Informatik
Hauptcampus

HOCHSCHULE TRIER

Context-specific Selection: Using Large Language Models to Select Commonsense Knowledge from Knowledge Bases

Oliver Jakobs

Bachelor-Abschlussarbeit

Betreuer: Prof. Dr. Claudia Schon

Bassenheim, December 11, 2023

Abstract

For large and complex knowledge bases, it is often necessary to select a subset of axioms for an automated theorem prover to efficiently solve a task. Especially in commonsense reasoning, the semantics of symbol names constitutes a valuable source of information that can be leveraged in the selection process. We introduce a new selection strategy based on the vector-based selection that uses sentence embeddings to encode axioms. For this, we have developed a compiler capable of generating sentences for axioms expressed in first-order logic. We also propose a variant of the vector-based approach that employs a threshold in the selection process. This threshold specifies how similar to the goal an axiom needs to be for it to be selected. Furthermore, we present a hybrid approach that incorporates the semantics similarities of symbol names into the syntactic selection strategy SInE. These new approaches are evaluated using the commonsense ontology Adimen-SUMO and the superposition-based theorem prover E.

Contents

1 Introduction

In automated reasoning, the goal is to automatically derive new insights from existing knowledge. The existing knowledge is represented by a set of formulas or axioms, also known as a knowledge base. An automated theorem prover can be used to determine if new knowledge follows from the known information. In many applications, the background knowledge is too large to be fully processed by such a prover within a realistic time frame. However, typically only a small subset of the axioms in a knowledge base is needed to solve a task. This observation has led to the development of various selection techniques, that try to find such a subset to solve a given problem. One common selection strategy is SInE [HV11], which selects axioms based on symbols occurring in the query. SInE has proven to be a valuable tool and is employed by many automated theorem provers.

In commonsense ontologies like Adimen-SUMO [ÁLR12], the names of symbols are often based on natural language words and therefore have a semantic meaning. Since SInE is a purely syntactic selection technique, any meaning a symbol name may carry is ignored. But this meaning can constitute a valuable information source during the selection process. For example, in the context of querying about *dogs*, axioms about *wolves* may be more relevant than those about *cars*.

An approach that considers the semantic meaning of symbol names is the vector-based selection, where axioms are selected based on the calculated similarity of their vector representation. One way to vectorize an axiom is the use of word embeddings, which are mappings learned from large text corpora using neural networks. A word embedding assigns vectors in $\mathbb{R}^n$ to words of an alphabet so that words occurring in a similar context are also assigned similar vectors. Such a selection strategy is introduced in [Sch23]. It uses a word embedding to encode symbols and combines these symbol vectors to get the final vector representation of an axiom.

In this thesis, we extend this idea and introduce a new vector-based approach, called SeVEn (**Se**ntence-based **V**ector **En**coding), which does not only consider individual symbols but the axiom as a whole. We implement this by using a sentence embedding. A sentence embedding works analogously to a word embedding. Instead of words, whole sentences are mapped to a vector in $\mathbb{R}^n$. To be able to encode an axiom with a sentence embedding, it first must be translated into a sentence in a natural language (in our case, the English language). For that purpose, we implement a compiler that generates a sentence for a given axiom.

In addition to the standard vector-based selection, we also introduce a variant that uses a threshold in the selection process. Instead of selecting a fixed number of axioms, any axiom whose similarity to the problem exceeds the threshold is selected. Furthermore, we present a hybrid selection approach that combines the syntactic selection SInE with SeVEn. It follows the idea of Similarity SInE [FKS19] and incorporates both the syntax of an axiom as well as the semantic meaning of symbol names into the selection process.

Through experiments, we conclude that while SeVEn can produce results comparable to SInE, we do not believe that this approach provides any benefit in most use cases due to the complex encoding and preprocessing step. We also observe that SeVEn with thresholds performs significantly worse than the regular SeVEn selection. The hybrid approach, both with and without thresholds, achieves the best results, outperforming its components, SeVEn and SInE.

The main contributions of this thesis are:

- A mapping that assigns all predicate and function symbols in Adimen-SUMO to a sentence fragment in the English language, thereby enabling the translation of all axioms in Adimen-SUMO.
- The introduction and evaluation of a new vector-based selection strategy for commonsense knowledge that utilizes sentence embeddings to encode axioms.
- A variant of vector-based selection that incorporates a threshold parameter in the selection process.
- The development of a hybrid approach that combines SeVEn with the syntactic selection strategy SInE.

This work is structured as follows: we start by discussing related work in Chapter 2 and preliminaries in Chapter 3. Next, we briefly revise the SInE selection strategy in Chapter 4. Then we show how vector-based selection uses statistical information in the selection process in Chapter 5. In the next chapter, we introduce SeVEn, both with and without thresholds, along with the hybrid approach SeVEn-union-SInE. In Chapter 7, we explain how axioms are translated into sentences. Finally, in Chapter 8, we present and evaluate the results of our experiments. We conclude by discussing future work.

2 Related Work

In this thesis, we focus on reasoning tasks where we want to show, using an automated theorem prover, that a conjecture is the logical consequence of existing background knowledge. These knowledge bases, such as Adimen-SUMO [ÁLR12], can grow quite large and complex. Therefore, it becomes necessary to select a subset of relevant axioms for the prover to still be able to solve the task at hand.

The selection of this subset is not trivial, and many selection strategies are incomplete. This means it is not always possible to find a proof for a problem with the selected axioms, even if there is one. The majority of selection strategies are purely syntactic, like [HV11], [MP09], and [RPS09]. These strategies only take the syntax of axioms into account and ignore the semantic information they may contain.

A widely used syntactic selection strategy is SInE [HV11]. SInE counts the occurrences of symbols in the knowledge base to determine how relevant an axiom is to the task at hand. For a more in-depth explanation of how SInE works, we refer to Chapter 4.

A semantic approach for axiom selection is presented in [SP07]. This approach is based on the computation of models for subsets of the available axioms and by consecutively extending these sets. Three more selection techniques are introduced in [LWWS20]. The authors of this paper also propose metrics for the evaluation of selection techniques. None of these selection strategies take the meaning of symbol names into account.

A strategy which considers the meaning of symbol names is Similarity SInE [FKS19], an extension to the syntactic selection strategy SInE [HV11]. While SInE selects axioms only based on the number of axioms a symbol occurs in, Similarity SInE extends this selection by using a word embedding to take the similarity of symbols into account. The combination of syntactic and statistical methods makes Similarity SInE a hybrid selection approach.

A purely statistical approach is presented in [Sch23]. This vector-based approach uses a word embedding to select axioms based on the relative similarities of their vector representations. To achieve this, the axioms are vectorized by first encoding the individual symbol names with a word embedding and then combining these symbol vectors to obtain the final vector representation. The vector-based approach is discussed in more detail in Chapter 5.

