A Survey of Axiom Selection as a Machine Learning Problem

Jasmin Christian Blanchette[1] and Daniel Kühlwein[2]

[1] Fakultät für Informatik, Technische Universität München, Germany
[2] ICIS, Radboud Universiteit Nijmegen, the Netherlands

Abstract. Automatic theorem provers struggle to discharge proof obligations of interactive theorem provers. This is partly due to the large number of background facts that are passed to the automatic provers as axioms. Axiom selection algorithms predict the relevance of facts, thereby helping to reduce the search space of automatic provers. This paper presents an introduction to axiom selection as a machine learning problem and describes the challenges that distinguish it from other applications of machine learning.

1 Introduction

The foundations of modern mathematics were laid at the end of the 19th century and the beginning of the 20th century. Seminal works such as Frege's *Begriffsschrift* [8] established the notion of mathematical proofs as formal derivations in a logical calculus. In *Principia Mathematica* [51], Whitehead and Russell set out to show by example that all of mathematics can be derived from a small set of axioms using an appropriate logical calculus. Even though Gödel later showed that no consistent axiom system can capture all mathematical truth [9], *Principia* showed that normal mathematics can indeed be catered for by a formal system. Proofs could now be rigidly defined, and verifying the validity of a proof was a simple matter of checking whether the rules of the calculus were correctly applied. But formal proofs were extremely tedious to write (and read), and so they found no audience among practicing mathematicians.

1.1 Interactive Theorem Proving

With the advent of computers, formal mathematics became a more realistic proposal. *Interactive theorem provers* (ITP), or *proof assistants*, are computer programs that support the creation of formal proofs. Proofs are written in the input language of the ITP, which can be thought of as being at the intersection between a programming language, a logic, and a mathematical typesetting system. In an ITP proof, each statement the user makes gives rise to a proof obligation. The ITP ensures that every proof obligation is discharged by a correct proof.

ACL2 [21], Coq [3], HOL4 [41], HOL Light [14], Isabelle [37], Mizar [12], and PVS [38] are perhaps the most widely used ITPs. Figures 1 and 2 show a simple informal proof and the corresponding Isabelle proof. Virtually all ITPs provide some built-in automation in the form of *tactics* that perform arbitrarily complex reasoning.

THEOREM There are infinitely many primes:
for every number n there exists a prime $p > n$.

PROOF [after Euclid]
Given n. Consider $k = n! + 1$, where $n! = 1 \cdot 2 \cdot 3 \cdot \ldots \cdot n$.
Let p be a prime that divides k.
For this number p we have $p > n$: otherwise $p \leq n$;
but then p divides $n!$, so p cannot divide $k = n! + 1$,
contradicting the choice of p. QED

Figure 1: An informal proof that there are infinitely many prime numbers [50]

In Figure 2, the **by** command specifies which tactic should be applied to discharge the current proof obligation.

Developing proofs in ITPs usually requires a lot more work than sketching a proof with pen and paper. Nevertheless, the benefit of gaining quasi-certainty about the correctness of the proof led a number of mathematicians to adopt these systems.

The largest mechanization project is probably the ongoing formalization of the proof of Kepler's conjecture by Thomas Hales and his colleagues in HOL Light [13]. Other major undertakings are the formal proofs of the four-color theorem [10] and of the odd-order theorem [11] in Coq, both developed under Georges Gonthier's leadership. In terms of mathematical breadth, the *Mizar Mathematical Library* [30] is perhaps the main achievement of the ITP community so far: With nearly 52 000 theorems, it covers a large portion of the mathematics taught at the undergraduate level.

1.2 Automatic Theorem Proving

In contrast to interactive theorem provers, *automatic theorem provers* (ATPs) work without human interaction. They take a problem as input, consisting of a set of *axioms* and a *conjecture*, and attempt to deduce the conjecture from the axioms. The TPTP (Thousands of Problems for Theorem Provers) library [42] has established itself as a central infrastructure for exchanging ATP problems. Its main developer also organizes an annual competition, CADE's ATP Systems Competition (CASC) [43], that measures progress in this field. E [40], SPASS [49], Vampire [39], and Z3 [35] are well-known ATPs for classical first-order logic.

Some researchers apply ATPs to open mathematical problems. William McCune's proof of the Robbins conjecture using a custom ATP is the main success story [31]. More recently, ATPs have also been integrated into ITPs [6, 19, 46], where they help increase the productivity by reducing the number of manual interactions needed to carry out a proof. Instead of using a built-in tactic, the ITP translates the current proof obligation (e.g., the lemma that the user has just stated but not proved yet) into an ATP problem. If the ATP can solve it, the proof is translated to the logic of the ITP and the user can proceed. In a recent study, about 70% of the proof obligations arising in a representative Isabelle corpus could be solved by ATPs [26].

```
theorem Euclid: ∃p ∈ prime. n < p
proof –
  let ?k = n! + 1
  obtain p where prime: p ∈ prime and dvd: p dvd ?k
    using prime-factor-exists by auto
  have n < p
  proof –
    have ¬ p ≤ n
    proof
      assume p ≤ n
      with prime-g-zero have p dvd n! by (rule dvd-factorial)
      with dvd have p dvd ?k − n! by (rule dvd-diff)
      then have p dvd 1 by simp
      with prime show False using prime-nd-one by auto
    qed
    then show ?thesis by simp
  qed
  from this and prime show ?thesis ..
qed

corollary ¬ finite prime
  using Euclid by (fastsimp dest!: finite-nat-set-is-bounded simp: le-def)
```

