Prover-independent Axiom Selection for Automated Theorem Proving in Ontohub

Eugen Kuksa
University of Bremen
Bremen, Germany
eugenk@informatik.uni-bremen.de

Till Mossakowski
University of Magdeburg
Magdeburg, Germany
till@iws.cs.uni-magdeburg.de

Abstract

In recent years, formalised large theories in form of ontologies, system specifications or mathematical libraries with thousands of sentences have become more and more common. Automatically proving properties in such theories is an important task in research as well as in industry, for instance, as part of system specifications. However, automated theorem provers struggle finding proofs when there are too many axioms available. This paper presents and investigates two approaches to reduce the load on automated theorem provers by preselecting axioms: 1) a generalisation of the SInE selection heuristics to arbitrary first-order provers, and 2) frequent item set mining. These selection methods are implemented in the web application Ontohub. One of the implementations' usefulness is validated while others' is refuted by an evaluation on well-known benchmarks for automated theorem proving.

1 Introduction

In recent years, the ability of systems to reason over large logical theories has become more and more important. Large theories, i.e. theories with many sentences and symbols, are becoming more common since large knowledge bases are arising in forms of ontologies and comprehensive mathematical libraries that have been translated to formal languages that are suitable for automated reasoning.

At the same time, theorem provers have evolved to feature efficient heuristics for finding proofs. These theorem provers, however, were traditionally designed for working with small to medium-sized collections of sentences, for example, an axiomatisation of set theory or of algebraic constructs like groups or vector spaces. Supplying these provers with larger theories reduces their chance of success because the search space grows exponentially with the number of sentences.

In most cases, only a few axioms are actually needed for proving a theorem in a theory containing thousands of axioms. Instead of only focusing on the improvement of the proving algorithm and the prover's heuristics, one can also reduce proving time by supplying the prover with a modified theory omitting sentences that are not needed. This significantly reduces the search space: instead of thousands of axioms, there might only be a few dozens or hundred axioms available for proving.

This work presents an implementation of two axiom selection algorithms that is not bound to a specific prover but rather works independently of the prover.

This paper is structured as follows: Section 2 provides basic knowledge of the concepts and systems used in this work, namely the software tools Ontohub and Hets (suited for proving with first-order theories) and

In: P. Fontaine, S. Schulz, J. Urban (eds.): Proceedings of the 5th Workshop on Practical Aspects of Automated Reasoning (PAAR 2016), Coimbra, Portugal, 02-07-2016, published at http://ceur-ws.org

the AI method of frequent item set mining (which we will employ for axiom selection). Section 3 presents the axiom selection algorithms whose implementation is described briefly in section 4 and evaluated in section 5. Afterwards, section 6 gives an overview of related work while section 7 concludes this paper.

2 Preliminaries

2.1 Ontohub

Ontohub [12] is an open source repository engine for managing distributed logical theories. While Ontohub focuses on ontologies, it supports arbitrary logical theories, let them be ontologies, models, specifications, or others. The distributed nature enables communities to share and exchange their contributions easily. Ontohub supports (among others) first-order logic based on the TPTP format [26] It also supports various structuring and modularity constructs and inter-theory (concept) mappings, building on the OMG-standardized DOL language.

Ontohub has several distinctive features. The following list gives a short overview of them.

- Different repositories organise logical theories. For each repository, access rights and ownership can be managed on a per-user or per-team basis.
- Logical theories (and all other files) are versioned. The underlying git[1] [27] version control system is an integral part of Ontohub. This feature allows the user to browse through the history of the repository and inspect any change in any file.
- Ontohub features an integrated editor for logic theories, which however may also be edited outside of Ontohub: files can either be uploaded via the web front end or via git with SSH.
- Ontohub is fully linked-data compliant. One and the same URL can be used for referencing a logical theory, downloading its defining file or examining it in a user-friendly presentation in the browser. The representation served by Ontohub depends on the requested MIME-type.
- Logical theories can be modular and distributed, which means they can be structured into many simpler theories depending on each other.
- As in BioPortal [17] and the NeOn Toolkit [8], logical theories can be aligned and combined along alignments.
- Logical relations like interpretations of theories, conservative extensions, etc. between logical theories are supported.
- Diverse structuring languages are supported, including DOL[2] [15] and CASL [4].
- As a novel contribution of this work, logical theories may contain conjectures which Ontohub recognises. There is an interface for proving these with different parametrisations.

2.2 Hets

Ontohub itself does only little analysis of logical theories. Instead, we use the Heterogeneous Tool Set (Hets)[3] [14] as a back end to parse and evaluate a logical theory. Hets features a RESTful web API: As soon as a logical theory is sent to Hets's HTTP server with the command to parse it, it analyses the theory file and answers the request with an XML or JSON representation of its details. Ontohub then parses the answer, post-processes it and then stores the theory's details in the database. The whole evaluation process is done asynchronously in the background, so that the user can still work with the web application.

Currently, Hets has an interface to several first-order automated provers (ATPs) as, for example, SPASS [28], E [21], Darwin [5], and via the MathServe Broker [30]: Vampire [19]. Similar to the analysis of logical theories, Hets can be called to run a proof attempt via the RESTful web API.

Hets also features user interfaces other than the web API, for example, a graphical user interface, a stateless command line interface or an interactive command prompt, but we do not use them in the context of Ontohub.

[1] Git is a popular decentralised version control system, usually utilised in software development. Many open source software projects use git in combination with Github: `https://github.com` (last checked: 13 June 2016).

[2] Details on DOL and its background can be found on `http://dol-omg.org` (last checked: 13 June 2016).

[3] Details on HETS can be found on `http://hets.eu` (last checked: 13 June 2016).

Before starting this work, Ontohub was not able to recognise conjectures in a logical theory. Hets, however, already provided proving functionality and thus, its analysis of the theory already discovered conjectures. However, Hets's proving interface has not been suitable for the use with Ontohub yet. For the preparation of the axiom selection, adjustments of Hets's interface and the creation of an infrastructure for proving conjectures in Ontohub were required.

