Proof Guidance in PVS with Sequential Pattern Mining

M. Saqib Nawaz[1], Meng Sun[1] and Philippe Fournier-Viger[2]

[1]LMAM & Department of Informatics, School of Mathematical Sciences,
Peking University, Beijing, China
[2]School of Humanities and Social Sciences,
Harbin Institute of Technology (Shenzhen), Shenzhen, China
{msaqibnawaz, sunm}@pku.edu.cn, philfv8@yahoo.com

Abstract. The recent introduction of the big data paradigm and advancements in machine learning and deep mining techniques have made proof guidance and automation in interactive theorem provers (ITPs) an important research topic. In this paper, we provide a learning approach based on sequential pattern mining (SPM) for proof guidance in the PVS proof assistant. Proofs in a PVS theory are first abstracted to a computer-processable corpus. SPM techniques are then used on the corpus to discover frequent proof steps and proof patterns, relationships of proof steps / patterns with each other, dependency of new conjectures on already proved facts and to predict the next proof step(s). Obtained results suggest that the integration of SPM in proof assistants can be used to guide the proof process and in the development of proof tactics / strategies.

Keywords: PVS, Proof development process, Proof corpus, Frequent patterns, Sequential pattern mining.

1 Introduction

Theorem provers allow the formal development and verification of system properties that can be defined in appropriate logical formalisms. Automated (first-order) theorem provers (ATPs) deal with the development of computer programs that can automatically perform logical reasoning. However, first-order logic (FOL) lacks the expressibility power that is required to define complex systems with an infinite domain. On the other hand, higher-order logic (HOL) allows quantification over predicates and functions. HOL based theorem provers, also known as interactive theorem provers (ITPs), offer support for rich logical formalisms such as dependent and (co)inductive types as well as recursive functions, which enable ITPs to model complex systems. Today, these mechanical reasoning systems are used in verification projects that range from operating systems, compilers and hardware components to prove the correctness of large mathematical proofs such as the Feit-Thomson Theorem and the Kepler conjecture [22]. However, automatic reasoning in ITPs is still a hard problem due to undecidable algorithms and proof methods [20].

Unlike ATPs where the proof process is generally automatic, ITPs follow the user driven proof development process. The user guides the proof process by providing the proof goal and by applying proof commands and tactics to prove the goal. Generally, the user does lots of repetitive work to prove a non-trivial theorem (goal), which is laborious and consumes a large amount of time. Proof guidance and proof automation in ITPs are two extremely desirable features. ITPs now do have a large corpora of computer-understandable formalized knowledge [5, 19] in the form of proof libraries. In PVS, proof scripts for a particular theory are stored separately in a file that can also be considered as a proof corpus for the theorems and lemmas in that theory. Proof scripts of different theories can be combined together to develop a more complex corpora. These corpora play an important role in artificial intelligence based methods, such as concept matching, structure formation and theory exploration. The ongoing fast progress in machine learning and data mining made it possible to use these learning techniques on such corpora in guiding the proof search process, in proof automation and in developing proof tactics / strategies, as indicated in recent works [8, 9, 15–17, 21, 26].

In this paper, the focus is on proof guidance and premise selection in ITPs from the perspective of sequential pattern mining (SPM) techniques. SPM techniques are used in data mining to find interesting and useful patterns (information) that are hidden in large corpora of sequential data [14]. A particular proof goal in PVS depends on the specifications inside the theory and it can be completed with different combinations of proof commands, inference rules and decision procedures [30]. This makes it difficult to infer useful proof tactics and strategies from specific examples that can be applied more generally. Moreover, a proof corpus contains too much information, which makes it hard to carry out the brute force approach for proof search. However, there is the potential to identify useful and interesting hidden proof patterns in these corpora and relationships of such proof patterns with each other. With such information, SPM techniques can be used to investigate the dependency of new conjectures on already proved facts and to predict the next proof step(s) or pattern(s) for guiding the proof of a novel non-trivial theorem / lemma.

We present an SPM-based proof process learning approach for the PVS proof assistant. The basic idea is to convert the PVS proofs for a theory into a proof corpus that is suitable for learning. SPM techniques are applied on the corpus to find frequent proof steps and patterns that are used in the proofs. Moreover, relationships of a proof steps / patterns with each other are discovered through sequential rule mining. The learning approach is also used to find the relevance of the new conjectures with the proofs and the performance of some state-of-the-art prediction models are examined by training and testing them on the corpus to predict the next proof step(s). Besides PVS, the proposed approach can also be used to guide the proof process in other proof assistants. The ultimate goal is to develop proof tactics / strategies with useful patterns that can be invoked directly by the user in the proof development process.

The rest of this paper is organized as follows: Section 2 elaborates the SPM-based learning approach that is used to discover useful proof steps / patterns in the proof corpus, their relationship and prediction of next proof step(s). Evaluation of the proposed approach on a case study and obtained results are discussed in Section 3. Related work on using the machine learning and data mining techniques for automated reasoning in ATPs and ITPs is presented in Section 4. Finally, the paper is concluded with some future directions in Section 5. PVS dump files and SPM related data for this work can be found at [31].