Figure 2: An Isabelle proof corresponding to the informal proof of Figure 1 [50]

1.3 Industrial Applications

Apart from mathematics, formal proofs are also used in industry. With the ever increasing complexity of software and hardware systems, quality assurance is a large part of the time and money budget of projects. Formal mathematics can be used to *prove* that an implementation meets a specification. Although tests might still be mandated by certification authorities, formal proofs can both drastically reduce the testing burden and increase confidence that the systems are bug-free.

AMD and Intel have been verifying floating-point procedures since the late 1990s [15, 34], partly as a consequence of the Pentium bug. Microsoft have had success applying formal verification methods to Windows device drivers [2]. One of the largest software verification projects so far is seL4, a verified operating system kernel [22].

1.4 Learning to Reason

One of the main reasons why formal mathematics and related technologies have not become mainstream yet is that developing ITP proofs is tedious. The reasoning capabilities of ATPs and ITP tactics are in many respects far behind what is considered standard for a human mathematician. Developing an interactive proof requires not only knowledge of the subject of the proof, but also of the ITP and its libraries.

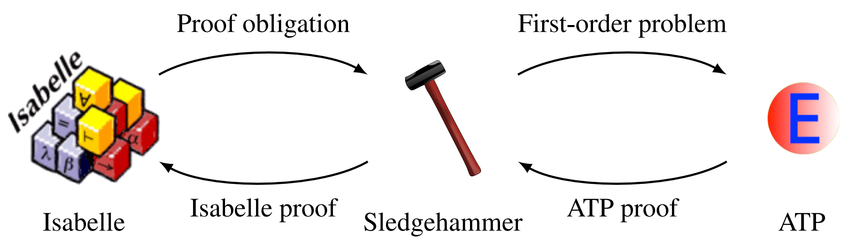

Figure 3: Sledgehammer [6] integrates ATPs (here E) into Isabelle

One way to make users of ITPs more productive is to improve the success rate of ATPs. ATPs struggle with problems that have too many unnecessary axioms since they increase the search space. This is especially an issue when using ATPs from an ITP, where users have access to thousands of facts (axioms, definitions, lemmas, theorems, and corollaries) in the background libraries. Each fact is a potential axiom for an ATP.[3] *Axiom selection algorithms* heuristically select facts that are likely to be useful for inclusion as axioms in the problem given to the ATP.

Learning mathematics involves studying proofs to develop a mathematical intuition. Experienced mathematicians often know how to approach a new problem by simply looking at its statement. Assume that p is a prime number and $a, b \in \mathbb{N} - \{0\}$. Consider the following statement:

$$\text{If } p \mid ab, \text{ then } p \mid a \text{ or } p \mid b.$$

Even though mathematicians know many areas of mathematics (e.g., linear algebra, probability theory, analysis), when trying to prove the above statement they would ignore those areas and rely on their knowledge about number theory. At an abstract level, they perform axiom selection to reduce their search space.

Axiom selection algorithms typically rely on static features of the conjecture and axioms [16, 33]. For example, if the conjecture involves π and sin, they will prefer axioms that contain either of these two symbols, ideally both. The main drawback of such approaches is that they focus exclusively on formulas, ignoring the rich information contained in proofs. In particular, they do not learn from previous proofs. In this paper, we present an overview to axiom selection as a machine learning problem, an idea introduced one decade ago by Urban [44]. In a way, we are trying to teach the computer mathematical intuition.

2 Machine Learning in a Nutshell

This section aims to provide a high-level introduction to machine learning; for a more thorough discussion, we refer to standard textbooks [4, 29, 36].

[3] A terminological note is in order. ITP axioms are fundamental assumption in the common mathematical sense (e.g., the axiom of choice). In contrast, ATP axioms are arbitrary formulas that can be used to establish the conjecture.

Machine learning concerns itself with extracting information from data. Some typical examples of machine learning are listed below:

Spam classification Predict if a new email is spam
Face detection Find human faces in a picture
Web search Predict the websites that contain the information the user is looking for

The results of a learning algorithm is a function that takes a new datapoint (email, picture, search query) and returns a target value (spam / not spam, location of faces, ranking of relevant websites). The learning is done by optimizing a *score function* over a *training dataset*. Typical score function are accuracy (how many emails were correctly labeled?) and the root mean square error (the Euclidean distance between the predicted values and the actual values). Elements of the training datasets are datapoints together with their expected value. For example:

Spam classification A set of emails together with their classification
Face detection A set of pictures where all faces are marked
Web search A set of query–websites tuples

The performance of the learned function critically depends on the quality of the training data, as expressed by the aphorism "Garbage in, garbage out." Getting training data that is representative for the problem, and hence generalizes well, is crucial.