The question if names are meaningful is evaluated in [dRBB+16]. The authors prove that the social semantics encoded in the names of IRIs (Internationalized Resource Identifiers) are meaningful.

3 Preliminaries

In the reasoning tasks we focus on in this thesis, knowledge is represented as axioms. Existing background knowledge, from which we aim to derive new insights, is gathered in a set of axioms commonly referred to as knowledge base (KB).

Axioms in our context are formulas in first-order logic (FOL), which is a generalization of propositional logic. FOL allows the use of quantifiers and predicates, enabling more expressive reasoning capabilities.

The syntax of the formal language defined by FOL consists of an alphabet and a set of formation rules. The alphabet specifies the symbols that can appear in a formula, while the formation rules determine the order in which these symbols can be arranged to construct well-formed formulas.

Definition 3.1 (Alphabet). *The* Alphabet *defines the set of symbols that are used to construct formulas in first-order logic. These symbols are divided into* ***logical symbols****:*

- *Logical connectives* ($\wedge, \vee, \neg, \Rightarrow, \Leftrightarrow$)
- *Quantifier symbols* ($\forall, \exists$)
- *Parentheses and other structural symbols.*
- *An infinite set of variable symbols.*

and ***non-logical symbols****, also called* ***signature****:*

- *A set $\mathcal{P}$ of predicate symbols with arity $n \geq 1$.*
- *A set $\mathcal{F}$ of function symbols with arity $n \geq 0$.*

Function symbols with an arity of 0 are called *constants*. Since they do not take any arguments, the parentheses after the symbol name can be omitted. Some definitions also include such a case for 0-arity predicates and call them *propositional variable*. However, we only deal with predicates with an arity > 0, so we do not include them in our alphabet. Furthermore, the equality operator ($=$) is often included in the set of logical symbols, but for our purposes, we consider it to be a 2-arity predicate written in infix notation.

The names of non-logical symbols and variables are dependent on the author of the system. Ontologies typically adhere to their own naming convention. For instance, Adimen-SUMO uses all capital letters for variable symbols while predicate and function symbols start with a prefix indicating the symbol type. The actual symbol name is then written in either *PascalCase* or *camelCase*.

Throughout this work, we will use the notation $\mathcal{S}_{\text{KB}}$ to represent the signature of the KB. We also extend that notation and use $\mathcal{S}_{\text{A}}$ and $\mathcal{S}_{\text{G}}$ for the set of all predicate and function symbols occurring in an axiom A and a goal G, respectively.

The first set of rules describe how *terms* are constructed. Terms are recursively constructed using variable and function symbols and form the basic building block of formulas.

Definition 3.2 (Terms [HR04]). *Terms are defined as follows.*

- *Any variable is a term.*
- *If $c \in \mathcal{F}$ is a 0-arity function, then c is a term.*
- *If $t_1, t_2, ..., t_n$ are terms and $f \in \mathcal{F}$ has arity $n > 0$, then $f(t_1, t_2, ..., t_n)$ is a term.*
- *Nothing else is a term.*

With the set of valid terms constructed using the rules from Definition 3.2, we can then define the formation rules for formulas.

Definition 3.3 (Formulas [HR04]). *We define the set of formulas over ($\mathcal{F}$, $\mathcal{P}$) inductively, using the already defined set of terms over $\mathcal{F}$:*

- *If $P \in \mathcal{P}$ is an predicate symbol with arity $n \geq 1$ and if $t_1, t_2, ..., t_n$ are terms over $\mathcal{F}$, then $P(t_1, t_2, ..., t_n)$ is a formula.*
- *If F is a formula, then so is $\neg F$.*
- *If F and G are formulas, then so are $(F \wedge G)$, $(F \vee G)$, $(F \Rightarrow G)$ and $(F \Leftrightarrow G)$.*
- *If F is a formula and x is a variable, then $(\forall x F)$ and $(\exists x F)$ are formulas.*
- *Nothing else is a formula.*

With the alphabet and this set of formation rules, we can now construct formulas that adhere to the syntax of FOL. However, these formulas still lack meaning, and thus we can not evaluate their truth value (`True` or `False`). To assign meaning to a formula, we have to interpret every symbol it contains. While the interpretation of logical symbols is defined by the logical operation they represent, interpreting non-logical symbols is a more involved process.

If, for example, our universe `A` is the set of real numbers, a possible interpretation for a 2-arity function $f \in \mathcal{F}$ could be the addition of its two arguments. Furthermore, a 2-arity predicate $P \in \mathcal{P}$ could be interpreted as the relation *greater than*. To assign a truth value to that predicate, we first have to evaluate both its argument terms.

Before we can give meaning to any non logical symbol, we first have to establish the domain of discourse or *universe* of our system. The universe is the set of all concrete values that a term can represent, i.e. all values that a variable may be assigned and all values that functions may return. With such an universe, we can then create an *interpretation* or *model* for a formula.

Definition 3.4 (Model [HR04]). *Let $\mathcal{F}$ be a set of function symbols and $\mathcal{P}$ a set of predicate symbols, each symbol with a fixed number of required arguments. A model $\mathcal{M}$ of the pair $(\mathcal{F}, \mathcal{P})$ consists of the following set of data.*

1. *A non-empty set* $\mathtt{A}$*, the universe of concrete values;*
2. *for each 0-arity function symbol $f \in \mathcal{F}$, a concrete element $f^{\mathcal{M}}$ of* $\mathtt{A}$*;*
3. *for each $f \in \mathcal{F}$ with arity $n > 0$, a concrete function $f^{\mathcal{M}} \colon \mathtt{A}^n \to \mathtt{A}$ from $\mathtt{A}^n$, the set of n-tuples over* $\mathtt{A}$*, to* $\mathtt{A}$*; and*
4. *for each $P \in \mathcal{P}$ with arity $n > 0$, a subset $P^{\mathcal{M}} \subseteq \mathtt{A}^n$ of n-tuples over* $\mathtt{A}$*.*

In addition to a model $\mathcal{M}$, we also need a mapping that assigns a concrete value of an universe $\mathtt{A}$ to every variable in a formula. This mapping is also called *environment* for $\mathtt{A}$.

Definition 3.5 (Environment for an universe $\mathtt{A}$ [HR04]). *A look-up table or environment for an universe $\mathtt{A}$ of concrete values is a function $l \colon \mathtt{var} \to \mathtt{A}$ from the set of variables $\mathtt{var}$ to $\mathtt{A}$. For such an l, we denote by $l[x \mapsto a]$ the look-up table which maps x to a and any other variable y to $l(y)$.*

Equipped with a model $\mathcal{M}$ for a pair $(\mathcal{F}, \mathcal{P})$ and an environment $l[x \mapsto a]$, we can calculate the truth value of formulas in FOL. For a more in-depth explanation of the semantics of FOL, we refer to [HR04].