2 Proof Corpus Mining with SPM

The structure of the SPM-based learning approach is shown in Figure 1. It consists of two main parts:

1. Development of proof corpus: PVS proof steps for theorems and lemmas in a theory are converted to a proof corpus, where each complete proof is abstracted to sequences of proof commands.
2. Learning through SPM: SPM algorithms are used on the corpus to discover the common proof steps and patterns, relationships of proof steps / patterns with each other, dependency of new conjectures on already proved facts and prediction of next proof step(s). Each part is further elaborated next.

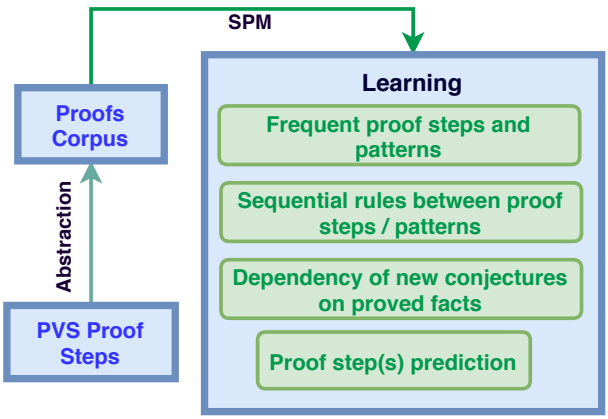

Fig. 1. SPM-based approach to learn the proof process

In general, data is assembled first, so that data mining algorithms can be used. To make the proof corpus suitable for learning, it should satisfy certain minimum requirements, such as:

- It is stored in a computational and electronic form.
- It contains many examples of proofs that offer diversity in kinds of proof steps. The corpus should have different proof steps so that useful proof patterns as well as the dependency of proof steps and prediction of next proof steps can be discovered.

– It is transformed in a suitable abstraction, so that no meaningful information from the proofs is left out. For this, we use the "*proof sequences to integers*" abstraction, where each proof command is converted to a distinct positive integer. Such abstraction allows wide diversity and makes the approach more general in nature.

Besides the dump file that contains the specifications for a particular theory with imported libraries and proof scripts (collection of proof steps) for theorems / lemmas, PVS also saves the proof scripts for a theory in a separate proof file. These files contain proof commands with some other information related to PVS. After removing the redundant information from the proof files, the complete proof is a sequence of proof steps. In the following we present some concepts related to sequences in the context of this work.

Let $PS = \{ps_1, ps_2, ..., ps_m\}$ represent the set of proof commands. A *proof steps set* PSS is a set of proof commands, that is $PSS \subseteq PS$. $|PSS|$ denotes the set cardinality. PSS has a length k (called *k-PSS*) if it contains k proof commands, i.e., $|PSS| = k$. For example, consider the set of PVS proof commands $PS = \{$*skolem, flatten, inst?, split, beta, iff, assert*$\}$. The set $\{$*skolem, flatten, assert*$\}$ is a proof steps set that contains three proof commands. For the purpose of processing commands in some order, a total order relation on proof commands is assumed to exist (e.g. the lexicographical order), denoted as $\prec$.

A proof sequence is a list of proof steps sets $S = \langle PSS_1, PSS_2, ..., PSS_n\rangle$, such that $PSS_i \subseteq PSS$ $(1 \leq i \leq n)$. For example, $\langle\{$*skolem, flatten*$\}$, $\{$*inst?*$\}$, $\{$*beta, iff*$\}$, $\{$*assert*$\}\rangle$ is a proof sequence which has four proof steps sets being used to prove a theorem. A *proof corpus* PC is a list of proof sequences $PC = \langle S_1, S_2, ..., S_p\rangle$, where each sequence has an identifier (ID). For example, Table 1 shows a PC that has five proof sequences with IDs 1, 2, 3, 4 and 5.

Table 1. A sample of a proof corpus

ID	Proof Sequence
1	$\langle\{$*inst 1 "lambda (x,y: sequence[Time]): false", grind*$\}\rangle$
2	$\langle\{$*skosimp, expand "Teq", flatten, assert*$\}\rangle$
3	$\langle\{$*skosimp, expand "Fifon", propax*$\}\rangle$
4	$\langle\{$*skeep, expand "Tle", typepred "<", expand "strict_ order?", flatten, expand "transitive?", inst -2 "T(s1)""T(s2)""T(s3)", assert* $\}\rangle$
5	$\langle\{$*induct n*$\}$,$\{$*expand "sum", propax*$\}$,$\{$*skosimp, expand "sum" +, assert*$\}\rangle$

The final step is to convert the proof sequences into sequences of integers to bring the corpus in a more suitable format for SPM techniques. In the final corpus, each line represents a proof sequence that was used for the proof of a theorem / lemma. Each proof command in the sequence is replaced by a positive integer. For example, the proof command *skosimp* is replaced by 1. Moreover, proof commands are separated with a single space followed by a negative integer -1. The negative integer -2 appears at the end of each line to indicate the end of a proof sequence. It is to note that the same integers are used for similar proof commands such as *(inst?)* and *(inst fnum constants)*, and *(skosimp)* and

(skosimp)*. This makes the learning process more general in nature and can be used for other PVS theories, in particular for the PVS library.