In addition to the training data, problem *features* are also essential. Features are the input of the prediction function and should describe the relevant attributes of the datapoint. A datapoint can have several possible *feature representations*. *Feature engineering* concerns itself with identifying relevant features [28]. To simplify computations, most machine learning algorithms require that the features are a (sparse) real-valued vector. Potential features are listed below.

Spam classification A list of all the words occurring in the email
Face detection The matrix containing the color values of the pixels
Web search The *n*-grams of the query

From a mathematical point of view, most machine learning problems can be reduced to an optimization problem. Let $D \subseteq X \times T$ be a dataset consisting of datapoints and their corresponding target values. Let $\varphi : X \to \mathfrak{F}$ be a feature function that maps a datapoint to its feature representation in the *feature space* $\mathfrak{F}$ (usually a subset of $\mathbb{R}^n$ some $n \in \mathbb{N}$). Furthermore, let F be a function space and s a (convex) score function $s : D \times F \to \mathbb{R}$. Elements of F map features to the target space T—i.e., $F \subseteq (\mathfrak{F} \to T)$. One possible goal is to find the function $f \in F$ that maximizes the average score over the training set D. The main differences between various learning algorithms are the function space and the score function they use.

If the function space is too expressive, *overfitting* may occur: The learned function might perform well on the training data, but poorly on unseen data. A simple example is trying to fit a polynomial of degree $n-1$ through n training datapoints; this will give perfect scores on the training data but is likely to yield a curve that behaves so wildly as to be useless to make predictions. The issue is well known from the world of finance, where very sophisticated models have been successfully applied to predict the past.

Regularization is used to balance function complexity with the result of the score function. To estimate how well a learning algorithm generalizes or to tune metaparameters (e.g., the regularization parameter or the maximum degree of a polynomial), *cross-validation* partitions the training data in two sets: one set used for training, the other for the evaluation.

3 Axiom Selection as a Machine-Learning Problem

Using an ATP within an ITP requires a method to filter out irrelevant axioms during the creation of the ATP problem. Since most ITPs libraries contain several thousands of theorems, simply translating every library fact into an ATP axiom overwhelms the ATP; indeed, parsing huge problem files has been an issue with some ATPs. To use machine learning to create such a relevance filter, we must first answer three questions:

1. What is the goal of the learning?
2. What is the training data?
3. What are the features?

At a first glance, the goal seems clear:

Given an ATP problem with axioms A and conjecture c,
predict a subset of axioms $B \subseteq A$ that is sufficient for proving c.

But what does "sufficient" mean exactly? Clearly, the ATP's chances of success depend on which ATP is used, the time limit, and even the computer hardware. Moreover, there can be several potentially disjoint subsets from which the conjecture can be derived, reflecting the existence of alternative proofs. Which subset should then be chosen? Before we can answer these questions, we must introduce the training data.

3.1 The Training Data

The training data is extracted from the proof library of the ITP. For Isabelle, this could mean the libraries included with the prover or the *Archive of Formal Proofs* [23]; for Mizar, the *Mizar Mathematical Library* [30]. The data could also include custom libraries defined by the user or third parties.

Abstracting from its source, we assume that the training data consists of a set of facts (axioms, definitions, lemmas, theorems, corollaries) equipped with

– a *visibility relation* that for each fact states which other facts appear before it;
– a *dependency tree* that for each fact shows which facts were used in its proof (for lemmas, theorems, and corollaries);
– a *formula tree representation* of each fact.

Example. Figure 4 introduces a simple, constructed library. For each statement, every statement that occurs above it is visible. Axioms 1 and 2 and Definitions 1 and 2 are visible from Theorem 1, whereas Corollary 1 is not visible. Figure 5 presents the corresponding dependency tree. Finally, Figure 6 shows the formula tree of $\forall x\, x+1 > x$.

Axiom 1. *A*

Axiom 2. *B*

Definition 1. *C* if and only if *A*

Definition 2. *D* if and only if *C*

Theorem 1. *C*

Proof. By Axiom 1 and Definition 1.

Corollary 1. *D*

Proof. By Theorem 1 and Definition 2.