By this work, Ontohub finds conjectures (called *theorems* for simplicity) and allows the user to run proof attempts with each of Hets's ATP that are suitable for the logic translation's target logic. For each finished proof attempt, Ontohub provides an overview of its details like the time it took, the axioms that were used and the prover's output. It is even possible to run multiple proof attempts in parallel with different ATP which yields a higher success rate [6].

2.3 Frequent Item Set Mining

Frequent item set mining [1] is a method commonly used for market basket analysis. It aims at revealing regularities in the shopping behaviour of customers of, for example, supermarkets and online shops: It finds products that are usually bought together. From a more technical perspective, it can also be used for dependence analysis. In the context of axiom selection, our approach is to find sentences that have symbols in common and hence are related to each other. This section briefly presents the basic notions of frequent item set mining. Details of implementations are not relevant at this point. The following definitions are based on Borgelt's lecture slides [7] on frequent item set mining.

Definition 2.1 *Let $B = \{i_1, \ldots, i_m\}$ be a set of items. B is called the item base. A subset $I \subseteq B$ of the item base is called an item set. A multiset[4] $T = \{t_1, \ldots, t_n\}$ with $\forall m \in \{1, \ldots, k\} : t_m \subseteq B$ is called a transaction database over B. An element t_m of T is called a transaction over B.*

Items are these entities whose occurrence together with other entities is analysed, for example, shopping products. A transaction database can be, for example, a collection of purchases in a shop over a given period of time. Note that the same transaction can occur multiple times in a transaction database because it is defined as a multiset. In practice, the item base is often given implicitly as the union of all the item sets in the transaction database.

Definition 2.2 *Let $T = \{t_1, \ldots, t_n\}$ be a transaction database, $I \subseteq B$ an item set of an item base. A transaction $t \in T$ covers I iff $I \subseteq t$. We define the multiset $K_T(I) \coloneqq \{t \in T \mid I \subseteq t\}$ with the characteristic*

$$\forall t \in K_T(I) : \text{multiplicity}_{K_T(I)}(t) = \text{multiplicity}_T(t)$$

meaning that each element of $K_T(I)$ occurs as many times in $K_T(I)$ as it occurs in T. $K_T(I)$ is called the cover of I with respect to T.

The cover is the multiset of transactions that cover the given item set. It is a tool for defining two of the central notions of this topic: the absolute and the relative support.

Definition 2.3 *$s_T(I) \coloneqq \mid K_T(I) \mid$ is the absolute support of I with respect to T. $\sigma_T(I) \coloneqq \frac{1}{n} \cdot \mid K_T(I) \mid$ is the relative support of I with respect to T.*

The support is the number or fraction of transactions in the transaction database that cover an item set. It is also called the absolute or relative frequency.

Definition 2.4 *Given*

$$\begin{aligned} \text{the size of the transaction base } & n \in \mathbb{N} \\ \text{an item base } & B = \{i_1, \ldots, i_m\} \\ \text{a transaction database } & T = \{t_1, \ldots, t_n\} \\ \text{and the minimum support of either } & s_{min} \in \{0, \ldots, n\} \subset \mathbb{N} \\ \text{or } & \sigma_{min} \in [0, 1] \subset \mathbb{R} \end{aligned}$$

[4] A multiset (also called a bag) M is an unordered collection of elements. It can contain the same elements multiple times. How many times an element is included is given by the function $\text{multiplicity}_M : M \to \mathbb{N}$.

frequent item set mining is about finding the set of frequent item sets:

$$\text{either } F_T(s_{min}) = \{I \subseteq B \mid s_T(I) \geq s_{min}\}$$
$$\text{or analogously } \Phi_T(\sigma_{min}) = \{I \subseteq B \mid \sigma_T(I) \geq \sigma_{min}\}$$

There are algorithms which compute the set of frequent item sets efficiently. In the worst case, the time complexity is exponential in the number of items. However, algorithms used in practice, like FP-Growth [9], Eclat [29] and Apriori [2], prune the search space in a way that mining is not only feasible but also finishes quickly with real world data. Except for some special cases, FP-Growth is the fastest algorithm in general [7].

3 Axiom Selection Algorithms

The basic idea of many selection methods is to find sentences that share a symbol with the already selected sentences. Standing out, SInE concentrates on the characteristics of definitions: They describe more special terms by using more general terms. It assumes that the more general terms occur more often in a theory.

An entirely different observation we made when manually solving mathematics-textbook exercises is that sentences are probably relevant if they share *many* symbols with the conjecture or the already selected sentences. This section covers axiom selection ideas based on this observation which are novel contributions of this work.

In the sequel, let symbols(φ) denote the set of non-logical symbols of a first-order sentence φ.

3.1 SInE

SInE operates heuristically on the syntax of first-order sentences and was first implemented in the Vampire Prover. It was first developed in Hoder's masters thesis 'Automated Reasoning in Large Knowledge Bases'. It entered the public as an implementation in Python with the Vampire V9 and E 0.999 as a back end at CASC-J4[5] [23]. Later, an implementation was included in both, the Vampire Prover and the E Theorem Prover. A detailed description of the algorithm can be found in [10]. This section only reiterates the original work while being formulated slightly differently since the definitions are also needed for the axiom selection methods with frequent item set mining.

Definition 3.1 *Let $k \in \mathbb{N}$, s be a symbol, φ, ψ be sentences. Furthermore, let ψ be the conjecture. Let* trigger $\subseteq \text{Sym} \times \text{Sen}$ *be an arbitrary relation.*

1. *s is 0-step triggered iff $s \in$ symbols(ψ).*
2. *φ is $k+1$-step triggered iff s is k-step triggered and* trigger(s, φ).
3. *s is k-step triggered iff there is a k-step triggered φ and $s \in$ symbols(φ).*

This recursive definition can be illustrated as follows:

$$\psi \xrightarrow{\ni} s_0 \xrightarrow{\text{trigger}} \varphi_1 \xrightarrow{\ni} s_1 \xrightarrow{\text{trigger}} \varphi_2 \xrightarrow{\ni} \dots$$

We now define a trigger relation as needed for Def. 3.1.

