(x_i, x_j)$, where $1 \leq i, j \leq n$, to speed up the computation. Instead of learning the classifiers C_p for each premise separately, we learn all the weights $\alpha_{p,i}$ simultaneously.

To do this, we first need some definitions. Let

$$A = (\alpha_{i,p})_{i,p} \quad (1 \leq i \leq n, p \in \Gamma).$$

A is the matrix where each column contains the parameters of one premise classifier. Define the kernel matrix K and the label matrix Y as

$$K := (k(x_i, x_j))_{i,j} \quad (1 \leq i, j \leq n)$$
$$Y := (\mu(x_i, p))_{i,p} \quad (1 \leq i \leq n, p \in \Gamma).$$

We can now rewrite (11) in matrix notation to state the problem for all premises:

$$\underset{A}{\arg\min} \operatorname{tr}\left((Y - KA)^{\mathrm{T}}(Y - KA) + \lambda A^{\mathrm{T}} KA\right) \tag{12}$$

where $\operatorname{tr}(A)$ denotes the trace of the matrix A. Taking the derivative with respect to A leads to:

$$\frac{\partial}{\partial A} \operatorname{tr}\left((Y - KA)^{\mathrm{T}}(Y - KA) + \lambda A^{\mathrm{T}} KA\right)$$
$$= -2K(Y - KA) + 2\lambda KA$$
$$= -2KY + (2KK + 2\lambda K)A$$

To find the minimum, we set the derivative to zero and solve with respect to A. This leads to:

$$A = (KK + \lambda K)^{-1} KY \tag{13}$$
$$= (K + \lambda I)^{-1} Y \tag{14}$$

In the experiments, we use the Gaussian kernel k_{gauss} we defined in Example 1. Ergo, if we fix the regularization parameter λ and the kernel parameter σ we can find the optimal weights through simple matrix computations. Thus, to fully determine the classifiers, it remains to find good values for the parameters λ and σ. This is done, as is common with such parameter optimization for kernel methods, by simple (logarithmically scaled) grid search and cross-validation on the training data using a 70/30 split.

5 Data: The MPTP2078 Benchmark

The effects of using the minimized dependency data (both for direct re-proving and for training premise selection), and the effect of using our kernel-based MOR algorithm are evaluated on a newly created large-theory benchmark[12] of 2078 related MML problems, which extends the older and smaller MPTP Challenge benchmark.

The original MPTP Challenge benchmark was created in 2006, with the purpose of supporting the development of ARLT (automated reasoning for large theories) techniques. It contains 252 related problems, leading to the Mizar proof of one implication of the Bolzano-Weierstrass theorem. The challenge has two divisions: *chainy* (harder) and *bushy* (easier). The motivation behind them is given below when we describe their analogs in the MPTP2078 benchmark.

Both the ARLT techniques and the computing power (particularly multi-core technology) have developed since 2006. Appropriately, we define a larger benchmark with a larger numbers of problems and premises, and making use of the more precise dependency knowledge. The larger number of problems together with their dependencies more faithfully mirror the setting that mathematicians are facing: typically, they know a number of related theorems and their proofs when solving a new problem.

The new MPTP2078 benchmark is created as follows: The 33 Mizar articles from which problems were previously selected for constructing the MPTP Challenge are used. We however use a new version of Mizar and MML allowing the precise dependency analysis, and use all problems from these articles. This yields 2078 problems. As with the MPTP Challenge benchmark, we create two groups (divisions) of problems.

Chainy: Versions of the 2078 problems containing all previous MML contents as premises. This means that the conjecture is attacked with "all existing knowledge", without any premise selection. This is a common use case for proving new conjectures fully automatically, see also Section 6.2. In the MPTP Challenge, the name *chainy* has been introduced for this division, because the problems and dependencies are ordered into a chronological chain, emulating the growth of the library.

Bushy: Versions of the 2078 problems with premises pruned using the new fine-grained dependency information. This use-case has been introduced in proof assistants by Harrison's `MESON_TACTIC` [12], which takes an explicit list of premises from the large library selected by a knowledgeable user, and attempts to prove the conjecture just from these premises. We are interested in how powerful ATPs can get on MML with such precise advice.