Figure 4: A simple library

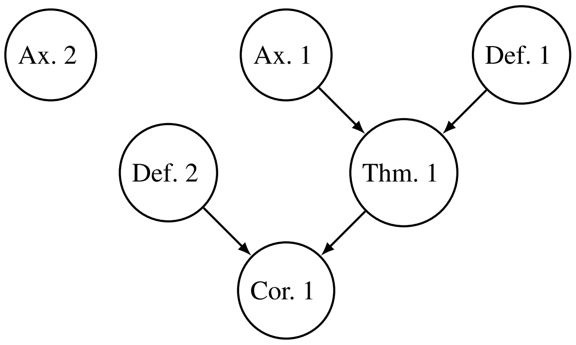

Figure 5: The dependency tree of the library of Figure 4, where edges denote dependency between facts

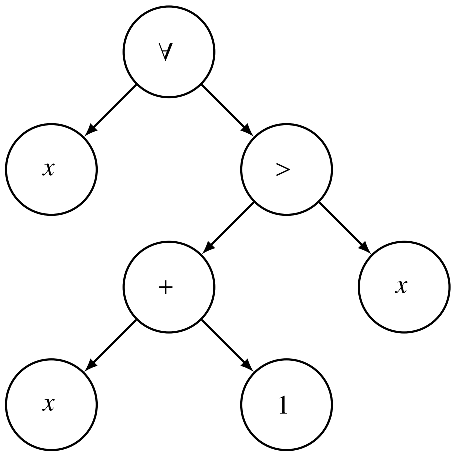

Figure 6: The formula tree for $\forall x\, x+1 > x$

3.2 What to Learn

Having defined the training data, we can now reconsider the initial learning goal:

> Given an ATP problem with axioms A and conjecture c,
> predict a subset of axioms $B \subseteq A$ that is sufficient for proving c.

In the ATP-as-tactic setting, the conjecture of the corresponding ATP problem is the current proof obligation the ITP user wants to discharge and the axioms are the visible facts. Since the score function only needs to be defined on the training dataset, the dependency tree can be used to define which axioms are sufficient. This allows us the restate the learning goal as follows:

> Given an ITP proof obligation c,
> predict the parents of c in the dependency tree.

For now, we ignore alternative proofs and assume that the dependencies extracted from the ITP are the dependencies that an ATP would use. Of course, predicting the exact parents is unrealistic. Treating axiom selection as a ranking rather than a subset selection problem allows more room for error and hence simplifies the problem. With this adjustment, we can state the final version of our learning goal:

> Given a training dataset and the formula tree of a proof obligation (Section 3.1),
> rank the visible facts according to their predicted usefulness.

In the training phase, the learning algorithm is allowed to learn from the proofs of all visible facts. The score function is chosen to optimize the ranks of the proof dependencies. For all facts in the training set, their corresponding dependencies should be ranked as high as possible.

It has often been observed that it is better to invoke an ATP repeatedly with different options for a short period of time (e.g., 5 seconds) than to let it run undisturbed until the user stops it. This optimization is called *time slicing*. Having a ranking function makes it possible to create different ATP problems for different slices, each with a different number of axioms, as illustrated in Figure 7. Slices with few axioms are more likely to find complex proofs involving a few obvious axioms, whereas those with lots of axioms might find simple proofs involving more obscure axioms.

3.3 Features

Almost all learning algorithms require the features of the input data to be a real vector. Therefore a method is needed to translate formula trees representing a proof obligation into real vectors.

Symbols. A simple approach is to take the set of symbols of a formula as its feature set. The symbols correspond to the node labels in the formula tree. It usually makes sense to leave out symbols corresponding to variables, since variable names are immaterial.

Let n denote the vector size, which should be at least as large as the total number of symbols in the library. Let i be an injective index function that maps each symbol s to

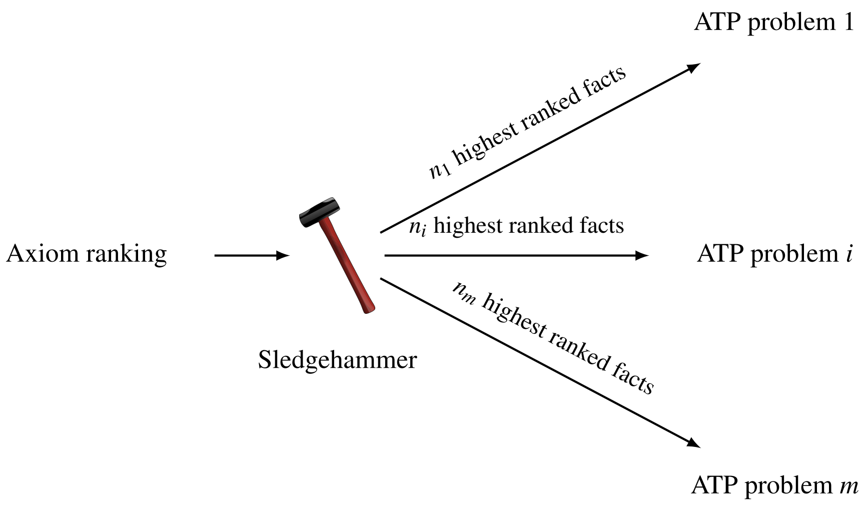
ATP problem 1
n_1 highest ranked facts
Axiom ranking
n_i highest ranked facts
ATP problem i
n_m highest ranked facts
Sledgehammer
ATP problem m

Figure 7: Sledgehammer generates several ATP problems (slices) from a single ranking

a positive number $i(s) \leq n$. The feature representation of a formula tree t is the binary vector $\varphi(t)$ such that $\varphi(t)(j) = 1$ if and only if the symbol with index j appears in t.