Definition 3.2 *The commonness of a symbol s, denoted by* occ(s)*, is the number of axioms which contain s. Note that (quantified) variables are not treated as symbols.*

Definition 3.3 *Let s be a symbol, φ be an axiom, $1 \leq t \in \mathbb{R}$ the tolerance parameter, $1 \leq g \in \mathbb{N}$ the generality threshold parameter. Let furthermore $s_{lcs} \in$ symbols(φ) with for all $s \in$ symbols(φ) : occ(s_{lcs}) $\leq$ occ(s) be the least common symbol of φ. The* trigger *relation of SInE is defined as:* trigger(s, φ) *iff $s \in$ symbols(φ) and (either* occ(s) $\leq g$ *or* occ(s) $\leq t \cdot$ occ(s_{lcs})*).*

This means that either a sentence is triggered because it contains very uncommon symbols (due to the generality threshold; occ(s) $\leq g$) or it is triggered by its least common symbol (or as close to it as the tolerance allows; occ(s) $\leq t \cdot$ occ(s_{lcs})).

SInE then selects all axioms that are at most d-step triggered given a $d \in \mathbb{N} \cup \{\infty\}$ as the depth limit parameter.

[5] The used SInE version and more information about it is available at `http://www.cs.man.ac.uk/~hoderk/sine/`. Last checked: 13 June 2016.

3.2 Simple Axiom Selection using Frequent Item Set Mining

Finding sentences that share many symbols with the conjecture is similar to finding frequent item sets: Frequent item set mining answers the question which items occur often in combination.

Definition 3.4 *Let $s_{min} \in \mathbb{N}$ and $\sigma_{min} \in [0,1] \subseteq \mathbb{R}$ be a minimum absolute respectively relative support. Let $\mathcal{T} = (A, C)$ be a theory consisting of a set of axioms A and a set of conjectures C. Let the multiset*

$$T = \{\text{symbols}(\varphi) \mid \varphi \in A \cup C\}$$

be a transaction database and let the set

$$B = \bigcup_{\varphi \in A \cup C} \text{symbols}(\varphi)$$

be the item set for frequent item set mining. Then, the frequent symbol sets are the elements of

$$\begin{aligned} \text{either} \quad & F_T(s_{min}) = \{I \subseteq B \mid s_T(I) \geq s_{min}\} \\ \text{or equivalently} \quad & \Phi_T(\sigma_{min}) = \{I \subseteq B \mid \sigma_T(I) \geq \sigma_{min}\} \end{aligned}$$

Frequent symbol set mining helps finding the symbols that occur together in a sentence and that occur in the conjecture or the already selected sentences as well. One can then select the sentences that have all the symbols of the frequent symbol sets. This would, however, ignore the short sentences which have less symbols than the absolute minimum support. To consider them as well, one can select all the short sentences that share a symbol with the already selected sentences.

Definition 3.5 *Let F be a set of frequent symbol sets for the corresponding theory (A, C) with a defined minimum support. Let s be a symbol and $\varphi \in A$ be an axiom while $\psi \in C$ is the conjecture. Let $2 \leq n \in \mathbb{N}$ be a minimal symbol set size, $t \in \mathbb{N}$ be a short axiom tolerance, $d \in \mathbb{N} \cup \{\infty\}$ be a depth limit.*

1. *φ is 0-step selected iff $\mid \text{symbols}(\varphi) \mid < n + t$.*
2. *s is 0-step selected iff $s \in \text{symbols}(\psi)$.*
3. *$S \in F$ is selection-enabled iff $n \leq \mid S \mid$.*
4. *φ is $k+1$-step selected iff*
   there is some $S \in F$ with: S is selection-enabled and $S \subseteq \text{symbols}(\varphi)$ and $s \in S$ for a k-step selected symbol s.
5. *s is k-step selected iff φ is k-step selected and $s \in \text{symbols}(\varphi)$, for $k \geq 1$.*

Then, the union of j-step selected axioms for all $j \leq d$ form the final axiom selection.

This definition describes a simple axiom selection idea that was implemented in Ontohub in this work. The recursive definition can be illustrated as follows:

$$\begin{array}{ccccccc} & & F & & F & & \\ & & \cup & & \cup & & \\ \psi & \overset{\ni}{\text{———}} & s_0 \in S_0 \subseteq \text{symbols}(\varphi_1) & \overset{\ni}{\text{———}} & s_1 \in S_1 \subseteq \text{symbols}(\varphi_2) & \overset{\ni}{\text{———}} & \dots \end{array}$$

3.3 SInE with Frequent Symbol Set Mining

Another idea to circumvent the problem of short sentences is to combine frequent item set mining with SInE. The k-step-trigger relation from Def. 3.1 is modified. Apart from this definition, the SInE algorithm is used as is.

Definition 3.6 *Let $k \in \mathbb{N}$, s, s' be symbols, φ, ψ be sentences. Furthermore, let ψ be the conjecture,* $\text{trigger} \subseteq \text{Sym} \times \text{Sen}$ *be a relation and F be the set of frequent symbol sets for the corresponding theory with a defined minimum support. Let $1 \leq \sigma \in \mathbb{R}$ be a tolerance value.*

1. *s is 0-step triggered iff $s \in \text{symbols}(\psi)$.*
2. *s is k-step-trigger-enabled iff there is an $S \in F : \{s_k, s\} \subseteq S$*
   with s_k being k-step triggered
   and for all $s' \in S : \text{occ}(s') \cdot \sigma \geq \text{occ}(s_{mcs})$
   where $s_{mcs} \in S$ is the most common symbol of S.
3. *φ is $k+1$-step triggered iff s is k-step-trigger-enabled and* $\text{trigger}(s, \varphi)$.
4. *s is k-step triggered iff φ is k-step triggered and $s \in \text{symbols}(\varphi)$.*

This recursive definition can be illustrated as follows (note that the condition involving s_{mcs} is not illustrated):

$$\begin{array}{ccccccccccc} & & F & & & & F & & & & \\ & & \cup & & & & \cup & & & & \\ \psi & \overset{\ni}{\text{———}} & s'_0 \in S_0 & \ni & s_0 & \overset{\text{trigger}}{\text{———}} & \varphi_1 & \overset{\ni}{\text{———}} & s'_1 \in S_1 & \ni & s_1 \overset{\text{trigger}}{\text{———}} \varphi_2 \overset{\ni}{\text{———}} \ldots \end{array}$$

This definition says that not the symbols that *occur* in a triggered sentence are allowed to trigger further axioms. Instead, those symbols are allowed that occur *in a frequent symbol set* in combination with a symbol from a triggered sentence. Additionally, only the most common symbols (with a tolerance) of such a frequent symbol set may trigger an axiom. Specifying a high tolerance value σ allows any symbol from such a set to trigger. The evaluation of SInE showed that there were still many axioms selected and the idea was to reduce the selection even further.