To evaluate the benefit of having precise minimal dependencies, we additionally also produce in this work versions of the 2078 problems with premises pruned by the old heuristic dependency-pruning method used for constructing re-proving problems by the MPTP system. The MPTP heuristic proceeds

---

[12] http://wiki.mizar.org/twiki/bin/view/Mizar/MpTP2078

by taking all *explicit* premises contained in the original human-written Mizar proof. To get all the premises used by Mizar *implicitly*, the heuristic watches the problem's set of symbols, and adds the implicitly used formulas (typically typing formulas about the problem's symbols) in a fixpoint manner. The heuristic attempts hard to guarantee completeness, however, minimality is not achievable with such simple approach.

All three datasets contain the same conjectures. They only differ in the number of redundant axioms. Note that the problems in the second and third dataset are considerably smaller than the unpruned problems. The average number of premises is 1976.5 for the unpruned (chainy) problems, 74 for the heuristically-pruned problems (bushy-old) and 31.5 for the problems pruned using fine-grained dependencies (bushy). Table 2 summarizes the datasets.

Table 2 Average Number of Premises in the three Datasets

Dataset	Premises used	Avg. number of premises
Chainy	All previous	1976.5
Bushy-Old	Heuristic dependencies	74
Bushy	Minimized dependencies	31.5

6 Experiments and Results

We use Vampire 0.6 [20] as the ATP system for all experiments conducted here. Adding other ATP systems is useful (see, e.g., [33] for recent evaluation), and there are metasystems like MaLARea which attempt to exploit the joint power of different systems in an organized way. However, the focus of this work is on premise selection, which has been shown to have similar effect across the main state-of-the-art ATP systems. Another reason for using the recent Vampire is that in [33], Vampire with the SInE preprocessor was sufficiently tested and tuned on the MML data, providing a good baseline for comparing learning-based premise-selection methods with robust state-of-the-art methods that can run on any isolated large problem without any learning. All measurements are done on an Intel Xeon E5520 2.27GHz server with 8GB RAM and 8MB CPU cache. Each problem is always assigned one CPU.

In Section 6.1 we evaluate the ATP performance when fine-grained dependencies (bushy problems) are used by comparing it to the ATP performance on the old MPTP heuristic pruning (bushy-old problems), and to the ATP performance on the large (chainy) versions of the MPTP2078 problems. These results show that there is a lot to gain by constructing good algorithms for premise selection. In Section 6.2 SNoW's naive Bayes and the MOR machine learning algorithms are incrementally trained on the fine-grained MML dependency data, and their precision in predicting the MML premises on new problems are compared. This standard machine-learning comparison is then in Section 6.3 com-

pleted by running Vampire on the premises predicted by the MOR and SNoW algorithms. This provides information about the overall theorem-proving performance of the whole dependency-minimization/learning/ATP stack. This performance is compared to the performance of Vampire/SInE.

6.1 Using the Fine-Grained Dependency Analysis for Re-proving

The first experiment evaluates the effect of fine-grained dependencies on re-proving Mizar theorems automatically. The results of Vampire/SInE run with 10s time limit[13] on the datasets defined above are shown in Table 3.

Table 3 Performance of Vampire (10s time limit) on 2078 MPTP2078 benchmark with different axiom pruning.

Pruning	Solved problems	Solved as percentage
Chainy	548	26.4
Chainy (Vampire `-d1`)	556	26.8
Bushy-old	1023	49.2
Bushy	1105	53.2

Vampire (run in the unmodified automated CASC mode) solves 548 of the unpruned problems. If we use the `-d1` parameter[14], Vampire solves 556 problems. Things change a lot with external premise pruning. Vampire solves 1023 of the 2078 problems when the old MPTP heuristic pruning (bushy-old) is applied. Using the pruning based on the new fine-grained analysis Vampire solves 1105 problems, which is an 8% improvement over the heuristic pruning in the number of problems solved. Since the heuristic pruning becomes more and more inaccurate as the MML grows (the ratio of MPTP Deps. to Fine Deps. in Table 1 has a growing trend from top to bottom), we can conjecture that this improvement will be even more significant when considering the whole MML. Also note that these numbers point to the significant improvement potential that can be gained by good premise selection: the performance on the pruned dataset is doubled in comparison to the unpruned dataset. Again, this ratio grows as MML grows, and the number of premises approaches 100.000.[15]