The example formula tree in Figure 6 contains the symbols $\forall$, $>$, $+$, and 1 (but not the variable x). Given $n = 10$, $i(\forall) = 1$, $i(>) = 4$, $i(+) = 6$, and $i(1) = 7$, the corresponding feature vector is $(1,0,0,1,0,1,1,0,0,0)$.

Subterms and subformulas. In addition to the symbols, one can also include as features the subterms and subformulas of the formula to prove—i.e., the subtrees of the formula tree [47]. For example, the formula tree in Figure 6 has subtrees associated with x, 1, $x+1$, $x > x+1$, and $\forall x\; x+1 > x$. Adding all subtrees significantly increases the size of the feature vector. Many subterms and subformulas appear only once in the library and are hence useless for making predictions. An approach to curtail this explosion is to consider only small subtrees (e.g., those with a height of at most 2 or 3).

Types. The formalisms supported by the vast majority of ITP systems are typed (or sorted), meaning that each term can be given a type that describes the values that can be taken by the term. Examples of types are *int*, *real*, *real* × *real*, and *real* → *real*. Adding the types that appear in the formula tree as additional features can help [18, 26]. Like terms, types can be represented as trees, and we may choose between encoding only basic types or also some or all complex subtypes.

Context. Due to the way humans develop complex proofs, the last few facts that were proved are likely to be useful in a proof of the current goal [7]. However, the machine learning algorithm might rank them poorly because they are new and hence little used, if at all. Adding the feature vectors of the nearby facts to the feature vector of the proof obligation, in a weighted fashion, is a method for ensuring that they obtain a better rank. This method is particularly useful when a formula has very few or very general features but occurs in a wider context.

4 Challenges

Axiom selection has several peculiarities that restrict which machine learning algorithms can be effectively used. In this section, we illustrate these challenges on a large fragment of Isabelle's *Archive of Formal Proofs* (AFP). The AFP benchmarks contain 165 964 facts distributed over 116 entries contributed by dozens of Isabelle users.[4] Most entries are related to computer science (e.g., data structures, algorithms, programming languages, and process algebras). The dataset was generated using Sledgehammer [26] and is available publicly at `http://www.cs.ru.nl/~kuehlwein/downloads/afp.tar.gz`.

4.1 Features

The features introduced in Section 3.3 are very sparse. For example, the AFP contains 20 461 symbols. Adding small subterms and subformulas as well as basic types raises the total number of features to 328 361. Rare features can be very useful, because if two facts share a very rare feature, the likelihood that one depends on the other is very high. However, they also lead to much larger and sparser feature vectors.

Figure 8 shows the percentage of features that appear in at least x facts in the AFP, for various values of x. If we consider all features, then only 3.37% of the features appear in more than 50 facts. Taking only the symbols into account gives somewhat less sparsity, with 2.65% of the symbols appearing in more than 500 facts. Since there are 165 964 facts in total, this means that 97.35% of all symbols appear in less than 0.3% of the training data.

Another peculiarity of the axiom selection problem is that the number of features is not a priori fixed. Introducing new names for new concepts is standard mathematical practice. Hence, the learning algorithm must be able to cope with an unbounded, ever increasing feature set.

4.2 Dependencies

Like the features, the dependencies are also sparse. On average, an AFP fact depends on 5.5 other facts; 19.4% of the facts (axioms and definitions) have no dependencies at all, and 10.7% have at least 20 dependencies. Figure 9 shows the percentage of facts that are dependencies of at least x facts in the AFP, for various values of x. Less than half of the facts (43.0%) are a dependency in at least one other fact, and 94 593 facts are never used as dependencies. This includes 32 259 definitions as well as 17 045 facts where the dependencies could not be extracted and were hence left empty. Only 0.08% of the facts are being used as dependencies more than 500 times.

[4] A number of AFP entries were omitted because of technical difficulties.

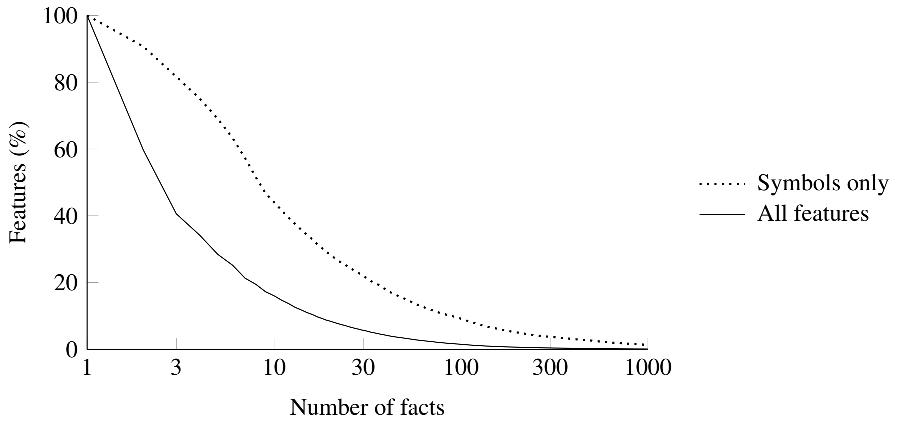

Figure 8: Distribution of the feature appearances in the *Archive of Formal Proofs*