4 Implementation

The described axiom selection methods were implemented in Ontohub which is based on the web framework Ruby on Rails. One of Ontohub's basic features is the analysis of logical theories which recognises sentences and symbols inside the sentences. These are stored in a relational database and queried when an axiom selection heuristic is invoked.

In our implementation, association objects are stored in the database such that SInE's commonness values and the frequent symbol sets don't need to be re-computed. Furthermore, the minimal tolerance parameter of SInE for each axiom/symbol combination for the symbol to trigger the axiom is saved. This way, the axiom selection of consecutive proof attempts with different parametrisations of our selection methods or for different conjectures in the same logical theory is very efficient.

The axiom selections that are using frequent symbol set mining are based on the mining algorithm FP-Growth. The FP-Growth application itself is implemented in C by Borgelt[6] [7] and invoked from Ontohub's Ruby code.

5 Evaluation

5.1 Setup

The axiom selection was evaluated on first-order logic with the MPTP2078[7] [3] benchmark which is used in the TPTP format and is based on the Mizar Mathematical Library (MML) [20].

Each problem of the MPTP2078 benchmark comes in a 'bushy' and a 'chainy' version. The 'bushy' version contains only the axioms that are used in a human-made proof while the 'chainy' version contains all axioms that were introduced in the MML before the conjecture. Proof attempts were started for each problem in both versions without any axiom selection. Their results are used as a baseline: The problems that could not be solved in the bushy category are considered too hard for the used ATP with the given resources because the bushy ones only contain axioms which contribute to a known human-made proof. The other ones are the ones whose corresponding chainy version might be solved with an appropriate axiom selection. For evaluating the axiom selection methods, only the chainy versions were used. The problems that could be solved in the chainy category without an axiom selection are supposed to be also provable with an axiom selection. An axiom selection is

[6] The FP-Growth implementations sources or binaries for various systems can be downloaded from `http://borgelt.net/fpgrowth.html` (last checked: 13 June 2016).

[7] Information about the MPTP2078 benchmark is available on `http://wiki.mizar.org/twiki/bin/view/Mizar/MpTP2078` (last checked: 13 June 2016).

parameter	SInE	SInE with Frequent Symbol Set Mining	Simple Axiom Selection using Frequent Item Set Mining
generality threshold g	0	0	-
depth limit d	$5, \infty$	∞	∞
tolerance value t	$1.0, 1.2, 1.5, 2.0$	$1.0, 2.0$	-
symbol set tolerance σ	-	$1.0, 2.0$	-
min. absolute support s_{min}	-	$1, 2$	$1, 2$
short axiom tolerance t	-	-	$0, 2$
minimal symbol set size n	-	-	2

Table 1: Parameters used in the evaluation for each axiom selection method

considered not effective if fewer conjectures can be proven using it than without using it in the chainy category. It is possible that a selection hides axioms that are relevant for a proof and, hence, leads to no conclusion. However, a well performing selection should enable proving *more* conjectures by selecting all the relevant axioms and hiding only irrelevant ones.

The experiments were conducted on an instance of Ontohub specifically set up for this work. This instance runs on a machine with two Intel® Xeon® E5-2667 v2 processors with HyperThreading (resulting in 32 logical processor cores) typically clocked at 3.3 GHz along with 256 GB of RAM.

For analysis and proving, 30 Hets instances were used in parallel. This puts heavy load on 30 of the cores. The asynchronous (to the web interface) processing of Hets responses is done by another 30 Ruby processes. Their workload alternates with that of the Hets processes, such that no additional cores are used extensively at the same time. The web user interface's workload is very low. During the evaluation, it is only called by Hets to fetch the theory files and it does not put heavy use to another core. In total 30 cores were used completely and one additional core was used with little load. This leaves one spare processor core for other tasks. During the experiments, processor utilisation of around 70% was observed.

For the benchmark, SPASS v3.7 and the E Theorem Prover 1.8 were used. These are among the best-rated first-order provers. Since E features an own implementation of SInE, this was not activated by only using the parameters `-xAuto -tAuto` among (soft) CPU time and memory limiting as well as input and output parameters.

The parameters for the axiom selections are shown in Table 1. Proof attempts were made with each combination of these parameters. Each of the provers was given a timeout of 10 seconds.

During the run with the first SInE with Frequent Symbol Set Mining configuration, a very slow runtime behaviour was observed for some examples. The selection for these examples was cancelled after about eight hours of selection runtime. As a consequence, a timeout for the axiom selection was implemented: If the axiom selection itself took longer than two minutes, the proof attempt was cancelled. The choice of two minutes is arbitrary. The selection timeout must be chosen considerably large to make sure that much CPU time is consumed because it was measured using wall clock time. The controlled environment ensured that no other CPU-hungry processes were running other than the axiom selection and proving (which are limited to less than one process per core). Hence, it is unlikely that other processes used much CPU time that should have been given to the selection for a perfect timing evaluation (as it would have happened by measuring CPU time instead of wall clock time). The selection timeout has to be much longer than the chosen proving timeout, though, because we want to analyse the selection run time as well. An axiom selection that consequently takes this much longer than proving a conjecture is not an acceptable method because the goal is not only to enable proofs but also shorten the overall proving time which includes the axiom selection.

5.2 Results

In the MPTP2078 benchmark, only 2044 problems are non-trivial. The other 34 conjectures are equal to an axiom and could be proved by Hets during static analysis using reflexivity of an entailment relation without invoking an external prover. Only the 2044 non-trivial problems were used for the axiom selection method evaluation.

The first figures in this section show stacked column charts of the performance: Figure 1 shows how many problems could be solved using SInE. The same goes for Figure 2 using SInE with Frequent Symbol Set Mining. Finally, Figure 3 shows the performance of the Simple Axiom Selection using Frequent Item Set Mining.