---

[13] There are several reasons why we use low time limits. First, Vampire performs reasonably with them in [33]. Second, low time limits are useful when conducting large-scale experiments and combining different strategies. Third, in typical ITP proof-advice scenarios [34], the preferable query response time is in (tens of) seconds. Fourth, 10 seconds in 2011 is much more than it was fifteen years ago, when the CASC competition started.

[14] The `-d` parameter limits the depth of recursion for the SInE algorithm. In [33] running Vampire with the `-d1` pruning parameter resulted in significant performance improvement on large Mizar problems.

[15] In the evaluation done in [33] on the whole MML with Vampire/SInE, this ratio is 39% to 14%.

6.2 Combining Fine-Grained Dependencies with Learning

For the next experiment, we emulate the growth of the library (limited to the
2078 problems), by considering all previous theorems and definitions when a
new conjecture is attempted. This is a natural "ATP advice over the whole
library" scenario, in which the ATP problems however become very large,
containing thousands of the previously proved formulas. Premise selection can
therefore help significantly.

  We use the fine-grained MML dependencies extracted from previous proofs[16]
to train the premise-selection algorithms, use their advice on the new prob-
lems, and compare the recall (and also the ATP performance in the next
subsection). For each problem, the learning algorithms are allowed to learn on
the dependencies of all previous problems, which corresponds to the situation
in general mathematics when mathematicians not only know many previous
theorems, but also re-use previous problem solving knowledge. This approach
requires us to do 2078 training steps as the problems and their proofs are
added to the library and the dataset grows. We compare the MOR algorithm
with SNoW's naive Bayes.

  Figure 1 shows the average recall of SNoW and MOR on this dataset.
The rankings obtained from the algorithms are compared with the actual
premises used in the MML proof, by computing the size (ratio) of the overlap
for the increasing top segments of the ranked predicted premises (the size of
the segment is the x axis in Figure 1). Formally, the recall $recall(c, n)$ for one
conjecture c when n premises are advised is defined as:

$$recall(c, n) = \frac{|usedPremises(c) \cap advisedPremises(c, n)|}{|usedPremises(c)|}$$

It can be seen that the MOR algorithm performs considerably better than
SNoW. E.g., on average 88% of the used premises are within the 50 high-
est MOR-ranked premises, whereas when we consider the SNoW ranking only
around 80% of the used premises are with the 50 highest ranked premises.

  Note that this kind of comparison is the standard endpoint in machine
learning applications like keyword-based document retrieval, consumer choice
prediction, etc. However, in a semantic domain like ours, we can go further,
and see how this improved prediction performance helps the theorem proving
process. This is also interesting to see, because having for example only 90%
coverage of the original MML premises could be insufficient for constructing an
ATP proof, unless the ATP can invent alternative (sub-)proofs.[17] This final
evaluation is done in the next section.

---

[16] We do not evaluate the performance of learning on the approximate bushy-old depen-
dencies here and in the next subsection. Table 1 and Section 6.1 already sufficiently show
that these data are less precise than the fine-grained MML dependencies.