A proof sequence $S_\alpha = \langle \alpha_1, \alpha_2, ..., \alpha_n \rangle$ is present or contained in another proof sequence $S_\beta = \langle \beta_1, \beta_1, ..., \beta_m \rangle$ iff there exist integers $1 \leq i_1 < i_2 < ... < i_n \leq m$, such that $\alpha_1 \subseteq \beta_{i1}, \alpha_2 \subseteq \beta_{i2}, ..., \alpha_n \subseteq \beta_{im}$ (denoted as $S_\alpha \sqsubseteq S_\beta$). If S_α is present in S_β, then S_α is a *subsequence* of S_β. In SPM, various measures are used to evaluate the importance and interestingness of a subsequence. The *support* measure is used by most SPM techniques. The *support* of S_α in PC is the total number of sequences (S) that contain S_α, and is represented by $sup(S_\alpha)$:

$$sup(S_\alpha) = |\{S | S_\alpha \sqsubseteq S \wedge S \in PC\}|$$

SPM is an enumeration problem that aims to find all the *frequent subsequences* in a sequential dataset. A sequence S is a *frequent sequence* (also called *sequential pattern*) iff $sup(S) \geq minsup$, where $minsup$ (minimum support) is the threshold being determined by the user. A sequence containing n items (proof commands in this work) in a corpus can have up to $2^n - 1$ distinct subsequences. This makes the naive approach to calculate the support of all possible subsequences infeasible for most corpora. Several efficient algorithms have been developed in recent years that do not explore all the search space for all possible subsequences.

All SPM algorithms investigate the patterns search space with two operations: *s-extensions* and *i-extensions*. A sequence $S_\alpha = \langle \alpha_1, \alpha_2, ..., \alpha_n \rangle$ is a *prefix* of another sequence $S_\beta = \langle \beta_1, \beta_1, ..., \beta_m \rangle$, if $n < m$, $\alpha_1 = \beta_1$, $\alpha_2 = \beta_2$, ..., $\alpha_{n-1} = \beta_{n-1}$, where α_n is equal to the first $|\alpha_n|$ items of β_n according to the $\prec$ order. Note that SPM algorithms follow a specific order $\prec$ so that the same potential patterns are not considered twice and the choice of the order $\prec$ does not affect the final result produced by SPM algorithms. A sequence S_β is an *s-extension* of a sequence S_α for an item x if $S_\beta = \langle \alpha_1, \alpha_2, ..., \alpha_n, \{x\} \rangle$. Similarly, for an item x, a sequence S_γ is an *i-extension* of S_α if $S_\gamma = \langle \alpha_1, \alpha_2, ..., \alpha_n \cup \{x\} \rangle$. SPM algorithms either employ a breadth-first search or a depth-first search. In the following, a brief description of state-of-the-art SPM algorithms is presented.

The TKS (Top-k Sequential) algorithm finds the top-k sequential patterns in a corpus, where k is set by the user and it represents the number of sequential patterns to be discovered by the algorithm. TKS employs the basic candidate generation procedure of SPAM and vertical database representation. With vertical representation, support for patterns can be calculated without performing costly database scans. This makes vertical algorithms to perform better on dense or long sequences. TKS also uses several strategies for search space pruning and depends on the PMAP (Precedence Map) data structure to avoid costly operations of bit vector intersection. Another SPM algorithm is the CM-SPAM algorithm that performs a depth-first search to discover frequent sequential patterns in a corpus. The CMAP (Co-occurrence MAP) data structure is used in CM-SPAM to store co-occurrence of item information. A generic pruning mechanism that is based on CMAP is used for pruning the search space with vertical

database representation, to efficiently discover sequential patterns. More detail on TKS and CM-SPAM can be found in [11], [10] respectively.

Sequential patterns that appear frequently in a corpus with low confidence are worthless for decision making or prediction. Sequential rules discover patterns by considering not only their support but also their confidence. A sequential rule $X \rightarrow Y$ is a relationship between two PSSs $X, Y \subseteq PS$, such that $X \cap Y = \emptyset$ and $X, Y \neq \emptyset$. The rule $r : X \rightarrow Y$ means that if items of X occur in a sequence, items of Y will occur afterward in the same sequence. X is contained in S_α (written as $X \sqsubseteq S_\alpha$) iff $X \subseteq \bigcup_{i=1}^{n} \alpha_i$. A rule $r : X \rightarrow Y$ is contained in S_α ($r \sqsubseteq S_\alpha$) iff there exists an integer k such that $1 \leq k < n$, $X \subseteq \bigcup_{i=1}^{k} \alpha_i$ and $Y \subseteq \cup_{i=k+1}^{n} \alpha_i$. The confidence of r in PC is defined as:

$$conf_{PC}(r) = \frac{|\{S | r \sqsubseteq S \wedge S \in PC\}|}{|\{S | X \sqsubseteq S \wedge S \in PC\}|}$$

The support of r in PC is defined as:

$$sup_{PC}(r) = \frac{|\{S | r \sqsubseteq S \wedge S \in PC\}|}{|PC|}$$

A rule r is a *frequent sequential rule* iff $sup_{PC}(r) \geq minsup$ and r is a *valid sequential rule* iff it is frequent and $conf_{PC}(r) \geq minconf$, where the thresholds $minsup, minconf \in [0, 1]$ are set by the user. Mining sequential rules in a corpus deals with finding all the valid sequential rules. For this, the ERMiner (Equivalence class based sequential Rule Miner) algorithm [12] is used. It relies on a vertical database representation and represents the search space of rules using equivalence classes of rules having the same antecedent or consequent. It employs two operations (left and right merges) to explore the search space of frequent sequential rules, where the search space is pruned with the Sparse Count Matrix (SCM) technique, which makes ERMiner more efficient than other sequential rule finding algorithms.