The main issue is that the dependencies in the training data might be incomplete or otherwise misleading. The dependencies extracted from the ITP are not necessarily the same as an ATP would use [1]. For example, Isabelle users can use induction in an interactive proof, and this would be reflected in the dependencies—the induction principle is itself a (higher-order) fact. But ATPs are limited to first-order logic without induction. If an alternative first-order proof is possible, this is the one that should be learned. Experiments with combinations of ATP and ITP proofs indicate that ITP dependencies are a reasonable guess, but learning from ATP dependencies yields better results [25, 47].

More generally, the training data lacks information about alternative proofs. In practice, this means that any evaluation method that relies only on the training data cannot reliably evaluate whether an axiom selection algorithm produces good predictions. There is no choice but to actually run ATPs—and even then the hardware, time limit, and version of the ATP can heavily influence the results.

4.3 Online Learning and Speed

Any algorithm for axiom selection must update its predictions model and create predictions fast. The typical use case is that of an ITP user who develops a theory fact by fact, proving each along the way. Usually these facts depend on one another, often in the familiar sequence definition–lemma–theorem–corollary. After each user input, the prediction model might need to be updated. In addition, it is not uncommon for users to alter existing definitions or lemmas, which should trigger some relearning.

Speed is essential for a axiom selection algorithm since the automated proof finding process needs to be faster than manual proof creation. The less time is spent on updating the learning model and predicting the axiom ranking, the more time can be used by ATPs. Users of ITPs tend to be impatient: If the automatic provers do not respond within half a minute or so, they usually prefer to carry out the proof themselves.

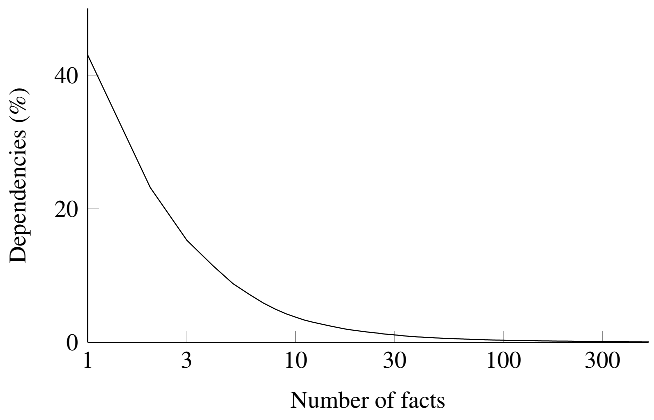

Figure 9: Distribution of the dependency appearances in the *Archive of Formal Proofs*

4.4 Translations between Logics

There is a wide gap between the ITPs' and the ATPs' logics. Much research has been concerned with bridging the gap between the two, by encoding ITP into ATP formulas. The main difficulties are connected with the ITPs' support for higher-order construct (e.g., quantification over functions and predicates, λ-expressions) and rich polymorphic type systems. Complete translations of both features are well known, but they lead to so much clutter that the proof search effectively grinds to a halt.

In practice, interactive problems are mostly first-order and their type information is largely irrelevant. This can be exploited to yield a lightweight translation, by encoding higher-order constructs locally (and leaving the first-order parts of the problem unchanged) [32] and by keeping a minimal amount of type information necessary to prevent the discovery of spurious proofs (i.e., proofs that are ill-typed in the ITP) [5].

5 Conclusion

This paper provided an introduction to the axiom selection problem. For further reading, we refer to Kühlwein et al. [27], which reviews several algorithms on a benchmark suite derived from the *Mizar Mathematical Library*, and Kaliszyk and Urban [19,20], which introduced a nearest-neighbor approach to axiom selection. Urban and Vyskočil give a more results-oriented introduction [48].

Machine learning has also been employed to improve other aspects of ATP reasoning. In particular, learning algorithms have been used to predict which search strategies are most likely to succeed in finding a proof [17,24,45].

Acknowledgments. The first author is supported by the Deutsche Forschungsgemeinschaft (DFG) project Hardening the Hammer (grant Ni 491/14-1). The second author is supported by the Nederlandse organisatie voor Wetenschappelijk Onderzoek (NWO) project Learning2Reason.