If a formula f is true under a model $\mathcal{M}$, we say that $\mathcal{M}$ *satisfies f*. Conversely, if there is no model that satisfies a formula f, then f is considered *unsatisfiable*. Furthermore, a *tautology* is a formula that is true under every possible interpretation. In other words, a formula is a tautology if its negation is unsatisfiable.

Building upon the satisfiability of formulas, we introduce the concept of entailment. A formula g is entailed by a formula f, or g logically follows from f, if every interpretation that satisfies f also satisfies g. This is denoted as $f \models g$. We can also express this relationship as g being the logical consequence of f.

With a basic understanding of FOL, we can now define the tasks we want to solve in this thesis. Given a large and complex KB and a goal, our objective is to prove that the goal is entailed by the KB. The goal comprises a set of assumptions $A_1, ..., A_n$ and a conjecture C. Following the notation established in [HV11], we formally define this task as KB $\models (A_1 \wedge ... \wedge A_n \Rightarrow C)$, where $A_1 \wedge ... \wedge A_n \Rightarrow C$ denotes the goal. Since in our scenario, the set of assumptions is always empty, we only need to show that $KB \models C$.

Automated theorem provers (ATPs) can be used to prove that a given goal is the logical consequence of the axioms in a KB. To find such a proof, the ATP starts with the conjecture and consecutively applies inference rules to generate new statements.

If the KB is too large or complex, the ATP may not be able to process it as a whole and thus may not be able to find a proof. However, in most cases, only a small subset of the KB is needed to solve the task at hand. So it becomes necessary to select a preferably small subset of axioms in the KB with which the ATP can find a proof.

4

Syntax-Based Selection: SInE

The SInE selection strategy [HV11] is a trigger-based approach used successfully by many automated theorem provers. It uses a *trigger* relation to determine which symbols can *trigger* and therefore select an axiom during the selection process. This relation is defined as follows:

Definition 4.1 (Trigger relation for the SInE selection [HV11]). *Let KB be a knowledge base, $A \in KB$ be an axiom and $s \in \mathcal{S}_A$ be a symbol. Let furthermore $occ(s)$ denote the number of axioms in which the symbol s occurs in KB. Then the trigger relation is defined as follows:*

$$trigger(s, A) \text{ iff for all symbols } s' \text{ occurring in } A \text{ we have } occ(s) \leq occ(s').$$

This relation ensures that only symbols present within an axiom can trigger its selection. Additionally, an axiom A can only be triggered by a symbol s if every other symbol s' in A has a higher or equal number of occurrences in the KB. These restrictions prevent frequently occurring symbols, like `instance` and `subclass`, from triggering every axiom they appear in, thus preventing them from dominating the selection process.

With this trigger relation in place, the selection process for SInE can be defined.

Definition 4.2 (Trigger-based selection [HV11]). *Let KB be a knowledge base, $A \in KB$ be an axiom and $s \in \mathcal{S}_{KB}$ be a symbol. Let furthermore G be a goal to be proven from KB.*

1. *If s is a symbol occurring in G, then s is 0-step triggered.*
2. *If s is k-step triggered and s triggers A, then A is $k+1$-step triggered.*
3. *A is k-step triggered and s occurs in A, then s is k-step triggered, too.*

An axiom or a symbol is called triggered if it is k-triggered for some $k \geq 0$.

We refer to the selection of all k-step triggered axioms as SInE with recursion depth k throughout the rest of this thesis.

A limitation of the trigger relation presented in Definition 4.1 arises when two symbols in an axiom have a nearly equal number of occurrences in the KB. In such instances, only the symbol with the least occurrences is allowed to trigger the axiom. To diminish this effect, the trigger relation is modified to include the tolerance or benevolence parameter, a real number $b \geq 1$.

Definition 4.3 (Trigger relation with benevolence [HV11]). *Given the benevolence $b \geq 1$, define the relation trigger as follows:*

$$\textit{trigger}(s, A) \textit{ iff for all symbols } s' \textit{ occurring in } A \textit{ we have } \textit{occ}(s) \leq b \cdot \textit{occ}(s').$$

The benevolence parameter b controls the strictness of the selection and enables symbols that occur b times more often than the rarest symbol to trigger an axiom. By adjusting the recursive depth k and the benevolence parameter b, the selection can be tailored to specific use cases and produce better results.

Considering the complexity of the SInE selection process, which may seem overwhelming at first glance, we want to demonstrate the selection process through an example. For this, we construct a small KB similar to the the example in [HV11] and go over each step involved.

The axioms in our example KB are real axioms from the ontology Adimen-SUMO, but we remove any prefixes the non-logical symbol names have.

```
A.1: ∀ X,Y,Z:(subclass(X,Y)∧subclass(Y,Z))⇒subclass(X,Z)
A.2: subclass(carnivore,mammal)
A.3: subclass(rodent,mammal)
A.4: subclass(canine,carnivore)
A.5: subclass(feline,carnivore)
```

Furthermore, we consider the goal for this example to be `subclass(canine, mammal)`.

The first step in the selection process is to count the number of axioms in which each symbol in our example KB occurs.

symbol s	**occurrences** *occ(s)*
subclass	5
mammal	2
carnivore	3
rodent	1
canine	1
feline	1

Table 4.1: Number of axioms in which each symbol occurs in the example KB.

Next, we determine which symbols are allowed to trigger the axioms in the example KB, using the number of occurrences for each symbol listed in Table 4.1.

axiom *A*	**symbols that can trigger *A***
`A.1`	`subclass`
`A.2`	`mammal`
`A.3`	`rodent`
`A.4`	`canine`
`A.5`	`feline`

Table 4.2: Symbols that can trigger each axiom.

Table 4.2 lists all symbols s for an axiom A, so that the trigger relation *trigger(s,A)*, as defined in Definition 4.1, holds true.

To keep this example simple, we are not using the benevolence parameter in the trigger relation. For the same reason, we use a recursion depth of *1*.

We now select all *1*-step triggered axioms using the symbols in the goal (`subclass`, `canine`, `mammal`).

```
∀ X,Y,Z:(subclass(X,Y)∧subclass(Y,Z))⇒subclass(X,Z)
subclass(canine,carnivore)
subclass(carnivore,mammal)
```

With this selected subset, we can then try to find a proof. In this case, the selected axioms are sufficient, and it is possible for the ATP to find a proof for the goal.

5 Vector-Based Selection

Since SInE is a syntax-based selection the meaning of symbol names is not taken into account which leads to potentially helpful information for the selection being lost. This is especially relevant in commonsense reasoning where the symbol names carry a semantic meaning that can be leveraged to improve the selection.