The statistical Naive Bayes (NB) classifier [32] is based on Bayes' theorem and is used to compute the probability of using the proof p in the corpus to prove a new conjecture c. A conjecture is a proposition or statement that has not been proved yet, but is thought to be true. The probability is based on the fact that some p are already used before in the proof of conjectures similar to c. As each p contains a set of proof steps, the conditional probability $P(PSS|c)$ estimates the relevance of PSS for c. The conditional probability is computed and multiplied to get the overall probability for c.

The Compact Prediction Tree+ (CPT+) model is used to predict the next proof step(s) [18]. CPT+ implements two strategies for compression to reduce the CPT size and one strategy for the reduction of prediction time. In the training phase, CPT+ takes a set of training sequences as input and generates three data structures: a prediction tree, a lookup table and an inverted index. These three structures are built incrementally by considering the sequence one by one during training. For a proof sequence S_α of n elements, the suffix of S_α of size y where $1 \leq y \leq n$ is defined as $P_y(S_\alpha) = \langle \alpha_{n-y+1}, \alpha_{n-y+2}, ..., \alpha_n \rangle$. Predicting the next

proof steps of S_α is done by finding those sequences that are similar to $P_y(S_\alpha)$ in any order. For prediction, CPT+ uses the *consequent* of each sequence that is similar to S_α. Let S_β be another proof sequence similar to S_α. The consequent of S_β with respect to S_α is the longest subsequence $\langle \beta_v, \beta_{v+1}, ..., \beta_m \rangle$ of S_β such that $\bigcup_{k=1}^{v-1}\{\beta_k\} \subseteq P_y(S_\alpha)$ and $1 \leq v \leq m$. Each proof command discovered in the consequent of a similar proof sequence of S_α is stored in the count table (CT) data structure. CPT+ in last returns the most supported proof step(s) in the CT as prediction(s).

3 Experiments

All the following experiments are performed on an HP laptop with a fifth generation Core i5 processor and 8 GB RAM. For the case study, we select our previous work [29], where PVS is used for the analysis and verification of Reo connectors composed of untimed and timed channels. The main reason to select the proofs in [29] is that we are extending the formalization framework to cover the probabilistic [3] and stochastic [4] behavior of Reo connectors. The approach not only enabled us to comprehend the proof process for probabilistic connectors but also can be considered far effective in providing the necessary guidance to attain the proofs of such connectors.

SPMF data mining library, developed in JAVA, is used to analyze the proof corpus. It is an open-source and cross-platform framework that is specialized in pattern mining tasks. SPMF offers implementations for more than 150 data mining algorithms. More detail on SPMF can be found in [13].

3.1 Case Study

Reo [2] is a channel-based exogenous coordination language that allows the construction of complex *connectors* from primitive *channels* through compositional operators. Connectors in Reo provide the protocol for controlling and organizing the communication, synchronization and cooperation between components. Each channel in Reo has two channel ends type *source* or *sink*. The connector behavior in PVS is formalized by means of data-flows on its sink and source nodes, which are essentially infinite sequences. In PVS, record structure named *TD* is used to represent the *timed-data* sequences on sink and source nodes, where *time* is defined as a *positive real number* ($\mathbb{R}^+$) and *data* is defined as a *positive* type. Three main composition operators (flow through, replicate and merge) are used in Reo for connector construction. Flow through and replicate operators can be achieved explicitly in PVS, whereas merge operator is defined inductively.

We omit the details of Reo connector modeling in PVS due to the length limitation. Interested readers can find more details in [29, 31]. Here, one example is provided to show how connectors are modeled and how properties for connectors are proved in PVS.

Example 1. Figure 2 shows a connector which consists of one *Synchronous* (*Sync*) channel (AB) and one *FIFO1* channel (BC), that accepts data items at source

node A and stores the data items in the buffer, before dispensing them through the sink node C. The mixed node B allows the data items to move from the $Sync$ channel to $FIFO1$ channel without any change.

Fig. 2. A connector composed of a Sync and a FIFO1 channel

Let a, b, c denote the time sequences when the corresponding data sequence flows through nodes A, B and C respectively. According to the semantics of $Sync$ and $FIFO1$ channels, $a = b < c$. Let α, γ represent the data sequences being observed at nodes A and C respectively, and $\alpha = \gamma$. In PVS, these results are proved with the following theorem.

Theorem 1. *Sync(A,B)* $\wedge$ *Fifo1(B,C)* $\Rightarrow$ *Tle(A,C)* $\wedge$ *Teq(A,B)* $\wedge$ *Deq(A,C)*

Proof. PVS prover is based on *sequent calculus* and it can build a graphical proof tree for a proof goal. The nodes in the proof tree are sequents. PVS proof commands may divide the main goal into sub-goals (tree leaves). The proof is completed when all the sub-goals are proved. The proof steps for theorem 1 are shown in Figure 3.

