Statistical Proof Pattern Recognition: Automated or Interactive?*

Jónathan Heras Ekaterina Komendantskaya

School of Computing, University of Dundee, UK
{jonathanheras,katya}@computing.dundee.ac.uk

Abstract: In this paper, we compare different existing approaches employed in data mining of big proof libraries in automated and interactive theorem proving.

1 Motivation

Over the last few decades, theorem proving has seen major developments. *Automated (first-order) theorem provers (ATPs)* (e.g. E, Vampire, SPASS) and SAT/SMT solvers (e.g. CVC3, Yices, Z3) are becoming increasingly fast and efficient. *Interactive (higher-order) theorem provers (ITPs)* (e.g. Coq, Isabelle/HOL, AGDA, Mizar) have been enriched with dependent types, (co)inductive types, type classes and provide rich programming environments.

The main conceptual difference between ATPs and ITPs lies in the styles of proof development. For ATPs, the proof process is primarily an automatically performed *proof search* in *first-order* language. In ITPs, the proof steps are suggested by *the user* who guides the prover by providing the tactics. ITPs work with *higher-order* logic and type theory, where many derivation algorithms (e.g. higher-order unification) are inherently undecidable.

Communities working on development, implementation and applications of ATPs and ITPs have accumulated big corpora of electronic proof libraries. However, the size of the libraries, as well as their technical and notational sophistication often stand on the way of efficient knowledge re-use. Very often, it is easier to start a new library from scratch rather than search the existing proof libraries for potentially common heuristics and techniques. Proof-pattern recognition is the area where statistical machine-learning is likely to make an impact. Here, we discuss and compare two different styles of proof-pattern recognition.

We will use the following convention: the term "goal" will stand for an unproven proposition in the language of a given theorem prover; the term "lemma" will refer to an already proven proposition in the library.

2 Proof-pattern recognition in ATPs

Given a proof goal, ATPs apply various lemmas to rewrite or simplify the goal until it is proven. The order in which different lemmas are used plays a big role in speed and efficiency of the automated proof search. Hence, machine-learning techniques can be used to improve the premise selection procedure on the basis of previous experience acquired from successful proofs; cf. [2, 6] .

The technical details of such machine-learning solutions would differ [3, 4, 5, 6], but we can summarise the common features of this approach, as follows:

1. Feature extraction:
• The features are extracted from first-order formulas (given by lemmas and goals). For every proposition (goal or lemma), the associated binary feature vector records, for every symbol and term of the library, whether it is present or absent in the proposition. As a result, the feature vectors grow to be as long as 10^6 features long.
• After the features are extracted, the machine-learning tool constructs a classifier (e.g. SVM) for every lemma of the library. For two lemmas A and B, if B was used in the proof of A, a feature vector $|A|$ is sent as a positive example to the classifier $< B >$, else $|A|$ is considered to be a negative example.

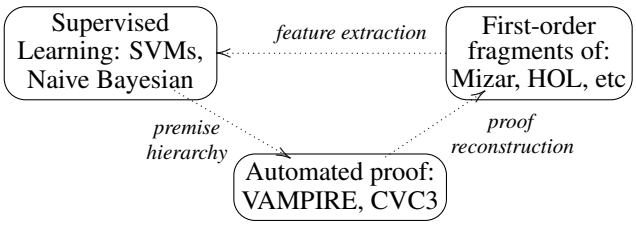
2. Machine-learning tools:
• Every classifier $< B >$ has its set of positive and negative examples, hence *supervised learning* is used for training.
• The classifier algorithms [3, 4, 5, 6] range from SVMs with various kernel functions to Naive Bayes learning.
• Feature vectors are too big for traditional machine-learning algorithms to tackle, and the special software *SNoW* is used to deal with the over-sized feature vectors.
• The output of machine-learning algorithm provides a "rank" of formula lying in the interval $[0, 1]$, where increasing values mean increased probability that B is used in the proof of A.

3. The mode of interaction between the prover and machine-learning tool:
• Given a new goal G, the feature vector $|G|$ is sent to the previously trained classifier $< L >$, for every Lemma L of the given library. The classifier $< L >$ then outputs a rank showing how useful lemma L can be in the proof of G.
• Once the ranking is computed, it is used to decide, for every lemma in the library, whether it should be used in the new proof.

4. Main improvement: the number of goals proven automatically increases by up to 20% - 40%, depending on the prover and the library in question.

Note that, if an ITP uses ATP tools to speed up the proof of first-order lemmas, the method above can be used to speed up the automated proof search, [2, 6]. The following figure shows this scheme of using machine-learning in ATPs and ITPs:

*The work was supported by EPSRC grant EP/J014222/1.