To quantify the semantic similarities between words, a common approach in natural language processing (NLP) is using distributional semantics of natural language. The concept of distributional semantics is based on the distributional hypothesis [MC91], which posits that words with similar distributional properties in large text corpora also have similar meanings. In simpler terms, words that appear in a similar context are likely to have a similar meaning.

A technique used in NLP that employs the distributional hypothesis is the use of word embeddings [MSC+13]. Word embeddings are mappings, obtained through the training of neural networks on extensive text corpora, that assign vectors in $\mathbb{R}^n$ to words in a vocabulary. These vectors have the property that their relative similarities directly translate to the semantic similarity of the corresponding words. A common way to measure this similarity between two vectors is the *cosine similarity.*

Definition 5.1 (Cosine similarity of two vectors [Sch23]). *Let $u, v \in \mathbb{R}^n$ be non-zero vectors. The cosine similarity of u and v is defined as:*

$$cos\_sim(u, v) = \frac{u \cdot v}{\|u\| \; \|v\|}$$

The cosine similarity of two vectors lies in the interval $[-1, 1]$ and depends solely on the angle between the two vectors and not their magnitude. A similarity value of -1 indicates that two vectors are opposite, while a value of 1 denotes that they are identical. If the value is 0, the vectors are orthogonal. In summary, the cosine similarity of two vectors increases with their similarity.

In [Sch23], a purely statistical selection approach is introduced. It uses a word embedding to encode axioms into a vector representation, aiming to map similar axioms to similar vectors. In an one-time preprocessing step, every axiom in the KB is vectorized using a word embedding. To now select axioms for a given goal G, vector-based selection first encodes G in the same way and using the same word embedding as the encoding of the KB. Then, it selects k axioms corresponding to the vectors most similar to the vector representation of the goal. To measure the similarity between two vectors, the cosine similarity is used.

Definition 5.2 (Vector-based selection [Sch23]). *Let KB be a knowledge base, G be a goal with $\mathcal{S}_G \subseteq \mathcal{S}_{KB}$ and $f\colon A \to \mathbb{R}^n$ a vectorization method for an axiom A. Let furthermore V_{KB} be a vector representation of KB and v_G a vector representation for G both constructed using f. For $n \in \mathbb{N}$, $n \leq |KB\,|$ the n axioms in KB most similar to G denoted as $mostsimilar(KB, G, n)$ are defined as the following set:*

$$\{A_1, ..., A_n \mid \{A_1, ..., A_n\} \subseteq KB \text{ and } \forall A' \in KB \setminus \{A_1, ..., A_n\} \\ cos\_sim(v_{A'}, v_G) \leq \min_{i=1,...,n} cos\_sim(v_{A_i}, v_G)\}.$$

For KB, G and $n \in \mathbb{N}$ given as above described, vector-based selection selects $mostsimilar(KB, G, n)$.

This definition is intentionally kept general and does not specify how the vector representation of an axiom is generated. The naive approach to vectorize an axiom is to first encode every symbol in the axiom and then average these symbol-vectors to get the final vector representation. However, treating all symbols equally in this manner may not always be desirable.

Following the basic idea of SInE, where only the rarest symbol can trigger the selection of an axiom, it would be advantageous for symbols that occur frequently in the KB, such as `instance` or `subclass`, to contribute less in the vector representation compared to symbols that occur less often, like `carnivore` or `canine`.

To address this issue, the author of [Sch23] uses the *inverse document frequency* (idf) to control the influence a symbol has on the final vector representation. The idf is a common tool in the domain of information retrieval to measure the significance a word w has to a document d in a set of documents D. With this technique, the impact of a word that occurs frequently in D is reduced.

Definition 5.3 (Inverse document frequency (idf) [Sch23]). *Let D be a set of documents and w be a word that occurs in at least one document in D. The idf for such a word w w.r.t. D is defined as*

$$idf(w, D) = \log \frac{|D|}{|\{d \in D \mid w \text{ occurs in } d\}|}$$

The idf value of a word occurring in every document in D is equal to 0 and increases when the percentage of documents in which a word appears in decreases.

To apply this concept to automated reasoning, the KB is interpreted as a set of documents, with each axiom representing a separate document and each symbol representing a word. With this interpretation, the idf-based vector representation for an axiom in the KB can be defined as follows:

Definition 5.4 (idf-based vector representation of a axiom, a KB [Sch23]). *Let KB be a knowledge base with $KB = \{A_1, ..., A_n\}$, $n \in \mathbb{N}$, $A \in KB$ be an axiom, V be a vocabulary and $f\colon V \to \mathbb{R}^n$ a word embedding. Let furthermore $\mathcal{S}_A \subseteq V$. The idf value for a symbol $s \in \mathcal{S}_A$ w.r.t. KB is defined as*

$$idf(s, KB) = \log \frac{|KB\,|}{|\{A' \in KB \mid s \in \mathcal{S}_{A'}\}|}$$

The idf-based vector representation of A is defined as

$$v_{idf}(A) = \frac{\sum_{s \in \mathcal{S}_A} (idf(s, KB) \cdot f(s))}{\sum_{s \in \mathcal{S}_A} idf(s, KB)}$$

Furthermore, $V_{idf}(KB) = \{v_{idf}(A_1), ..., v_{idf}(A_n)\}$ denotes the idf-based vector representation of KB.

Definition 5.4 assumes that $\mathcal{S}_{\text{KB}}$ is a subset of the vocabulary V. In practice, this assumption is not suitable, necessitating the creation of a mapping that assigns each symbol in the KB a word in the vocabulary before the symbols names can be encoded.

Each unique combination of a KB and word embedding requires its own distinct mapping. One such mapping is described in [Sch22].

6 SeVEn: Sentence-Based Vector Encoding

Following the ideas presented in [Sch23], we want to further investigate how the semantic similarities of symbol names can be incorporated in the selection process. For that purpose, we introduce a new vector-based approach called SeVEn (**Se**ntence-based **V**ector **En**coding). In contrast to the previous approach, which only considers the symbol names as words and encodes them with a word embedding, we encode the axioms as a whole using a sentence embedding. Sentence embeddings are similar to word embeddings. They are also obtained through the training of neural networks on large text corpora, and their vectors have characteristics similar to those of word embeddings. However, unlike word embeddings, which are limited to individual words, sentence embeddings can encode entire sentences or even documents. For more information on sentence embeddings, we refer to [KZS+15] and [CYK+18].

Since SeVEn is a vector-based selection, we can use Definition 5.2 and only need to change the method with which an axiom is vectorized. This transformation requires two stages. First, the axiom needs to be translated into a sentence that describes the semantic meaning of the axiom. In the second stage, this sentence is then encoded using a sentence embedding.