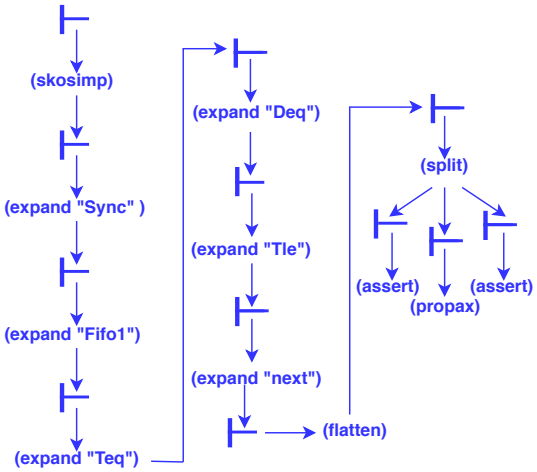

Fig. 3. PVS proof tree for theorem 1

3.2 Results and Discussion

Results obtained by applying SPM algorithms on the proof corpus are discussed in this section.

The TKS algorithm is first applied on the corpus to find hidden proof steps and patterns. TKS takes a corpus and a user specified parameter k as input and returns the top-k most frequent patterns as output. The parameter k is used in place of *minusp* threshold due to the following two reasons:

1. Selection of a proper minimum support to discover the desired amount of useful patterns has an effect on the performance of SPM algorithms.
2. The minimum support fine-tuning process is hard and time consuming.

To overcome these drawbacks, the parameter k puts a bound on the total number of patterns to be discovered by the algorithm. Some proof patterns discovered by the TKS algorithm with varying length are shown in Table 2. The column **Sup** indicates the occurrence count of each pattern in the corpus. Table 2 provides some useful information related to frequent occurrence of proof steps and patterns that are used in the verification of Reo channels and connectors.

Table 2. Extracted proof steps / patterns with TKS algorithm

Pattern (length = 1)	**Sup**	**Pattern (length = 2)**	**Sup**
expand	40	*expand, expand*	34
skosimp	39	*skosimp, expand*	31
assert	33	*skosimp, assert*	25
inst	24	*flatten, assert*	19
flatten	20	*expand, inst* and *skosimp, inst*	14
typepred	11	*typepred, expand*	10
grind and *propax*	10	*flatten, split*	9
Pattern (length = 3)	**Sup**	**Pattern (length = 4)**	**Sup**
expand, expand, expand	25	*expand, expand, expand, expand*	22
typepred, expand, assert	9	*expand, expand, flatten, assert*	14
expand, expand, flatten	15	*expand, expand, expand, assert*	20
skosimp, expand, expand	26	*skosimp, expand, expand, expand*	19
skosimp, expand, assert	20	*induct, skosimp, expand, assert*	5
skosimp, expand, inst	10	*skosimp, flatten, split, assert*	7
expand, flatten, split	9	*typepred, expand, expand, inst*	5

Pattern (length $\geq$ 5)	**Sup**
expand, expand, expand, expand, assert	17
skosimp, expand, expand, expand, expand	19
skosimp, expand, expand, expand, assert	15
skosimp, expand, expand, flatten, assert	13
skosimp, expand, flatten, split, assert	7
typepred, expand, flatten, expand, inst, assert	4
skosimp, expand, expand, expand, assert, split	6
induct, skosimp, expand, expand, assert	5
skosimp, expand, expand, expand, assert, split	5

Unlike TKS, the CM-SPAM algorithm offers the *minsup* threshold. Table 3 lists some of the most useful frequent proof patterns in the corpus which are extracted with the CM-SPAM algorithm. The first six proof patterns appear in at least 50% of the sequences in the corpus. The next six patterns appear in at least 40% of the sequences and last two patterns appear in at least 10% of the sequences. Discovered patterns with the CM-SPAM algorithm are almost similar to the results obtained with the TKS algorithm. As the outputs of TKS and CM-SPAM are very similar, the performance of TKS with CM-SPAM is compared in

terms of execution time and memory used. The CM-SPAM is fine tuned with the *minsup* threshold to generate the k proof patterns. For optimal support, TKS execution time is very similar to CM-SPAM. Similarly, TKS showed excellent scalability. These results, which are consistent with [11], are important because finding the top-k sequential proof patterns is a harder problem than mining all proof patterns, as the *minsup* requires dynamic raising that starts from 0.

Table 3. Frequent proof patterns extracted with CM-SPAM

Pattern	Sup	Min. Sup
expand	40	0.5
assert	33	0.5
skosimp	39	0.5
expand, expand	34	0.5
expand, assert	28	0.5
skosimp, expand	31	0.5
inst	24	0.4
expand, expand, expand	25	0.4
expand, expand, expand, expand	22	0.4
expand, expand, assert	25	0.4
skosimp, expand, expand	26	0.4
skosimp, assert	25	0.4
inst, assert	6	0.1
expand, typepred, inst	6	0.1

Figure 4 shows the relationships between proof steps / patterns that are discovered through sequential rule mining with the ERMiner algorithm. The confidence ($misconf$) threshold is set to 70%, which means that rules have a confidence of at least 70% (a rule $X \rightarrow Y$ has a confidence of 70% if the set of proof commands in X is followed by the set of proof commands in Y at least 70% of the times when X appears in a proof sequence). The value above the arrow is for the support and the value below the arrow indicates the confidence (probability). For example, the first rule in Figure 4 indicates that 94.7% of the time, the *assert* command is followed after the *expand* command. With the ERMiner algorithm, we found some interesting relationship and dependency of proof steps / patterns with each other. Results obtained so far indicate that the total number of proof steps in each proof (abstraction simplicity) has a direct correlation on the efficiency of SPM algorithms.