[17] See [3] for an initial exploration of the phenomenon of alternative ATP proofs for MML
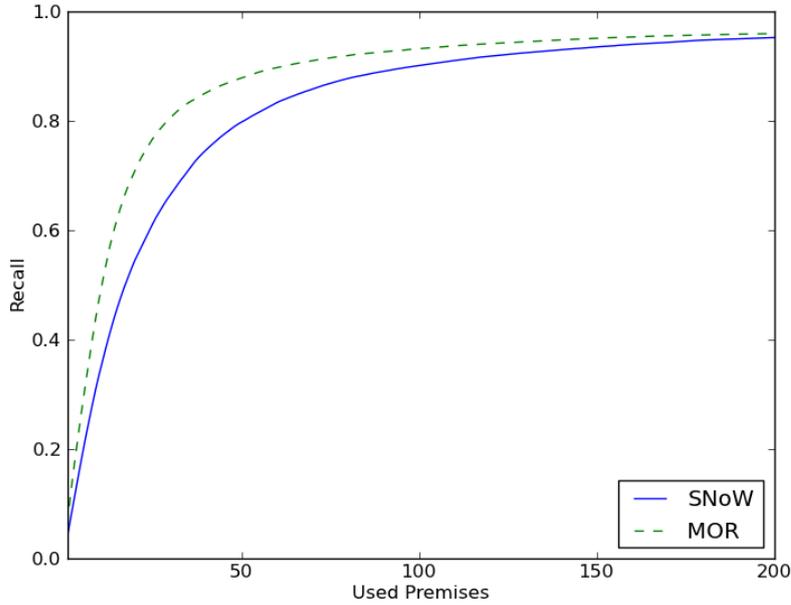theorems.

Fig. 1 Comparison of the SNoW and MOR average recall of premises used in the Mizar proofs. The x-axis shows the number of premises asked from SNoW and MOR, and the y-axis shows their relative overlap with the premises used in the original Mizar proof.

6.3 Combining It All: ATP Supported by Learning from Fine Dependencies

In the last experiment, we finally chain the whole ITP/Learning/ATP stack together, and evaluate how the influence of the improved premise selection reflects on performance of automated theorem proving on new large-theory conjectures. Both the naive Bayes (SNoW) and the new MOR learning algorithms are evaluated.

Figure 2 shows the numbers of problems solved by Vampire using different numbers of the top premises predicted by SNoW and MOR, and a 5 seconds time limit. The maximum number of problems solved with MOR is 729 with the top 60 advised premises. SNoW's maximum is 652 with the top 70 premises. The corresponding numbers for a 10 second time limit are 795 solved problems for MOR-60, and 722 for SNoW-70. Table 4 compares these data with the overall performance of Vampire with a 10 second time limit run on problems with pruning done by SInE. The SNoW-60 resp. MOR-70 runs give a 32% resp. 45% improvement over the 548 problems Vampire solves in auto-mode, and a 30% resp. 43% improvement over the 556 problems solved by Vampire using the -d1 option.
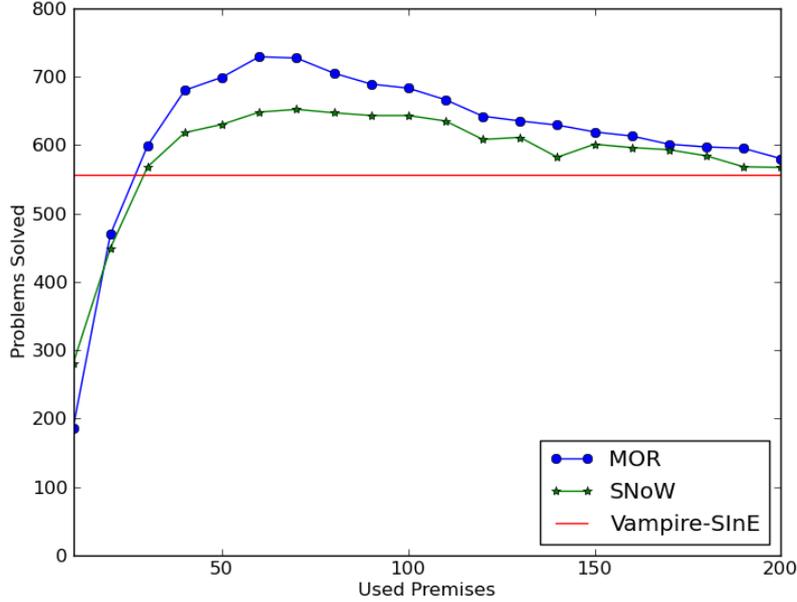
Fig. 2 Comparison of the number of solved problems by SNoW and MOR. The x-axis shows the number of premises given to the Vampire, and the y-axis shows the number of problems solved within 5 seconds. The number of problems solved by Vampire/SInE in 10 seconds is given as a baseline.

Table 4 Comparison of Vampire (10s time limit) performance on MPTP2078 with different premise selections.