References

1. Alama, J., Kühlwein, D., Urban, J.: Automated and human proofs in general mathematics: An initial comparison. In: Bjørner, N., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning (LPAR-18), Lecture Notes in Computer Science, vol. 7180, pp. 37–45. Springer (2012)
2. Ball, T., Bounimova, E., Levin, V., Kumar, R., Lichtenberg, J.: The Static Driver Verifier Research Platform. In: Touili, T., Cook, B., Jackson, P. (eds.) Computer Aided Verification (CAV 2010), Lecture Notes in Computer Science, vol. 6174, pp. 119–122. Springer (2010)
3. Bertot, Y., Castéran, P.: Interactive Theorem Proving and Program Development—Coq'Art: The Calculus of Inductive Constructions. Texts in Theoretical Computer Science, Springer (2004)
4. Bishop, C.M.: Pattern Recognition and Machine Learning. Information Science and Statistics, Springer (2006)
5. Blanchette, J.C., Böhme, S., Popescu, A., Smallbone, N.: Encoding monomorphic and polymorphic types. In: Piterman, N., Smolka, S. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2013). Lecture Notes in Computer Science, vol. 7795, pp. 493–507. Springer (2013)
6. Blanchette, J.C., Böhme, S., Paulson, L.C.: Extending Sledgehammer with SMT solvers. Journal of Automated Reasoning 51(1), 109–128 (2013)
7. Cramer, M., Koepke, P., Kühlwein, D., Schröder, B.: Premise selection in the Naproche system. In: Giesl, J., Hähnle, R. (eds.) International Joint Conference on Automated Reasoning (IJCAR 2010), Lecture Notes in Computer Science, vol. 6173, pp. 434–440. Springer (2010)
8. Frege, G.: Begriffsschrift, eine der arithmetischen nachgebildete Formelsprache des reinen Denkens. Verlag von Louis Nebert (1879)
9. Gödel, K.: Über formal unentscheidbare Sätze der Principia Mathematica und verwandter Systeme I. Monatshefte für Mathematik und Physik 38(1), 173–198 (1931)
10. Gonthier, G.: Formal proof—The four-color theorem. Notices of the AMS 55(11), 1382–1393 (2008)
11. Gonthier, G., Asperti, A., Avigad, J., Bertot, Y., Cohen, C., Garillot, F., Roux, S.L., Mahboubi, A., O'Connor, R., Biha, S.O., Pasca, I., Rideau, L., Solovyev, A., Tassi, E., Théry, L.: A machine-checked proof of the odd order theorem. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving (ITP 2013). Lecture Notes in Computer Science, vol. 7998, pp. 163–179. Springer (2013)
12. Grabowski, A., Korniłowicz, A., Naumowicz, A.: Mizar in a Nutshell. Journal of Formalized Reasoning 3(2), 153–245 (2010)
13. Hales, T.C.: Introduction to the Flyspeck project. In: Coquand, T., Lombardi, H., Roy, M.F. (eds.) Mathematics, Algorithms, Proofs. Dagstuhl Seminar Proceedings, vol. 05021. Schloss Dagstuhl (2005)
14. Harrison, J.: HOL Light: A tutorial introduction. In: Srivas, M., Camilleri, A. (eds.) Formal Methods in Computer-Aided Design (FMCAD '96). Lecture Notes in Computer Science, vol. 1166, pp. 265–269. Springer (1996)
15. Harrison, J.: Formal verification of IA-64 division algorithms. In: Aagaard, M., Harrison, J. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2000), Lecture Notes in Computer Science, vol. 1869, pp. 233–251. Springer (2000)
16. Hoder, K., Voronkov, A.: Sine qua non for large theory reasoning. In: Bjørner, N., Sofronie-Stokkermans, V. (eds.) Conference on Automated Deduction (CADE-23), Lecture Notes in Computer Science, vol. 6803, pp. 299–314. Springer (2011)
17. Hutter, F., Hoos, H.H., Leyton-Brown, K., Stützle, T.: ParamILS: An automatic algorithm configuration framework. Journal of Artificial Intelligence Research 36, 267–306 (2009)

18. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. CoRR abs/1211.7012 (2012)
19. Kaliszyk, C., Urban, J.: Automated reasoning service for HOL Light. In: Carette, J., Aspinall, D., Lange, C., Sojka, P., Windsteiger, W. (eds.) Conferences on Intelligent Computer Mathematics (CICM 2013), Lecture Notes in Computer Science, vol. 7961, pp. 120–135. Springer (2013)
20. Kaliszyk, C., Urban, J.: Stronger automation for Flyspeck by feature weighting and strategy evolution. In: Blanchette, J.C., Urban, J. (eds.) Proof Exchange for Theorem Proving (PxTP 2013). EPiC, vol. 14, pp. 87–95. EasyChair (2013)
21. Kaufmann, M., Manolios, P., Moore, J.S.: Computer-Aided Reasoning: An Approach. Kluwer Academic Publishers (2000)
22. Klein, G., Andronick, J., Elphinstone, K., Heiser, G., Cock, D., Derrin, P., Elkaduwe, D., Engelhardt, K., Kolanski, R., Norrish, M., Sewell, T., Tuch, H., Winwood, S.: seL4: formal verification of an operating-system kernel. Communications of the ACM 53(6), 107–115 (2010)
23. Klein, G., Nipkow, T., Paulson, L. (eds.): Archive of Formal Proofs. `http://afp.sf.net/`
24. Kühlwein, D., Urban, J.: MaLeS: A framework for automatic tuning of automated theorem provers. CoRR abs/1308.2116 (2013)
25. Kühlwein, D., Urban, J.: Learning from multiple proofs: First experiments. In: Fontaine, P., Schmidt, R.A., Schulz, S. (eds.) Practical Aspects of Automated Reasoning (PAAR 2012). EPiC, vol. 21, pp. 82–94. EasyChair (2013)
26. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: MaSh: Machine learning for sledgehammer. In: Blazy, S., Paulin-Mohring, C., Pichardie, D. (eds.) Interactive Theorem Proving (ITP 2013), Lecture Notes in Computer Science, vol. 7998, pp. 35–50. Springer (2013)
27. Kühlwein, D., Laarhoven, T., Tsivtsivadze, E., Urban, J., Heskes, T.: Overview and evaluation of premise selection techniques for large theory mathematics. In: Gramlich, B., Miller, D., Sattler, U. (eds.) International Joint Conference on Automated Reasoning (IJCAR 2012), Lecture Notes in Computer Science, vol. 7364, pp. 378–392. Springer (2012)
28. Liu, H., Motoda, H.: Feature Selection for Knowledge Discovery and Data Mining. Kluwer Academic Publishers (1998)
29. MacKay, D.J.: Information Theory, Inference and Learning Algorithms. Cambridge University Press (2003)
30. Matuszewski, R., Rudnicki, P.: Mizar: The first 30 years. Mechanized Mathematics and Its Applications 4, 3–24 (2005)
31. McCune, W.: Solution of the Robbins problem. Journal of Automated Reasoning 19(3), 263–276 (1997)
32. Meng, J., Paulson, L.C.: Translating higher-order clauses to first-order clauses. Journal of Automated Reasoning 40(1), 35–60 (2008)
33. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. Journal of Applied Logic 7(1), 41–57 (2009)
34. Moore, J.S., Lynch, T.W., Kaufmann, M.: A mechanically checked proof of the AMD5_K86^{TM} floating point division program. IEEE Transactions on Computers 47(9), 913–926 (1998)
35. Moura, L., Bjørner, N.: Z3: An efficient SMT solver. In: Ramakrishnan, C., Rehof, J. (eds.) Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008), Lecture Notes in Computer Science, vol. 4963, pp. 337–340. Springer (2008)
36. Murphy, K.P.: Machine Learning: A Probabilistic Perspective. MIT Press (2012)
37. 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)
38. Owre, S., Shankar, N.: A brief overview of PVS. In: Theorem Proving in Higher Order Logics (TPHOLs 2008). Lecture Notes in Computer Science, vol. 5170, pp. 22–27. Springer (2008)

39. Riazanov, A., Voronkov, A.: The design and implementation of VAMPIRE. AI Communications 15(2-3), 91–110 (Aug 2002)
40. Schulz, S.: E—A Brainiac Theorem Prover. AI Communications 15(2-3), 111–126 (2002)
41. Slind, K., Norrish, M.: A Brief Overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) Theorem Proving in Higher Order Logics (TPHOLs 2008), Lecture Notes in Computer Science, vol. 5170, pp. 28–32. Springer (2008)
42. Sutcliffe, G.: The TPTP problem library and associated infrastructure. Journal of Automated Reasoning 43(4), 337–362 (2009)
43. Sutcliffe, G.: The 6th IJCAR automated theorem proving system competition—CASC-J6. AI Communications 26(2), 211–223 (2013)
44. Urban, J.: MPTP—motivation, implementation, first experiments. Journal of Automated Reasoning 33(3-4), 319–339 (2004)
45. Urban, J.: BliStr: The blind strategymaker. CoRR abs/1301.2683 (2013)
46. Urban, J., Rudnicki, P., Sutcliffe, G.: ATP and presentation service for Mizar formalizations. Journal of Automated Reasoning 50(2), 229–241 (2013)
47. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLARea SG1—Machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) International Joint Conference on Automated Reasoning (IJCAR 2008), Lecture Notes in Computer Science, vol. 5195, pp. 441–456. Springer (2008)
48. Urban, J., Vyskočil, J.: Theorem proving in large formal mathematics as an emerging AI field. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics—Essays in Memory of William W. McCune, Lecture Notes in Computer Science, vol. 7788, pp. 240–257. Springer (2013)
49. Weidenbach, C., Dimova, D., Fietzke, A., Kumar, R., Suda, M., Wischnewski, P.: SPASS version 3.5. In: Schmidt, R.A. (ed.) Conference on Automated Deduction (CADE-22). Lecture Notes in Computer Science, vol. 5663, pp. 140–145. Springer (2009)
50. Wenzel, M., Wiedijk, F.: A comparison of Mizar and Isar. Journal of Automated Reasoning 29(3-4), 389–411 (2002)
51. Whitehead, A.N., Russell, B.: Principia Mathematica (Second Edition). Cambridge University Press (1927)