In [7], common proof patterns are found in the Isabelle proofs with a variable length Markov Chain. Proofs are represented in a tree structure format, which are linearized, such as the proofs are split into separate sequences and given weights accordingly. However, linearization means losing any important connections (information) between different branches in the proofs due to which interesting patterns may well be lost. In this work, the proof corpus contains all the necessary important information for pattern discovery and SPM algorithms, which are more user-friendly and work efficiently on the corpus.

The NB classifier implemented in SPMF is used to check the dependency of new conjectures on already proved proofs. For that, the classifier is trained on

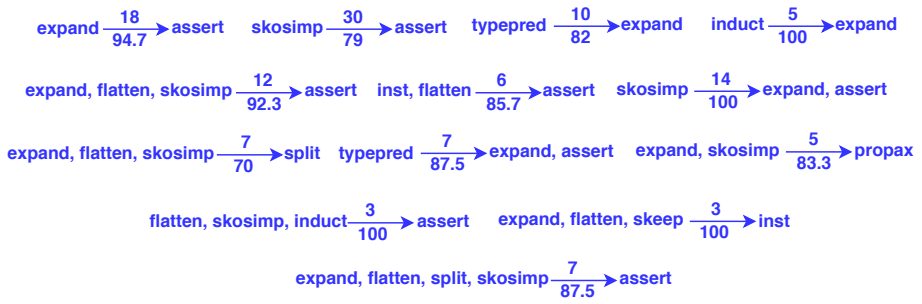

Fig. 4. Sequential rules discovered in corpus

the proofs presented in the corpus. We then provide new conjectures from our ongoing work on probabilistic Reo connectors. In the output, the classifier successfully classified the new conjectures, which shows that the proofs can be used in guiding the proof process of new conjectures. Moreover, for conjectures taken from PVS libraries, the classifier was unable to classify, which means that their proofs are not dependent on the facts present in the corpus. NB classifiers are also used in [23] for computing the proof dependencies for new conjectures from the theorems taken from the Coq repository. Obtained results are presented with measures such as precision, recall and rank. In SPMF, the NB implementation only provides the binary type output for classification and does not provide information for the measures. In future, we would like to enhance the implementation of NB to provide statistics about the measures.

Predicting the next proof steps for the new conjecture or unproved theorem / lemma has gained increased importance in last few years. The CPT+ model is used for predicting the next proof steps. The model is first trained on the proof sequences in the corpus. The prediction model is then used to predict the next proof step for a new proof sequence. Prediction of the next proof step is based on the scores calculated by the model for each proof command. For example, CPT+ predicted *assert* for the proof sequence <*flatten, split*>. The statistics and scores assigned by the model to each proof step for the previous example are listed in Table 4. It is to note here that a higher score is considered better for CPT+.

Table 4. Results for TPC+ prediction model

Statistics	Value	Proof step	Score
Number of distinct items	13	*skosimp*	2.018
Itemsets per sequence	12.342	*expand*	20.244
Distinct item per sequence	5.142	*assert*	**26.297**
Occurrence for each item	2.4	*inst*	2.778
Corpus size in MB	124.001	*grind*	2.636

To check the efficiency of CPT+, we compared its performance with various other state-of-the-art prediction models such as Dependency Graph (DG), Transition Directed Acyclic Graph (TDAG), CPT (the predecessor of CPT+),

AKOM (All-K-Order-Markov) and LZ78. Each model is trained and tested with 10-fold cross-validation. The cross-validation technique characterizes the performance of each model by evaluating the generalization of independent set over statistical results provided by the model. In k-fold cross-validation, the dataset is randomly partitioned into k sub-datasets. One sub-dataset is then selected as validation set for model testing and the remaining $k-1$ sub-datasets are used for model training. This process is continued for k times and each sub-dataset is used exactly once as the validation set. Single estimation of the result is obtained by taking the average of k results. The main reason to use 10-fold cross-validation is to achieve low variance in each run. Obtained results for various prediction models are shown in Table 5.

Table 5. Accuracy of prediction models

Models	**DG**	**TDAG**	**CPT+**	**CPT**	**AKOM**	**LZ78**
Success	41.176	73.529	79.412	**85.714**	73.529	50
Failure	**58.824**	26.471	20.588	14.286	26.471	50
No Match	0.00	0.00	0.00	**17.647**	0.00	0.00
Overall	41.176	73.529	**79.412**	70.588	73.529	50

For evaluation of prediction models, three measures are used. The result of a prediction can be:

- a *success* if the model predicts accurately,
- a *failure* if the model predicts inaccurately and
- *no match* if the model cannot perform the prediction.