Top-rated first-order provers return a status of the proof attempt telling the result in a single word. These statuses are defined in the SZS ontology [22]. The most common statuses used by provers are

(i) THM (Theorem): All models of the axioms are models of the conjecture.

(ii) CSA (CounterSatisfiable): Some models of the axioms are models of the conjecture's negation.

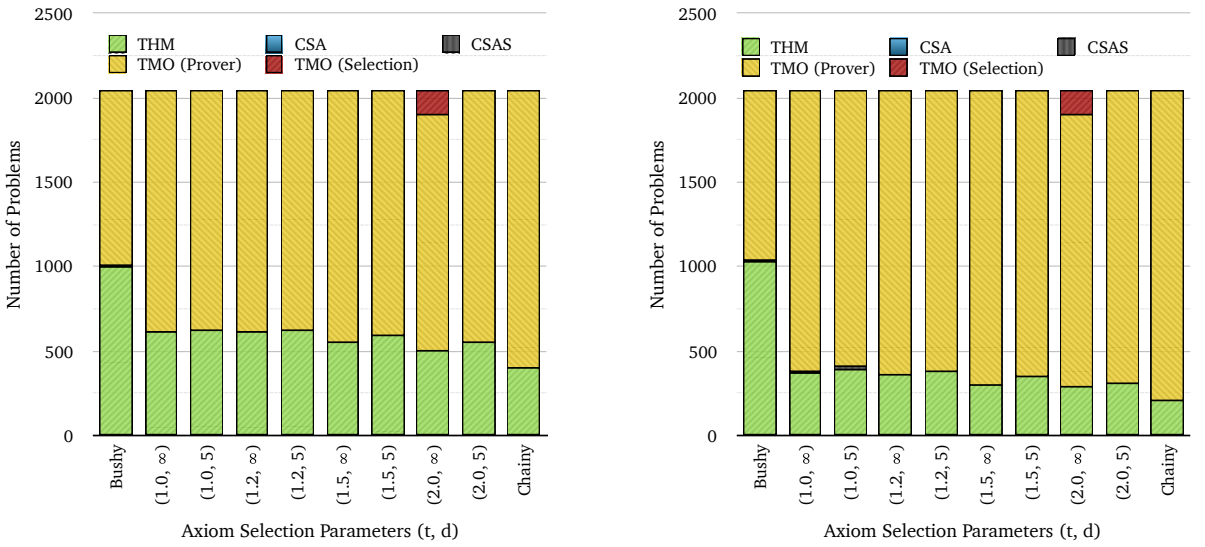
(iii) TMO (Timeout): The prover exceeded the time limit.

Of these statuses, 'THM' and 'CSA' indicate a successful prover run, while 'TMO' shows that the prover did not finish successfully. There are more such values indicating failures which are actually used by software, but ATP only use the two values 'THM' and 'CSA' in case of success. We extended this ontology[8] by a status specifically for proving with reduced axiom sets:

(iv) CSAS (CounterSatisfiableWithSubset): Some models of the *selected* axioms are models of the conjecture's negation.

If a refutation of the conjecture is found using a strict subset of the axioms (which means that the prover returns with 'CSA'), we do not know whether the conjecture is really false or we have excluded an axiom that is crucial to a potentially existing proof. If the prover returns with 'THM', we know by monotonicity of entailment relations that the found proof is also a proof with the full axiom set.

When a prover returns with a proof (marked in the diagrams as 'THM') or a refutation (marked as 'CSA'), a problem is considered solved. When it returns with a refutation while only a true subset of the axioms was used (marked as 'CSAS'), or with a proving timeout (marked as 'TMO (Prover)') or when the axiom selection takes more than two minutes (marked as 'TMO (Selection)'), the problem is considered still unsolved.



(a) E Theorem Prover

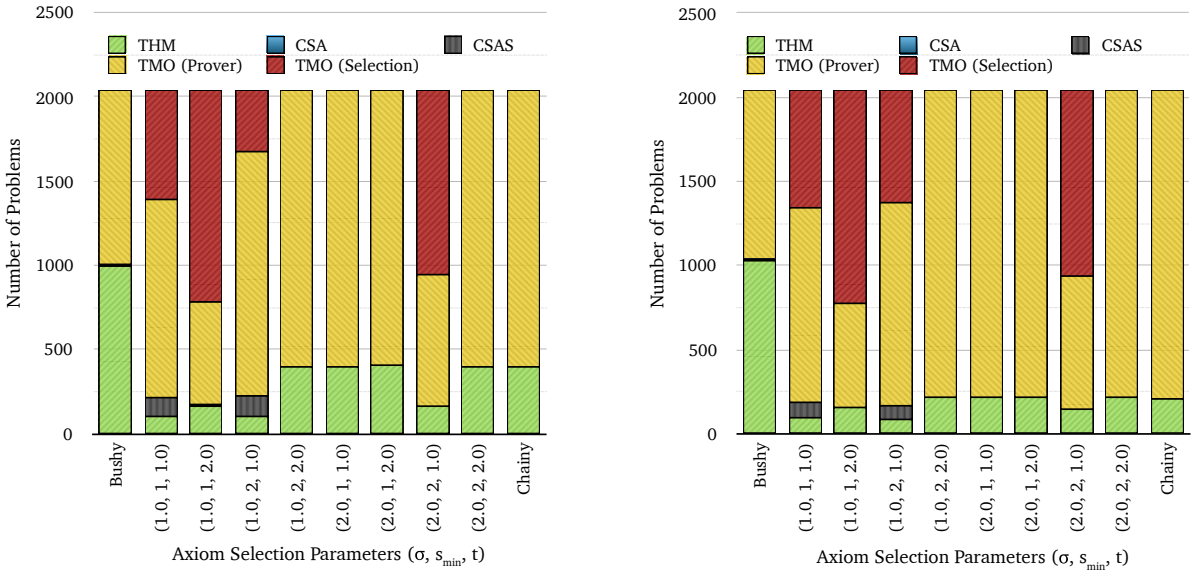
(b) SPASS

Figure 1: Performance of the provers in MPTP2078 using SInE

Figure 4 shows the average runtime in milliseconds of each axiom selection method over all parametrisations. It contains two values for each method. The first one is the average over all selections that did not reach the two-minute timeout. The second one is the average runtime of the selection, regardless whether or not the timeout was reached. This, of course, counts every selection that was cancelled as two minutes.