3 Proof-pattern recognition in ITPs

Interactive style of theorem proving differs significantly from that of ATPs. In particular, a given ITP will necessarily depend on user instructions (e.g. in the form of tactics). Because of the inherently interactive nature of proofs in ITPs, user interfaces play an important role in the proof development. One example is *Proof General* – a general-purpose, emacs-based interface created for communication with a range of higher-order theorem provers. In this setting, machine-learning algorithms need to gather statistics from the user's behaviour, and feed the results back to the user *during* the proof process. Proof-pattern recognition must become an integral part of the user interface. The first tool achieving this is ML4PG [1].

Similar interfacing trend exists in the machine learning community. Since users need to monitor results computed by the statistical tools, the community has developed uniform interfaces (*Matlab, Weka*) – environments in which the user can choose which algorithm to use for processing the data and for interpreting the results. ML4PG integrates a range of machine-learning algorithms provided by Matlab and Weka into the Proof General interface.

Comparing with the ATP-based machine-learning tools, ML4PG can be characterised as follows:

1. Feature extraction:

• The features are extracted directly from higher-order propositions and proofs.

• Feature extraction is built on the method of *proof-traces*: the structure of the higher-order proposition is captured by analysing several proof steps the user takes when proving it, this includes the statistics of tactics, tactic arguments, tactic argument types, top symbols of formulas and number of generated subgoals, see [1].

• The feature vectors are fixed at the size of 30. This size is manageable for literally any existing statistical machine-learning algorithm.

• Longer proofs are analysed by means of the *proof-patch* method: when features of one big proof are collected by taking a collection of features of smaller proof fragments.

2. Machine-learning tools:

• As higher-order proofs in general can take a variety of shapes, sizes and proof-styles, ML4PG does not use any *a priori* given training labels. Instead, it uses unsupervised learning (*clustering*), and in particular, Gaussian, k-means, and farthest-first algorithms.

• The output of clustering algorithm provides proof families based on some user defined parameters – e.g. cluster size, and proximity of lemmas within the cluster.

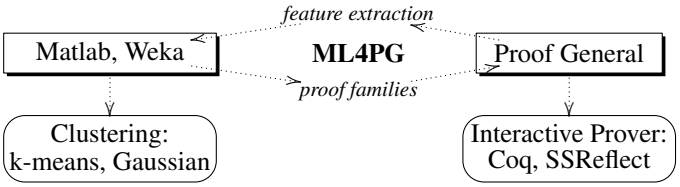
3. The mode of interaction between the prover and machine-learning tool:

• ML4PG works on the background of Proof General, and extracts the features interactively in the process of Coq compilation.

• On user's request, it sends the gathered statistics to a chosen machine-learning interface and triggers execution of a clustering algorithm of the user's choice, using adjustable user-defined clustering parameters.

• ML4PG does some gentle post-processing of the results given by the machine-learning tool, and displays families of related proofs to the user.

4. Main improvement: ML4PG makes use of the rich interfaces in ITPs and machine learning. It assists the user, rather than the prover: the user may treat the suggested similar lemmas as proof hints. The interaction with ML4PG is fast and easy, so the user may receive these hints interactively, and in real time. The process is summarised below:



4 Conclusions

The automated and interactive styles of proof-pattern recognition described here have been successfully applied in big proof libraries in Mizar, HOL, Isabelle, Coq, and SSReflect. The methods complement each other: one aims to speed up the first-order proofs, and the other one provides guidance where proofs cannot be fully automated.

References

[1] J. Heras and E. Komendantskaya. ML4PG: downloadable programs, manual, examples, 2012–2013. `www.computing.dundee.ac.uk/staff/katya/ML4PG/`.

[2] C. Kaliszyk and J. Urban. Learning-assisted Automated Reasoning with Flyspeck, 2012. arXiv:1211.7012.

[3] D. Kühlwein et al. Overview and evaluation of premise selection techniques for large theory mathematics. In *IJCAR'12*, LNCS 7364, pages 378–392, 2012.

[4] E. Tsivtsivadze et al. Semantic graph kernels for automated reasoning. In *SDM'11*, pages 795–803, 2011.

[5] J. Urban. MPTP 0.2: Design, Implementation, and Initial Experiments. *JAR*, 37(1-2):21–43, 2006.

[6] J. Urban et al. Malarea sg1- machine learner for automated reasoning with semantic guidance. In *IJCAR'08*, LNCS 5195, pages 441–456, 2008.