The overall performance of each model is measured through its *accuracy*, which is the total number of successful predictions performed by the model against the total number of test sequences. Two other measures *training time* and *testing time* are not included in the results here as all the models take almost the same time for training and testing. CPT+ achieved higher accuracy (79.412) as compared to other prediction models. CPT has a higher *success rate* than CPT+, but the higher *no match rate* makes its accuracy lower than CPT+. Markov based prediction models DG achieved the lowest *success rate* and highest *failure rate*, while TDAG and AKOM have the same results for all four parameters.

4 Related Work

Using machine learning and data mining in theorem provers is not a new idea and they are used mainly for three tasks: premise selection, strategy selection and internal guidance. Support vector machines (SVMs) and Gaussian processes (GPs) were used in [6] for selection of a good heuristics in the E theorem prover. In [27], kernel methods were applied for strategy scheduling and strategy finding problems in three ATPs: E, Satallax and LEO-II. Deep networks have been used in [28] for internal guidance in E, where deep learning based proof guidance increases the total number of theorems proved while reducing the average number of proof search steps. Moreover, internal proof guidance methods

based on the *watchlist* technique were developed in [17] for E prover. A proof search guidance technique based on leanCoP was presented in [24] to guide the tableaux proof search. In [33], GRU networks were used in MetaMath for guiding the proof search of a tableau style proof process. Monte-Carlo tree search methods added with a connection tableau were studied and implemented in leanCoP in [9] for guiding the proof search. A new theorem proving algorithm (implemented in *rlCoP*) was recently presented in [26] for proof guidance that uses Monte-Carlo simulations with reinforcement iterations. *rlCoP* showed better performance than *leanCoP* in solving unseen problems when trained on a large corpus.

For HOL based theorem provers, variable length Markov models (VLMM) technique has been applied in [7] on a proof corpus of the Isabelle prover to identify sequences of proof steps and these sequences were used to form tactics. Particle swarm optimization and NB based techniques were proposed in [8] to internally guide the given-clause algorithm in the Satallax prover. Premise selection techniques were developed in [23] for the Coq system, where machine learning methods are compared on Coq proofs taken from the CoRN repository. Recurrent and convolutional neural networks were used in [21] for premise selection in the Mizar prover. A corpus of proofs was constructed in [1] for training a kernalized classifier with bag-of-word features that show the term occurrences in a vocabulary. Premise selection based on machine learning and automated reasoning for the HOL4 is provided in [15] by adapting HOL(y)Hammer [25]. A learning based automation technique was recently developed in [16] called *TacticToe* on top of the HOL4 for automation of theorems proofs. The *HolStep* dataset, introduced in [22], consists of 10K conjectures and 2M statements to develop new machine learning based proof strategies.

5 Conclusion

The proof development process in ITPs requires heavy interactions between users and the proof assistants, where users are forced to do lots of repetitive work which makes the proving process a more time consuming activity. To make the proof process simpler and for proof guidance, the SPM-based learning approach is adopted in this work to find the frequent proof steps / patterns and their relationship in a PVS theory. NB classifier is used to check the dependency of new conjectures on the already proved proofs. Moreover, the performance of some models for the prediction of next proof step(s) is compared, where CPT+ performs better than other models. Some interesting proof patterns are found with SPM and obtained results show that the number of proof steps in each proof has a direct correlation on the efficiency of SPM algorithms.

There are several directions of future work. First, we would like to use the SPM algorithms on the corpora of proof steps for theories included in PVS library, which contains thousands of theorems. This will enable us to develop a more general learning approach for the proofs of new conjunctures. Another direction is to use evolutionary and heuristics techniques such as genetic programming and particle swarm optimization for the development of PVS strategies

from frequently occurring proof patterns. Some other future work includes the implementation of some famous classifiers such as k-nearest neighbor in SPMF and enhancing the implementation of NB to provide statistics about common measures such as precision, recall and f-measure. Last but not the least, using SPM algorithms on the dataset provided by [22] is in our future plan as well.

Acknowledgement. The work was partially supported by the National Natural Science Foundation of China under grant no. 61772038, 61532019, 61202069 and 61272160.