Figure 5 shows how many axioms were selected on average using SInE. Both versions of the MPTP2078 without a selection are displayed for reference. Due to a high amount of timeouts in the other two selection methods, they are not included in these statistics. The distortion caused by leaving out selections that encountered a timeout would be too high.

[8]Our modified SZS ontology can be found on `http://ontohub.org/meta/proof_statuses`.



(a) E Theorem Prover

(b) SPASS

Figure 2: Performance of the provers in MPTP2078 using SInE with Frequent Symbol Set Mining

Benchmark Size.

To interpret the results properly, we need to know the size of the benchmarks. For implementation reasons, there is no very easy way to count the symbols including duplicates. So, only the distinct symbols are taken into account. This is regarded as a sufficient approximation for interpreting the results.

The MPTP2078 benchmark in the bushy version has between 2 and 232 axioms per problem. Its average is 31.5 axioms with a standard deviation of 23.4. It has between 2 and 93 distinct symbols per problem with an average of 18.8 and a standard deviation of 12.1. Counting the distinct symbols per axiom, there are between 0 and 26 symbols, 2.9 on average with a standard deviation of 2.7.

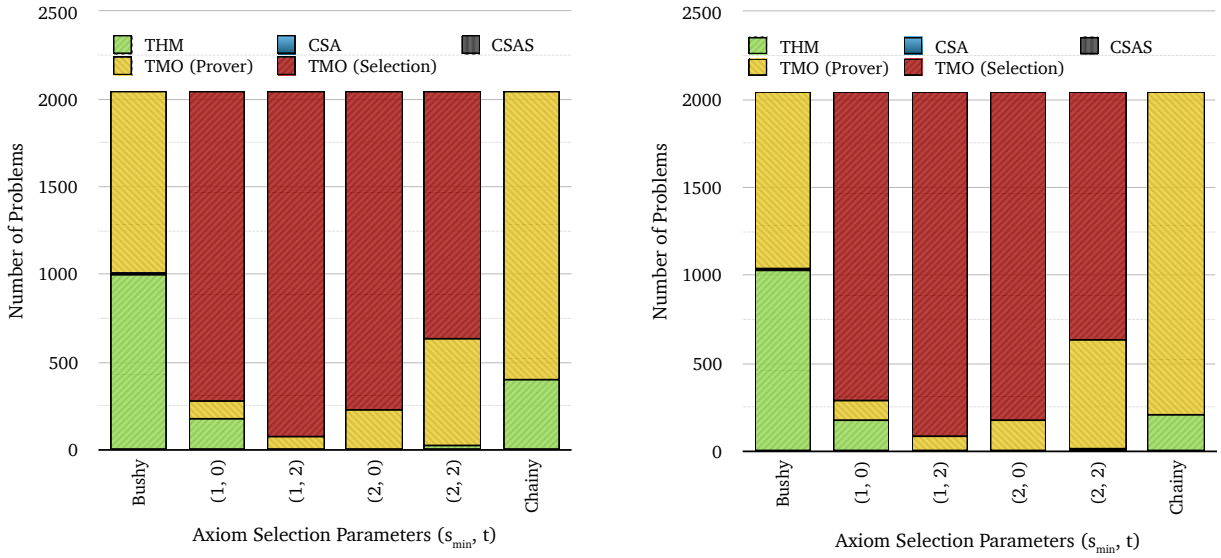
The chainy version of the MPTP2078 has between 10 and 4563 axioms per problem. The average axiom count is 1876.4 with a standard deviation of 1288.6. The number of distinct symbols per problem ranges from 5 to 784 with an average of 266.7 and a standard deviation of 229.2. In the axioms, there are between 0 and 26 distinct symbols, 4.9 on average with a standard deviation of 3.1.

5.3 Discussion

SInE.

Inspecting Figure 1, one can see that the E Theorem Prover improves by about 50% for most of the parametrisations of SInE (45% on average) compared to the chainy versions without any selection. SPASS also performs much better with a selection than without one: Its manages to solve 64% more problems on average across the SInE parametrisations. As a side note, the E Theorem Prover performs better than SPASS, even when comparing the E Theorem Prover without preselecting axioms against the best result with the selection for SPASS. Considering all SInE parametrisations, 769 problems could be solved, 330 of them could not be solved without an axiom selection. Without axiom selection, 444 problems could be solved.

For both provers, the number of 'CSA' and 'CSAS' results is negligible. For the largest tolerance value without a depth limit, meaning $(t, d) = (2.0, \infty)$, the axiom selection takes more than two minutes for 294 examples. These examples have 2165.5 axioms on average with a standard deviation of 1680.1, ranging from 51 to 4563 and 332.5 distinct symbols on average with a standard deviation of 298.7, ranging from 12 to 784. This means that not only the largest problems take long during axiom selection but also some of the small problems. Having a timeout on some of the small problems may be a result of a high CPU load caused by other tasks performed on the hardware. It is imaginable that the timeout at the larger problems is due to the problem size, though.



(a) E Theorem Prover

(b) SPASS

Figure 3: Performance of the provers in MPTP2078 with the Simple Axiom Selection using Frequent Item Set Mining

SInE with Frequent Symbol Set Mining.

Figure 2 clearly shows that there is no proving performance gain by SInE with Frequent Symbol Set Mining. Not only do both provers perform maximally as well as without the axiom selection but also the selection itself is cancelled frequently due to timeouts. The number of 'CSAS' results increases for some parametrisations, but regarding there is such a high amount of selection timeouts, this can be ignored. In total, 475 MPTP2078 problems could be solved using SInE with Frequent Symbol Set Mining, only 35 of which were not solved without an axiom selection.

Simple Axiom Selection using Frequent Item Set Mining.

Figure 3 shows that the timeout of the axiom selection is dominating the results. This means that for most of the problems, two minutes of selection time are not enough. Only 239 MPTP2078 problems could be solved in total with this method, 9 of which could not be solved without a preselection in the chainy versions.