Definition 6.1 (Sentence-based vector representation of a axiom, a KB). *Let KB be a knowledge base with* $KB = \{A_1, ..., A_n\}$, $n \in \mathbb{N}$. *The sentence-based vector representation* $v_S(A)$ *of an axiom* $A \in KB$ *can be constructed following these steps:*

1. *Translate the axiom* A *into a sentence* S *with* $t\colon A \to S$.
2. *Use the sentence embedding* $f\colon S \to \mathbb{R}^n$ *to encode the sentence* S.

Furthermore, $V_S(KB) = \{v_S(A_1), ..., v_S(A_n)\}$ *denotes the sentence-based vector representation of KB.*

Definition 6.1 assumes that axioms can be directly translated into sentences. This process, however, is not straightforward and requires further consideration. We refer to the next chapter, where the translation process is discussed in detail.

6.1 Selection with Thresholds

The standard vector-based selection has a particular flaw. For example, if the task is to select 100 axioms, only the first ten selected axioms may be actually similar to the goal. The remaining axioms are chosen solely to complete the task, which contradicts the concept of selecting based on similarity. To address this issue, we propose an alternative approach that incorporates a threshold parameter into the vector-based selection. This parameter prevents the selection of axioms that do not meet the specified similarity threshold.

Definition 6.2 (Vector-based selection with threshold). *Let KB be a knowledge base, G be a goal with $\mathcal{S}_G \subseteq \mathcal{S}_{KB}$ and $f\colon A \to \mathbb{R}^n$ a vectorization method for an axiom A. Let furthermore V_{KB} be a vector representation of KB and v_G a vector representation for G both constructed using f. For $t \in \mathbb{R}$, $t \in [-1, 1]$ all axioms in KB with a similarity to G of at least t are given as*

$$mostsimilar(KB, G, t) = \{A \mid A \in KB \text{ and } cos\_sim(v_A, v_G) \geq t\}.$$

Unlike the previous approach, the number of axioms selected using this method is not fixed and can vary greatly depending on the goal.

6.2 Combining Vector-Based Selection with Syntactic Selection

With selection based on semantic similarities, axioms containing only structural predicates, like *agent*, *instance* or *subclass*, are often disregarded, as they are not similar enough to the goal. However, these axioms describe the mathematical structure of the ontology and are often essential for the ATP to be able to find a proof. To address this shortcoming, statistical and syntactic selection can be combined to create a hybrid selection strategy.

One such hybrid approach is Similarity SInE [FKS19]. It incorporates semantic similarities between symbol names into the SInE selection by extending the trigger relation of SInE so that not only the rarest symbol s, but also symbols with an high enough cosine similarity to s can trigger the selection of an axiom.

This thesis introduces a new hybrid approach, *SeVEn-union-SInE*, which combines SeVEn with the syntactic selection strategy SInE. First, the goal is broadened by adding axioms selected with SeVEn as conjectures to it. In a second step, SInE then selects axioms based on all conjectures (including the original one).

In the following, we use $sine(\text{KB}, \{A_1, ..., A_n\}), n \in \mathbb{N}$ to denote the set of axioms obtained by applying the SInE selection on a set of conjectures $\{A_1, ...A_n\}$. Furthermore, we use $seven(\text{KB}, G)$ to denote the set of axioms selected with SeVEn for a goal G. For the sake of simplicity, we omit the configuration parameters for both selection strategies.

With this notation we can then formally define the hybrid selection approach as follows:

Definition 6.3 (SeVEn-union-SInE selection). *Let KB be a knowledge base and G be a goal with* $\mathcal{S}_G \subseteq \mathcal{S}_{KB}$*. The combined selection function union_select is then defined as:*

$$union\_select(KB, G) = sine(KB, G \cup seven(KB, G))$$

The selection strategies used in Definition 6.3 can be replaced by other suitable approaches.

7 Translating Axioms

In this chapter, we discuss how axioms can be translated and what steps are involved in the process. The knowledge base (KB) we use in this work is Adimen-SUMO [ÁLR12], an ontology for first-order reasoning. Adimen-SUMO provides its axioms in many different formats, including the TPTP language format [Sut17]. This format is maintained by the *Thousands of Problems for Theorem Provers* (TPTP) project and specifies an extensive formal grammar written in the *extended Backus–Naur form* (EBNF) that contains rules for many different logic systems. But since the axioms we want to translate are in first-order logic, we only care about that part of the grammar and can ignore rules for higher-order logic. The modified subset of rules we use for the translation is the following:

```
<formula>    ::= <binary> ;
<binary>     ::= <unit> ('<=>' | '=>') <unit>
               | <unit> ('|' unit>)*
               | <unit> ('&' <unit>)* ;
<unit>       ::= <unitary> | '~' <unit> ;
<unitary>    ::= <quantified> | <atomic> | '(' <formula> ')' ;
<quantified> ::= ('!' | '?') '[' <var_list> ']' ':' <unit> ;
<var_list>   ::= <variable> (',' <variable>)* ;
<atomic>     ::= <predicate> '(' <term_list> ')'
               | <term> '=' <term> ;
<term_list>  ::= <term> (',' <term>)* ;
<term>       ::= <variable> | constant
               | <function> '(' <term_list> ')' ;
<predicate>  ::= <lower_word> ;
<function>   ::= <lower_word> ;
<constant>   ::= <lower_word> ;
<variable>   ::= A-Z (A-Z | a-z | 0-9 | '_')* ;
<lower_word> ::= a-z (A-Z | a-z | 0-9 | '_')* ;
```

Using this grammar, we create a parser that generates an abstract syntax tree (AST) for an axiom. An AST is a tree representation of the structure of an axiom. This representation is *abstract* inasmuch as the tree does not depict the actual syntax one-to-one. Structural symbols often can be omitted if they are implied by the tree structure. We also transform the AST in such a way, that negations only occur in front of predicate symbols, to simplify the further translation process.

To clarify the parsing process, we show with an example how an axiom can be converted into a tree representation. We start with the following axiom:

$$\texttt{![A]: (subclass(A, canine) => } \sim\texttt{subclass(A, feline))} \tag{7.1}$$

This axiom is then parsed, and a corresponding AST, depicted in Figure 7.1, is generated. This AST presents Axiom (7.1) in such a way that every tree node contains a single symbol. Furthermore, every leaf of the tree may only contain a variable or a constant.