System	Solved Problems	Gain over Vampire
Vampire/SInE	548	0 %
Vampire/SInE (-d1)	556	1.5%
SNoW-70	722	31.8%
MOR-60	795	45.1%

Table 5 additionally compares the performance of Vampire/SInE with the performance of SNoW and MOR when computed for each of them as a union of the two 5s runs with the largest joint coverage. Those are obtained by using the top 40 advised premises and the top 180 advised premises for SNoW, and the top 40 and top 100 advised premises for MOR. These SNoW resp. MOR combined runs give a 44% resp. 50% improvement over the 548 problems Vampire solves in auto-mode, and a 42% resp. 48% improvement over the 556 problems solved by Vampire using the -d1 option.

Note that Vampire/SInE does strategy scheduling internally, and with different SInE parameters. Thus combining two different premise selection strategies by us is perfectly comparable to the way Vampire's automated mode is constructed and used. Also note that combining the two different ways in which unadvised Vampire/SInE was run is not productive: the union of both unadvised runs is just 559 problems, which is only 3 more solved problems (generally in 20s) than with running Vampire/SInE with -d1 for ten seconds.

Table 5 10s performance of the two strategies with the largest joint coverage for SNoW and MOR.

System	Solved Problems	Gain over Vampire
Vampire/SInE	548	0 %
Vampire/SInE (-d1)	556	1.5%
SNoW-40/180	788	43.7%
MOR-40/100	824	50.4%

Finally, Figure 3 and Figure 4 compare the cumulative and average performance of the algorithms (combined with ATPs) at different points of the MPTP2078 benchmark, using the chronological ordering of the MPTP2078 problems. The average available number of premises for the theorems ordered chronologically grows linearly (the earlier theorems and definitions become eligible premises for the later ones), making the later problems harder on average. Figure 3 shows the performance computed on initial segments of problems using step value of 50. The last value (for 2050) corresponds to the performance of the algorithms on the whole MPTP2078 (0.38 for MOR-60), while for example the value for 1000 (0.60 for MOR-60) shows the performance of the algorithms on the first 1000 MPTP2078 problems. Figure 4 compares the average performance of the algorithms when the problems are divided into four successive segments of equal size. Note that even with the precise use of the MML premises the problems do not have uniform difficulty across the benchmark, and on average, even the bushy versions of the later problems get harder. To visualize this, we also add the values for Vampire-bushy to the comparison.

Except from small deviations, the ratio of solved problems decreases for all the algorithms. Vampire/MOR-60 is able to keep up with Vampire-bushy in the range of the initial 800 problems, and after that the human selection increasingly outperforms all the algorithms. Making this gap as small as possible is an obvious challenge on the path to strong automated reasoning in general mathematics.