Selection Runtime.

Figure 4 shows the average runtime over all parametrisations of the presented selection methods. When only considering the problems that did not reach a timeout (red/left columns), SInE is the slowest method with an average runtime of 18.6 seconds per problem. This is not a fair comparison though, because the other two methods often reached the two-minute timeout while SInE did not. When counting two minutes for every timeout-reached selection (which actually passed during the selection), SInE achieves the best result (yellow/right columns). Two minutes are the smallest value that can be used without distorting the data too much. The distortion caused by counting two minutes is in favour of the axiom selection methods using frequent item set mining which are slower nevertheless. A runtime of several hours for a single problem was observed with SInE with Frequent Symbol Set Mining, before a timeout was implemented. This time was mainly consumed by the external frequent item set mining process. The same behaviour is suspected for the Simple Axiom Selection using Frequent Item Set Mining. Since the MPTP2078 comprises large theories, these two slower methods are again considered unsuitable.

Number of Selected Axioms.

Figure 5 shows the average number of selected axioms using SInE. When the depth limit is set to 5, considerably less axioms get selected than without a limitation in MPTP2078. Also, the difference between infinite depth

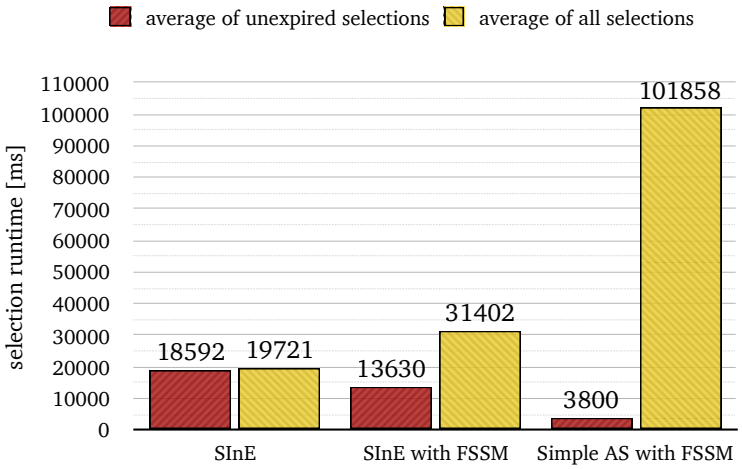


Figure 4: Runtime of axiom selection methods

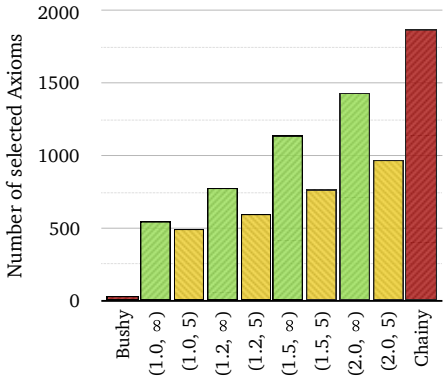


Figure 5: Average number of selected axioms by SInE

limit and a limit of 5 increases with a growing tolerance value. As expected, the number of selected axioms highly increases with a growing tolerance value.

Conclusion.

Considering the evaluation results, of the three selection methods, SInE clearly is the only usable method for large theories. The methods based on frequent symbol set mining take too much time for large problems and sometimes even for small problems. Even if the selection itself finishes quickly, the selected axioms do not enable proving considerably more conjectures. SInE on the other hand, allows to prove significantly more conjectures. For this reason, only SInE will be kept in Ontohub.

6 Related Work

Axiom selection was actively researched in the recent years. A simple algorithm called RedAx [24] was developed to provide evidence for the effectiveness of preselected axioms when proving. SRASS [25] is a more sophisticated semantically operating algorithm to find the right axioms for a proof. MePo [13] as part of Isabelle/Sledgehammer [16, 18] which is adding axioms that share a feature with the conjecture or already added axioms in several iterations. In each iteration, a pass mark is increased which makes it harder for axioms to be added to the selection. This method was followed by MaSh [11] to improve Isabelle/Sledgehammer's axiom selection with machine learning methods that rank axioms by the amount and kind of features they share with the conjecture and takes into account previous proofs in the same theory. Alama et al. [3] use learning in MOR, in particular kernel methods, to determine how similar sentences are to each other and uses these similarity values to find feature weights and to eventually rank the available axioms. SInE [10] is a heuristic that, similarly to MePo, iteratively adds axioms to the selection that share symbols with the conjecture or already selected axioms. Instead of increasing a pass mark, it only selects axioms that contain symbols whose number of occurrences in the overall theory meets certain requirements. This method proved very successful by executing fast and enabling about as many proofs as some learning methods.

Because of saving intermediate information in a database, our implementation of SInE is very efficient for consecutive runs, although because of the implementation in Ruby and because of being bound to a relational database the first run performs slower than an implementation with native/compiled code such as it exists directly in Vampire or E. One notable feature is the prover-independent operability of the axiom selection. This implementation is usable with any prover that interfaces with Hets.

7 Conclusions and Future Work

An implementation of axiom selection heuristics was presented: the already existing SInE and two methods based on frequent item set mining. These are implemented in the combination of the web application Ontohub and Hets and run independently of the selected prover, which is a first step towards an implementation of logic-independent axiom selections in Ontohub. Currently, this implementation is compatible to every Hets-interfaced

first-order prover. The evaluation of the selection methods showed once again the effectiveness of the SInE algorithm but also revealed that frequent item set mining is not an appropriate tool for selecting axioms because of the slow computation of frequent item sets.

Future work in this field should extend the feature of independence such that the axiom selections operate independently of the logic as well. A new evaluation could feature a comparison between SInE as it is implemented in provers to the SInE implementation of Ontohub, measuring the performance of consecutive axiom selections that are applied to the same theory (with different conjectures) such that the commonness values can be read from the database. Also learning methods could be incorporated into the selection, e.g. learning the best parameters for SInE or entirely learning-based methods similar to MaSh or MOR could be incorporated. This is very promising in the context of Ontohub because of its repository structure which stores information on every logical theory including proof attempts – both successful and failed.