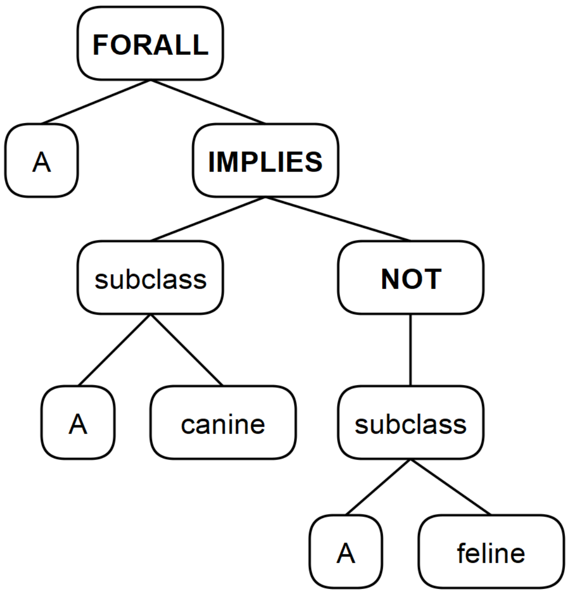

Fig. 7.1: Tree representation of Axiom (7.1).

To now get the final sentence, the AST is traversed with depth-first search to recursively translate every node. For that purpose, we need some kind of translation for every symbol a node may contain.

The mapping of symbol names to the words in a vocabulary of an word embedding, as described in Chapter 5, is not applicable to sentence embeddings. Fixing the number of sentences or words that can be used in these sentences greatly limits the usefulness of a sentence embedding. To alleviate this problem, the language model BERT [DCLT19] uses a WordPiece model [WSC$^+$16][SN12] which allows it to process sentences without being constrained by a fixed vocabulary. BERT still relies on a vocabulary but has a special technique to handle unknown words.

In a preprocessing step, the input sentence is transformed into tokens by the WordPiece tokenizer. These tokens are an internal representation of the words (and word-segments) in a sentence that the language model can work with. Instead of assigning an out of vocabulary word to an arbitrary *unknown vocabulary* token, the word is split into word-segments. Each segment is then represented by a separate token. This allows less common words, like `eating`, to be represented by the tokens `eat` and `##ing`, where `##` is a special marker to denote a word-segment. Since even individual characters can be represented this way, any sentence can be encoded with a finite vocabulary.

In this thesis, we use Sentence-BERT or SBERT [RG19] to encode the sentence representation of axioms. SBERT is a modification of BERT, which also benefits from the previously mentioned technique.

7.1 Translating Symbols

For logical symbols, excluding variables, we use the trivial English translation that describes the operation. Into these translations, we then insert the translations off all child-nodes in the corresponding locations. For example, the translation for the binary operation `implies (=>)` is defined as `"if LEFT, then RIGHT"` where `LEFT` and `RIGHT` denote the translations of what is left or right of the operator. On the other hand, variables are not translated at all; that is, the variable `X` translates to `"X"`.

Next, we define a translation for all non-logical symbols. For that purpose, we further differentiate between predicates, functions and constants. Adimen-SUMO contains 607 predicate symbols, 73 function symbols (excluding constants), and 3238 constants. While this may seem overwhelming, fortunately, the translation of all constants can be automated.

We differentiate between four types of constants, each with its own automated translation process. Since constants in Adimen-SUMO are prefixed with `c__`, it is possible for their actual name to begin with an upper case letter, without breaking any rules defined by the grammar. This prefix is removed before the constant symbol is assigned a type and processed further.

1. **Constants talking about functions:** Symbols ending on `"Fn"`. Translates to `"the function ..."` (e.g. the symbol `AssignmentFn` translates to `"the function AssignmentFn"`)
2. **Constants talking about predicates:** Symbols starting with a lower case letter. Translates to `"the predicate ..."` (e.g. the symbol `givenName` translates to `"the predicate givenName"`)
3. **Numbers:** These symbols are just numbers (e.g. `100`)
4. **Rest:** The remaining symbols are written in *PascalCase*. For the translation the name is split on capital letters (e.g. the symbol `AuditoriumSeat` translates to `"auditorium seat"`)

In contrast, functions and predicates require a mapping for translation. This mapping was created by hand with the help of the comments provided in Adimen-SUMO and the context in which the symbols are used.

The resulting translations are sentence fragments that include a special marker `{}` in places where the translated arguments of the predicate or function should be inserted. The special marker allows us to use Python's built-in functionalities for string formatting. We can also place numbers inside `{}` to specify the order in which the arguments should be inserted. This allows us to define a translation for every symbol with any number of arguments.

As an example, we illustrate the translation process for the predicate `between(X,Y,Z)` where `X`, `Y` and `Z` are variables. The symbol name `between` maps to the following sentence fragment:

`"{1} is between {0} and {2}"` (7.2)

By inserting the translated arguments of the predicate, this fragment can then be resolved to generate the following sentence:

`"Y is between X and Z"`

7.2 Negating Predicates

On top of the regular translations, we also need a negated translation for every predicate. In most cases, this can be automated with a simple pattern shown in Table 7.1. If a pattern (including the spaces before and after) occurs in the translation of a predicate, the negated pattern will be substituted to obtain the negated translation.

pattern	**negation**
`" is "`	`" is not "`
`" are "`	`" are not "`
`" has "`	`" does not have "`
`" have "`	`" do not have "`

Table 7.1: Substitution patterns to negate the translations of predicates

Taking the example from the last section, we now generate the negated translation for the sentence fragment (7.2). Using the substitution patterns defined in Table 7.1, the negated translation for the predicate `between(X,Y,Z)` is:

`"Y is not between X and Z"`

If the translation for a predicate does not contain any of the patterns defined in Table 7.1, the negation needs to be defined manually. For that purpose, we create a special syntax with which both the negated and non-negated translation can be specified. This syntax is defined as `(non-negated|negated)`. Occurrences of this pattern will be resolved during the translation process to generate the non-negated or negated form of the translation.

As an example, the predicate `contains` maps to the following sentence fragment:

```
"{} (contains|does not contain) {}"
```

This resolves to the negated form `"{} does not contain {}"` and the non-negated form `"{} contains {}"`.

It may seem unnecessary to create this special syntax instead of just adding a `"does not"` in front of the non-negated translation. The grammatical error of not removing the `-s` at the end of the verb should not affect the language model negatively. But there are other translations where such a syntax is required to create the correct negation.

For example, the translation for the predicate `externalImage` is defined as:

```
"the image at {1} (exemplifies|does not exemplify) {0}"
```

The special syntax allows the negation to occur in the middle of the sentence fragment. Furthermore, it is possible to have even more complex negations, like in the translation of the predicate `contraryAttribute2`:

```
"(nothing|things) can simultaneously be {} and {}"
```

Given that we have to implement this syntax anyway, we decided to use it for simpler negations as well. This allows us to keep the translations as grammatically correct as possible.

7.3 Putting it all together

To put all of this together, we demonstrate how such a translation would work using an example. For that, we use the Axiom (7.1) and its AST representation in Figure 7.1.

The translation for this axiom is generated by recursively translating every node in the AST. To illustrate this process, we translate the tree layer by layer, starting at the bottom.

Throughout this example, we depict the translated sentence fragments in quotation marks.