7 Conclusion and Future Work

The performance of automated theorem proving over real-world mathematics has been significantly improved by using detailed minimized formally-assisted
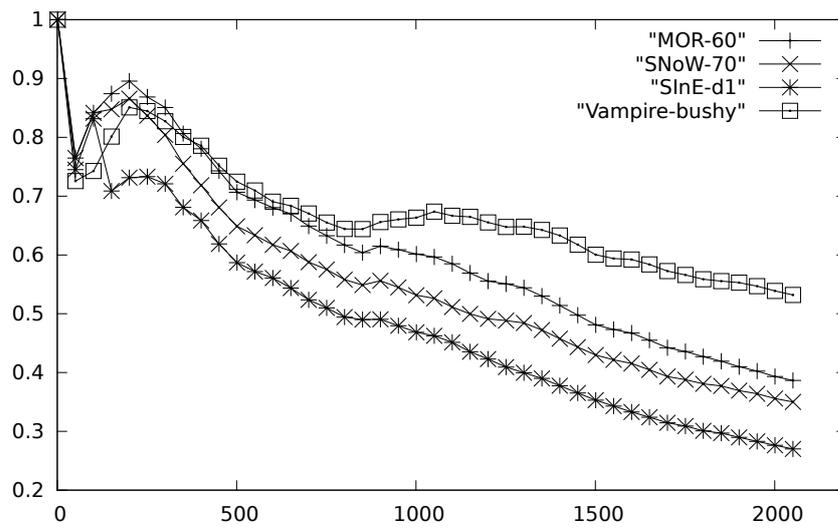
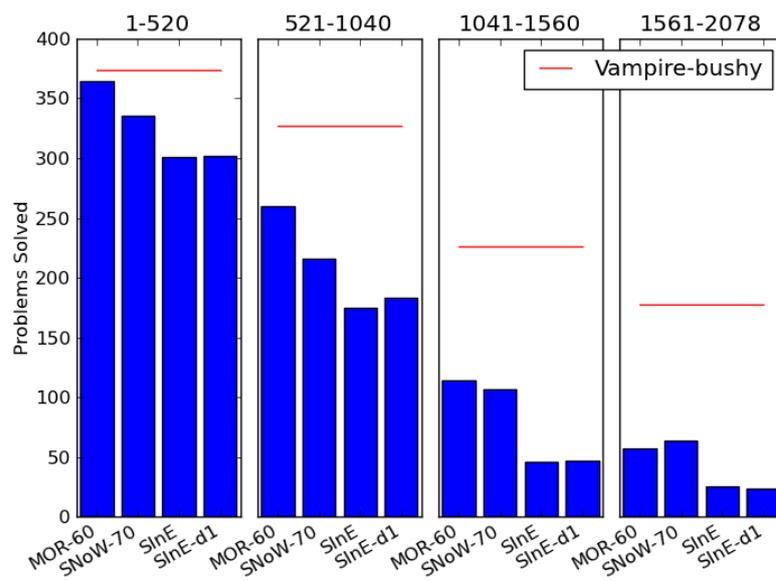**Fig. 3** Performance of the algorithms on initial segments of MPTP2078.

Fig. 4 Average performance of the algorithms on four successive equally sized segments of MPTP2078.

analysis of a large corpus of theorems and proofs, and by using improved prediction algorithms. In particular, it was demonstrated that premise selection based on learning from exact previous proof dependencies improves the ATP performance in large mathematical theories by about 44% when using off-the-shelf learning methods like naive Bayes in comparison with state-of-the-art general premise-selection heuristics like SInE. It was shown that this can be further improved to about 50% when employing state-of-the-art kernel-based learning methods.

Automated reasoning in large mathematical libraries is becoming a complex AI field, allowing interplay of very different AI techniques. Manual tuning of strategies and heuristics does not scale to large complicated domains, and data-driven approaches are becoming very useful in handling such domains. At the same time, existing strong learning methods are typically developed on imprecise domains, where feedback loops between prediction and automated verified confirmation as done for example in MaLARea are not possible. The stronger such AI systems become, the closer we get to formally assisted mathematics, both in its "forward" and "reverse" form. And this is obviously another positive feedback loop that we explore here: the larger the body of formally expressed and verified ideas, the smarter the AI systems that learn from them.

The work started here can be improved in many possible ways. While we have achieved 50% ATP improvement on large problems by better premise selection resulting in 824 problems proved within 10 seconds, we know (from 6.1) that with a better premise selection it is possible to prove at least 1105 problems. Thus, there is still a great opportunity for improved premise selection algorithms. Our dependency analysis can be finer and faster, and combined with ATP and machine learning systems, can be the basis for a research tool for experimental formal (reverse) mathematics. An interesting AI problem that is becoming more and more relevant as the ATP methods for mathematics are getting stronger, is translation of the (typically resolution-based) ATP proofs into human-understandable [10, 23] formats used by mathematicians. We believe that machine learning from large human-proof corpora like MML is likely to be useful for this task, in a similar way to how it is useful for finding relevant premises.

The MOR algorithm has a number of parameterizations that we have fixed for the experiments done here. Further experiments with different loss functions could yield better results. One of the most interesting parameterizations is the right choice of features for the formal mathematical domain. So far, we have been using only the symbols and terms occurring in formulas as their feature characterizations, but other features are possible, and very likely used by mathematicians. In particular, for ad hoc problem collections like the TPTP library, where symbols are used inconsistently across different problems, formula features that abstract from particular symbols will likely be needed. Also, the output of the learning algorithms does not have to be limited to the ranking of premises. In general, all kinds of relevant problem-solving parameterizations can be learned, and an attractive candidate for such treatment is the large set of ATP strategies and options parameterizing the proof search. With such ex-

periments, a large number of alternative ATP proofs are likely to be obtained, and an interesting task is to productively learn from such a combination of alternative (both human and machine) proofs. Premise selection is only one instance of the ubiquitous *proof guidance* problem, and recent prototypes like the MaLeCoP system [37] indicate that guidance obtained by machine learning can considerably help also *inside* automated theorem provers.