References

[1] Charu C. Aggarwal and Jiawei Han. *Frequent Pattern Mining.* Springer Publishing Company, Incorporated, 2014.

[2] Rakesh Agrawal and Ramakrishnan Srikant. Fast Algorithms for Mining Association Rules. In *Proc. 20th int. conf. very large data bases, VLDB*, volume 1215, pages 487–499, 1994.

[3] Jesse Alama, Tom Heskes, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. *Journal of Automated Reasoning*, 52(2):191–213, 2014.

[4] Egidio Astesiano, Michel Bidoit, Hélene Kirchner, Bernd Krieg-Brückner, Peter D Mosses, Donald Sannella, and Andrzej Tarlecki. CASL: The Common Algebraic Specification Language. *Theoretical Computer Science*, 286(2):153–196, 2002.

[5] Peter Baumgartner, Alexander Fuchs, and Cesare Tinelli. Darwin: A Theorem Prover for the Model Evolution Calculus. In Stephan Schulz, Geoff Sutcliffe, and Tanel Tammet, editors, *IJCAR Workshop on Empirically Successful First Order Reasoning (ESFOR (aka S4))*, Electronic Notes in Theoretical Computer Science, 2004.

[6] Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In *Automated Reasoning*, pages 107–121. Springer, 2010.

[7] Christian Borgelt. Frequent Pattern Mining. Lecture slides. URL `http://www.borgelt.net/teach/fpm/slides.html`. last checked: 13 June 2016.

[8] Peter Haase, Holger Lewen, Rudi Studer, Duc Thanh Tran, Michael Erdmann, Mathieu dAquin, and Enrico Motta. The Neon Ontology Engineering Toolkit. *WWW*, 2008.

[9] Jiawei Han, Jian Pei, and Yiwen Yin. Mining Frequent Patterns Without Candidate Generation. In *ACM Sigmod Record*, volume 29, pages 1–12. ACM, 2000.

[10] Krystof Hoder and Andrei Voronkov. Sine Qua Non for Large Theory Reasoning. In *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, pages 299–314, 2011.

[11] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine Learning for Sledgehammer. In *Interactive Theorem Proving*, pages 35–50. Springer, 2013.

[12] Eugen Kuksa and Till Mossakowski. Ontohub — version control, linked data and theorem proving for ontologies. In *FOIS 2016*. To appear.

[13] Jia Meng and Lawrence C Paulson. Lightweight Relevance Filtering for Machine-Generated Resolution Problems. *Journal of Applied Logic*, 7(1):41–57, 2009.

[14] Till Mossakowski. Heterogeneous Specification and the Heterogeneous Tool Set. University of Bremen Germany, 2005. Habilitation thesis.

[15] Till Mossakowski, Mihai Codescu, Fabian Neuhaus, and Oliver Kutz. The Distributed Ontology, Modeling, and Specification Language - DOL. In A. Koslow and A. Buchsbaum, editors, *The Road to Universal Logic*, volume II of *Studies in Universal Logic*. Springer, 2015.

[16] Tobias Nipkow, Lawrence C Paulson, and Markus Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283. Springer Science & Business Media, 2002.

[17] Natalya F Noy, Nigam H Shah, Patricia L Whetzel, Benjamin Dai, Michael Dorf, Nicholas Griffith, Clement Jonquet, Daniel L Rubin, Margaret-Anne Storey, Christopher G Chute, et al. BioPortal: Ontologies and Integrated Data Resources at the Click of a Mouse. *Nucleic acids research*, page gkp440, 2009.

[18] Lawrence C Paulson and Jasmin Christian Blanchette. Three Years of Experience With Sledgehammer, a Practical Link Between Automatic and Interactive Theorem Provers. *IWIL-2010*, 2010.

[19] Alexandre Riazanov and Andrei Voronkov. The Design and Implementation of VAMPIRE. *AI communications*, 15(2, 3):91–110, 2002.

[20] Piotr Rudnicki. An Overview of the Mizar Project. In *Proceedings of the 1992 Workshop on Types for Proofs and Programs*, pages 311–330, 1992.

[21] Stephan Schulz. System Description: E 1.8. In Ken McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Proc. of the 19th LPAR, Stellenbosch*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.

[22] G. Sutcliffe. The SZS Ontologies for Automated Reasoning Software. In G. Sutcliffe, P. Rudnicki, R. Schmidt, B. Konev, and S. Schulz, editors, *Proceedings of the LPAR Workshops: Knowledge Exchange: Automated Provers and Proof Assistants, and The 7th International Workshop on the Implementation of Logics*, number 418 in CEUR Workshop Proceedings, pages 38–49, 2008.

[23] Geoff Sutcliffe. The 4th ijcar automated theorem proving system competition–casc-j4. *Ai Communications*, 22(1):59–72, 2009.

[24] Geoff Sutcliffe and Alexander Dvorskỳ. Proving Harder Theorems by Axiom Reduction. In *FLAIRS Conference*, pages 108–113, 2003.

[25] Geoff Sutcliffe and Yury Puzis. SRASS - A Semantic Relevance Axiom Selection System. In *Automated Deduction–CADE-21*, pages 295–310. Springer, 2007.

[26] Geoff Sutcliffe, Jürgen Zimmer, and Stephan Schulz. TSTP Data-Exchange Formats for Automated Theorem Proving Tools. *Distributed Constraint Problem Solving and Reasoning in Multi-Agent Systems*, 112:201–215, 2004.

[27] Linus Torvalds and Junio Hamano. Git: Fast Version Control System, 2010. `http://git-scm.com` last checked: 13 June 2016.

[28] Christoph Weidenbach, Dilyana Dimova, Arnaud Fietzke, Rohit Kumar, Martin Suda, and Patrick Wischnewski. SPASS Version 3.5. In *Automated Deduction–CADE-22*, pages 140–145. Springer, 2009.

[29] Mohammed Javeed Zaki, Srinivasan Parthasarathy, Mitsunori Ogihara, and Wei Li. New Algorithms for Fast Discovery of Association Rules. In *KDD*, volume 97, pages 283–286, 1997.

[30] Jürgen Zimmer and Serge Autexier. The MathServe System for Semantic Web Reasoning Services. In *Automated Reasoning*, pages 140–144. Springer, 2006.