The first layer
In the first layer, we only encounter the variable `A` and the constant `feline`. For a variable the translation is simply its name. In this case, `"A"`.

The translation of constants is described in Section 7.1. Since `feline` is originally called `c__Feline` in Adimen-SUMO, it falls under the fourth type of constants (**Rest**: Symbols written in PascalCase) and is therefore translated as `"feline"`.

The second layer

The next layer again contains the variable `A` and a constant. This time, the constant is `canine`, which translates to `"canine"`. Additionally, this layer also contains the predicate `subclass`. The translation for this predicate is defined as follows:

```
"{} is a subclass of {}"
```

Inserting the translated arguments in this fragment yields the following:

```
"A is a subclass of feline"
```

The third layer

The left node in the next layer contains the predicate `subclass` and is therefore translated in the same manner as above.

The right node contains a *negation*, which means the translation of its child node is negated. To negate the sentence fragment `"A is a subclass of feline"`, we apply the pattern defined in Table 7.1 with the follwing result:

```
"A is not a subclass of feline"
```

The fourth layer

The fourth layer contains a new kind of node, namely the logical symbol `IMPLIES`. The translation of this symbol is defined as follows:

```
"if {left}, then {right}"
```

The specifiers `left` and `right` inside the braces refers to the translation of the left and right child node, respectively. And when both are inserted, the following translation is generated:

```
"if A is a subclass of canine, then A is not a subclass
 of feline"
```

The last layer

The final node contains the logical symbol `FORALL`. The translation this symbol is defined as follows:

```
"{right} for every {vars}"
```

This translation is special since `vars` refers to the combined translation of every variable bound by the quantifier represented by the node. In this example, the translation of all bound variables is simply `"A"`. But if there were more variables `X`, `Y` and `Z`, the combined translation would be `"X, Y and Z"`.

Furthermore, `right` refers to the subformula of the quantifier, which is always the last child node. With this, we can now create the final translation.

```
"If A is a subclass of canine, then A is not a subclass
 of feline for every A."
```

Since this was the final node, we also capitalized the first letter of the sentence and appended a period.

8

Evaluation

In this chapter, we evaluate our newly developed selection strategies and determine whether they provide any measurable improvements compared to common approaches. To achieve this, we need a large and complex knowledge base (KB) along with some benchmark goals. The symbol names in the KB should also have a meaning and be semantically similar to the concept they represent. Additionally, selection should be necessary to solve a significant number of the benchmark goals.

A KB that satisfies all these requirements is Adimen-SUMO [ÁLR12]. Adimen-SUMO provides a range of benchmark goals, including 8010 white-box truth tests, which we use as benchmark goals in our evaluation. We also need an automated theorem prover (ATP) to verify the effectiveness of our selection strategies or the lack thereof. Our ATP of choice is E [Sch02], a performant superposition-based theorem prover for first-order logic.

For the evaluation, we first randomly select 1000 white-box tests for which E was not able to find a proof within a 15 second time frame without selection. This time restriction is enforced by E's internal timer using the option `--soft-cpu-limit=15`. With these goals selected, we can then compare the effectiveness of different selection strategies with changing parameters.

This 15 second time limit was chosen to keep the scope of our evaluation manageable. It is based on experiments form [ÁHLR19], where the authors observed, that if the ATP was able to find a proof, most of the times this proof was found in up to 10 seconds.

Letting E try to find a proof for a goal can lead to different types of results. The desired result is *ProofFound*, which is returned if E was able to find a proof. This is the case if the goal is the logical consequence of the KB.

The next type of result is *TimeOut*. As the name suggests, this occurs when the E exceeds the time limit specified by the user. In our case, this means 15 seconds have elapsed and E was unable to find a proof. This result does not necessarily mean that there is no proof; it simply indicates that E could not find it within 15 seconds.

The last possible outcome is that E cannot find a proof with the provided axioms, regardless of the time limit. This is denoted by either *CounterSatisfiable* or *GaveUp*. If a result of type *CounterSatisfiable* is returned, at least one interpretation that satisfies the KB also satisfies the negation of the conjecture. The result *GaveUp* is a special case that only occurs when E uses a SInE-filter. It denotes that E has run out of unprocessed clauses and more information is needed to potentially find a proof. Since a filter was applied, E is aware that a selection occurred and there is more information.

In our experiments, *CounterSatisfiable* was only returned when the selection was done by SeVEn or SeVEn-union-SInE. This aligns with the fact that for both of these approaches, E was provided with a preselected set of axioms rather than having access to the entire KB and applying the filter itself, as in SInE.

8.1 Setting a Baseline

Before we can assess the effectiveness of SeVEn, we first have to establish a basis against which we can compare the results of our experiments. We accomplish this by testing the SInE selection strategy with varying parameters. The values for these parameters were chosen from the list of predefined SInE-filter provided by E.

Half of these built-in filters differ only in the parameter that determines whether formulas of type hypothesis are used as additional seeds for the analysis. Since there are no formulas of such type in our tests, we can disregard this parameter. Furthermore, we are only interested in two parameters, namely the benevolence b and the recursion depth k. So we only use filters where those parameters change and discard the rest. With the remaining variations, shown in Figure 8.1, we then let E run on the previously selected goals to find out which combination of parameters produces the best results.

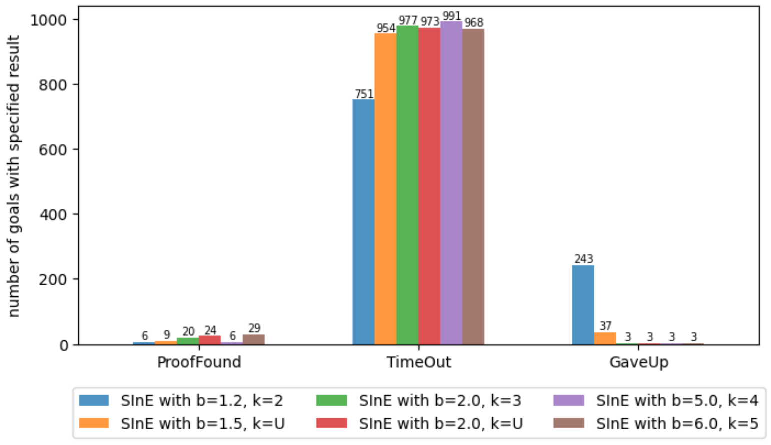

Fig. 8.1: Results of the evaluation of SInE on 1000 goals that E, without selection, could not solve. The different SInE configurations use different values for the benevolence parameter b and the recursion depth k.

The value `U` denotes an *unlimited value*, indicating that the corresponding parameter is set to the highest possible value and thus has no impact on the selection. Similarly, the parameters we are not interested in are assigned values that do not affect the selection. The parameter *generosity* is set to the largest possible value for a 64-bit signed integer, while the parameters *set size* and *set fraction* are assigned values of 20,000 and 1.0, respectively.