Finally, we hope that this work and the performance numbers obtained will provide a valuable feedback to the CADE competition organizers: Previous proofs and theory developments in general are an important part of real-world mathematics and theorem proving. At present, the LTB division of CASC does not recognize proofs in the way that we are recognizing them here. Organizing large-theory competitions that separate theorems from their proofs is like organizing web search competitions that separate web pages from their link structure [8]. We believe that re-introducing a large-theory competition that does provide both a large number of theorems and a large number of proofs will cover this important research direction, and most of all, properly evaluate techniques that significantly improve the ATP end-user experience.

8 Acknowledgment

References

1. Alama, J.: Formal Proofs and Refutations. Ph.D. thesis, Stanford University (2009)
2. Alama, J., Brink, K., Mamane, L., Urban, J.: Large formal wikis: Issues and solutions. In: Davenport, J., Farmer, W., Urban, J., Rabe, F. (eds.) Intelligent Computer Mathematics, Lecture Notes in Computer Science, vol. 6824, pp. 133–148. Springer Berlin / Heidelberg (2011)
3. Alama, J., Kühlwein, D., Urban, J.: Automated and human proofs in general mathematics: An initial comparison. In: Bjørner, N., Voronkov, A. (eds.) LPAR. Lecture Notes in Computer Science, vol. 7180, pp. 37–45. Springer (2012)
4. Alama, J., Mamane, L., Urban, J.: Dependencies in formal mathematics: Applications and extraction for Coq and Mizar (2011), arxiv.org pre-print abs/1109.3687
5. Aronszajn, N.: Theory of reproducing kernels. Transactions of the American Mathematical Society 68 (1950)
6. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development. Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer Verlag (2004)
7. Blanchette, J.C., Bulwahn, L., Nipkow, T.: Automatic proof and disproof in Isabelle/HOL. In: Tinelli, C., Sofronie-Stokkermans, V. (eds.) FroCos. Lecture Notes in Computer Science, vol. 6989, pp. 12–27. Springer (2011)
8. Brin, S., Page, L.: The anatomy of a large-scale hypertextual web search engine. Computer Networks 30(1-7), 107–117 (1998)
9. Carlson, A., Cumby, C., Rizzolo, N., Rosen, J., Roth, D.: SNoW user manual (1999), http://l2r.cs.uiuc.edu/~cogcomp/software/snow-userguide.pdf
10. Davis, M.: Obvious logical inferences. In: Hayes, P.J. (ed.) IJCAI. pp. 530–531. William Kaufmann (1981)

11. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a nutshell. Journal of Formalized Reasoning 3(2), 153–245 (2010)
12. Harrison, J.: Optimizing proof search in model elimination. In: McRobbie, M.A., Slaney, J.K. (eds.) CADE. Lecture Notes in Computer Science, vol. 1104, pp. 313–327. Springer (1996)
13. Harrison, J., Slind, K., Arthan, R.: HOL. In: Wiedijk, F. (ed.) The Seventeen Provers of the World. Lecture Notes in Computer Science, vol. 3600, pp. 11–19. Springer (2006)
14. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Automated Deduction – CADE-23, Lecture Notes in Computer Science, vol. 6803, pp. 299–314. Springer Berlin / Heidelberg (2011)
15. Matuszewski, R., Rudnicki, P.: Mizar: the first 30 years. Mechanized Mathematics and Its Applications 4, 3–24 (2005)
16. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. J. Autom. Reasoning 40(1), 35–60 (2008)
17. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL - A Proof Assistant for Higher-Order Logic, Lecture Notes in Computer Science, vol. 2283. Springer (2002)
18. Paulson, L.C., Susanto, K.W.: Source-level proof reconstruction for interactive theorem proving. In: Schneider, K., Brandt, J. (eds.) TPHOLs. Lecture Notes in Computer Science, vol. 4732, pp. 232–245. Springer (2007)
19. Pease, A., Sutcliffe, G.: First order reasoning on a large ontology. In: Sutcliffe, G., Urban, J., Schulz, S. (eds.) ESARLT. CEUR Workshop Proceedings, vol. 257. CEUR-WS.org (2007)
20. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Commun. 15(2-3), 91–110 (2002)
21. Richard, M.D., Lippmann, R.P.: Neural Network Classifiers Estimate Bayesian a posteriori Probabilities. Neural Computation 3(4), 461–483 (2010)
22. Rifkin, R., Yeo, G., Poggio, T., Rifkin, R., Yeo, G., Poggio, T.: Regularized Least-Squares Classification. In: Suykens, J., Horvath, G., Basu, S., Micchelli, C., Vandewalle, J. (eds.) Advances in Learning Theory: Methods, Model and Applications NATO Science Series III: Computer and Systems Sciences, vol. 190, pp. 131–154. IOS Press (2003)
23. Rudnicki, P.: Obvious inferences. J. Autom. Reasoning 3(4), 383–393 (1987)
24. Schoelkopf, B., Herbrich, R., Williamson, R., Smola, A.J.: A Generalized Representer Theorem. In: Helmbold, D., Williamson, R. (eds.) Proceedings of the 14th Annual Conference on Computational Learning Theory. pp. 416–426. Berlin, Germany (2001)
25. Scholkopf, B., Smola, A.J.: Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond. MIT Press, Cambridge, MA, USA (2001)
26. Shalev-Shwartz, S., Singer, Y., Srebro, N., Cotter, A.: Pegasos: primal estimated subgradient solver for SVM. Math. Program. 127(1), 3–30 (2011)
27. Shawe-Taylor, J., Cristianini, N.: Kernel Methods for Pattern Analysis. Cambridge University Press, New York, NY, USA (2004)
28. Simpson, S.G.: Subsystems of Second Order Arithmetic. Perspectives in Mathematical Logic, Springer, 2 edn. (2009)
29. Solovay, R.: AC and strongly inaccessible cardinals. Available on the Foundations of Mathematics archives at http://www.cs.nyu.edu/pipermail/fom/2008-March/012783.html (March 29 2008)
30. Tarski, A.: On well-ordered subsets of any set. Fundamenta Mathematicae 32, 176–183 (1939)
31. Tsivtsivadze, E., Pahikkala, T., Boberg, J., Salakoski, T., Heskes, T.: Co-regularized least-squares for label ranking. In: Hüllermeier, E., Fürnkranz, J. (eds.) Chapter in Preference Learning Book). pp. 107–123 (2010)
32. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. J. Autom. Reasoning 37(1-2), 21–43 (2006)
33. Urban, J., Hoder, K., Voronkov, A.: Evaluation of automated theorem proving on the Mizar mathematical library. In: Fukuda, K., van der Hoeven, J., Joswig, M., Takayama, N. (eds.) ICMS. Lecture Notes in Computer Science, vol. 6327, pp. 155–166. Springer (2010)
34. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. CoRR abs/1109.0616 (2011)

35. Urban, J., Sutcliffe, G.: Automated reasoning and presentation support for formalizing mathematics in Mizar. In: Autexier, S., Calmet, J., Delahaye, D., Ion, P.D.F., Rideau, L., Rioboo, R., Sexton, A.P. (eds.) AISC/MKM/Calculemus. Lecture Notes in Computer Science, vol. 6167, pp. 132–146. Springer (2010)
36. Urban, J., Sutcliffe, G., Pudlák, P., Vyskocil, J.: MaLARea SG1–machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR. Lecture Notes in Computer Science, vol. 5195, pp. 441–456. Springer (2008)
37. Urban, J., Vyskocil, J., Stepánek, P.: MaLeCoP: Machine learning connection prover. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX. Lecture Notes in Computer Science, vol. 6793, pp. 263–277. Springer (2011)