References

1. J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning*, 52(2):191–213, 2014.
2. F. Arbab. Reo: A channel-based coordination model for component composition. *Mathematical Structures in Computer Science*, 14(3):329–366, 2004.
3. C. Baier. Probabilistic models for Reo connector circuits. *Journal of Universal Computer Science*, 11(10):1718–1748, 2005.
4. C. Baier and V. Wolf. Stochastic reasoning about channel-based component connectors. In *Proceedings of COORDINATION 2006*, volume 4038 of *LNCS*, pages 1–15. Springer, 2006.
5. J. C. Blanchette, M. P. L. Haslbeck, D. Matichuk, and T. Nipkow. Mining the archive of formal proofs. In *Proceedings of CICM 2015*, volume 9150 of *LNCS*, pages 3–17. Springer, 2015.
6. J. P. Bridge, S. B. Holden, and L. C. Paulson. Machine learning for first-order theorem proving - Learning to select a good heuristic. *Journal of Automated Reasoning*, 53(2):141–172, 2014.
7. H. Duncan. *The use of data-mining for the automatic formation of tactics*. PhD thesis, University of Edinburgh, UK, 2007.
8. M. Färber and C. E. Brown. Internal guidance for Satallax. In *Proceedings of IJCAR 2016*, volume 9706 of *LNCS*, pages 349–361. Springer, 2016.
9. M. Färber, C. Kaliszyk, and J. Urban. Monte carlo tableau proof search. In *Proceedings of CADE 2016*, volume 10395 of *LNCS*, pages 563–579. Springer, 2017.
10. P. Fournier-Viger, A. Gomariz, M. Campos, and R. Thomas. Fast vertical mining of sequential patterns using co-occurrence information. In *Proceedings of PAKDD 2014*, volume 8443 of *LNCS*, pages 40–52. Springer, 2014.
11. P. Fournier-Viger, A. Gomariz, T. Gueniche, E. Mwamikazi, and R. Thomas. TKS: Efficient mining of top-k sequential patterns. In *Proceedings of ADMA 2013*, volume 8346 of *LNCS*, pages 109–120. Springer, 2013.
12. P. Fournier-Viger, T. Gueniche, S. Zida, and V. S. Tseng. ERMiner: Sequential rule mining using equivalence classes. In *Proceedings of IDA 2014*, volume 8819 of *LNCS*, pages 108–119. Springer, 2014.
13. P. Fournier-Viger, J. C. Lin, A. Gomariz, T. Gueniche, A. Soltani, Z. Deng, and H. T. Lam. The SPMF open-source data mining library version 2. In *Proceedings of the ECML/PKDD 2016*, volume 9853 of *LNCS*, pages 36–40. Springer, 2016.
14. P. Fournier-Viger, J. C. W. Lin, R. U. Kiran, Y. S. Koh, and R. Thomas. A survey of sequential pattern mining. *Data Science and Pattern Recognition*, 1(1):54–77, 2017.

15. T. Gauthier and C. Kaliszyk. Premise selection and external provers for HOL4. In *Proceedings of CPP 2015*, pages 48–57. ACM, 2015.
16. T. Gauthier, C. Kaliszyk, and J. Urban. TacticToe: Learning to reason with HOL4 tactics. In *Proceedings of LPAR 2017*, volume 46 of *EPiC Series in Computing*, pages 125–143, 2017.
17. Z. Goertzel, J. Jakubuv, S. Schulz, and J. Urban. Proofwatch: Watchlist guidance for large theories in E. In *Proceedings of ITP 2018*, volume 10895 of *LNCS*, pages 270–288. Springer, 2018.
18. T. Gueniche, P. Fournier-Viger, R. Raman, and V. S. Tseng. CPT+: Decreasing the time/space complexity of the compact prediction tree. In *Proceedings of PAKDD 2015*, volume 9078, pages 625–636. Springer, 2015.
19. J. Harrison, J. Urban, and F. Wiedijk. History of interactive theorem proving. In *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 135–214. Elsevier, 2014.
20. O. Hasan and S. Tahar. Formal verification methods. In *Encyclopedia of Information Science and Technology, Third Edition*, pages 7162–7170. IGI Global, 2015.
21. G. Irving, C. Szegedy, A. A. Alemi, N. Eén, F. Chollet, and J. Urban. Deepmath - Deep sequence models for premise selection. In *Proceedings of NIPS 2016*, pages 2243–2251. ACM, 2016.
22. C. Kaliszyk, F. Chollet, and C. Szegedy. HolStep: A machine learning dataset for higher-order logic theorem proving. *CoRR*, abs/1703.00426, 2017.
23. C. Kaliszyk, L. Mamane, and J. Urban. Machine learning of Coq proof guidance: First experiments. In *Proceedings of SCSS 2014*, volume 30 of *EPiC Series in Computing*, pages 27–34, 2014.
24. C. Kaliszyk and J. Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In *Proceedings of LPAR 2015*, volume 9450 of *LNCS*, pages 88–96. Springer, 2015.
25. C. Kaliszyk and J. Urban. Hol(y)Hammer: Online ATP service for HOL light. *Mathematics in Computer Science*, 9(1):5–22, 2015.
26. C. Kaliszyk, J. Urban, H. Michalewski, and M. Olsák. Reinforcement learning of theorem proving. In *Proceedings of NeurIPS 2018*, pages 8836–8847, 2018.
27. D. Kühlwein and J. Urban. MaLeS: A framework for automatic tuning of automated theorem provers. *Journal of Automated Reasoning*, 55(2):91–116, 2015.
28. S. M. Loos, G. Irving, C. Szegedy, and C. Kaliszyk. Deep network guided proof search. In *Proceedings of LPAR 2017*, volume 46 of *EPiC Series in Computing*, pages 85–105, 2017.
29. M. S. Nawaz and M. Sun. Reo2PVS: Formal specification and verification of component connectors. In *Proceedings of SEKE 2018*, pages 391–396. KSI Research Inc., 2018.
30. S. Owre, N. Shankar, J. M. Rushby, and D. W. J. Stringer-Calvert. PVS system Guide, PVS prover Guide, PVS language reference. Technical report, SRI International, November 2001.
31. PVS and SPM data. Available at: https:/github.com/saqibdola/SPM-in-PVS.
32. S. J. Russell and P. Norvig. *Artificial Intelligence - A Modern Approach, Third Edition*. Pearson Education, 2010.
33. D. Whalen. Holophrasm: A neural automated theorem prover for higher-order logic. *CoRR*, abs/1608.02644, 2016.