Figure 8.1 shows that the performance of SInE is greatly dependant on the chosen values for the parameters. With the best performing SInE configuration, we are able to find proofs for 20 to 29 problems.

8.2 Testing SeVEn

Knowing what is possible with established selection strategies, we can now proceed to evaluate SeVEn and determine if it can achieve comparable or even better results. Once again, we let the selection strategy we want to assess run with different configurations and compare the results. For SeVEn, this only involves one parameter, namely the number of selected axioms n.

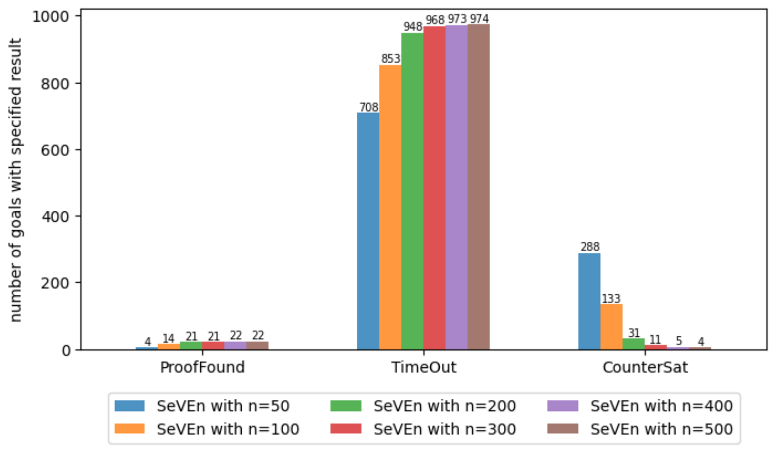

Fig. 8.2: Results of the evaluation of SeVEn on 1000 goals that E, without selection, could not solve. For each test a different value for the parameter n was used.

Figure 8.2 shows, that SeVEn, with the right parameter, produces promising results. Starting with $n = 200$, we find proofs for 21 goals. However, further increasing the number of selected axioms beyond this point yielded no significant improvement. Consequently, we stopped at n=500, with 22 proofs found.

Compared to the results of the previous experiment, SeVEn performs slightly worse. But since it is possible that these results slightly differ for other sets of goals, we still consider SInE and SeVEn similar in performance. However, due to the complex encoding of the axioms and the required preprocessing step of encoding the entire KB, we do not believe that this approach provides any benefit compared to SInE in most use cases.

8.3 Testing the Hybrid Approach: SeVEn-union-SInE

Next, we evaluate the hybrid selection strategy SeVEn-union-SInE, described in Definition 6.3. Since this approach allows us to change the parameters of SeVEn as well as those of SInE, we have many different possible configurations. To limit this number, we only use the three best performing SInE configurations taken from Figure 8.1 and combine them with a changing n-parameter for the SeVEn selection. The results of this are shown in Figure 8.3.

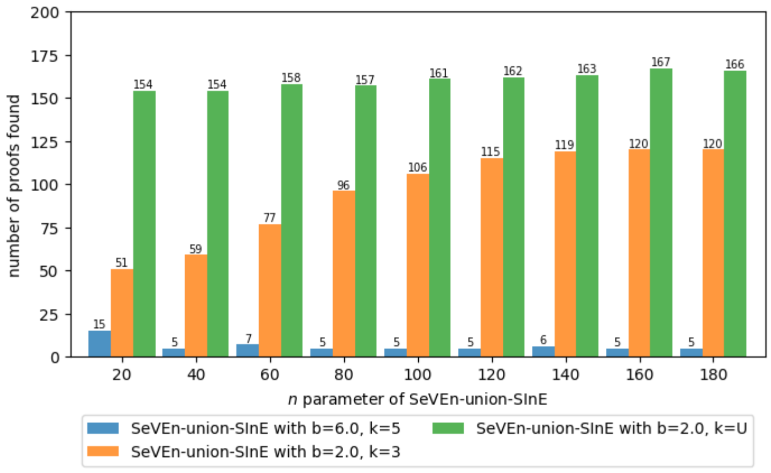

Fig. 8.3: Results of the evaluation of SeVEn-union-SInE on 1000 goals that E, without selection, could not solve. Each test uses a different SInE configurations with different values for the benevolence parameter b and the recursion depth k. A value of U denotes that this parameter is unlimited.

We observe that the hybrid approach constitutes a significant improvement over both its components. With the best configuration showing an improvement by a factor of at least six over the pure SInE selection with the corresponding parameters.

It is interesting to note that the approach with the best performing SInE configuration (SInE with b=6.0 and k=5) produced consistently the worst results when combined with SeVEn. The results are even worse than those of this SInE configuration alone.

A possible explanation for this is poor performance could be the number of selected axioms. This assumption is supported by the results depicted in Figure 8.4. We observe that significantly more axioms are selected on average when using this configuration. In contrast, the best performing configuration selects the fewest axioms on average.

Nonetheless, it seems that integrating statistical information into a syntactic selection strategy can yield better results than either selection approach alone, if properly configured. Comparable results were found in [Sch22], where the hybrid

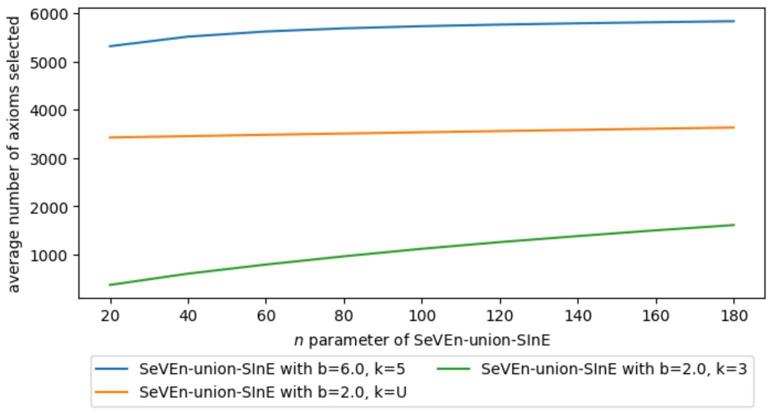

Fig. 8.4: Average number of axioms selected for different parameter n when using SeVEn-union-SInE with the three previously evaluated configurations.

approach Similarity SInE also outperformed SInE and the vector-based approach introduced in that paper.

8.4 Selecting with Thresholds

Lastly, we want to evaluate the selection with threshold (Definition 6.2). We split this evaluation into two parts: one for SeVEn and one for SeVEn-union-SInE. Starting with pure SeVEn, we proceed in the same manner as in section 8.2, but replace the parameter n with a threshold t.

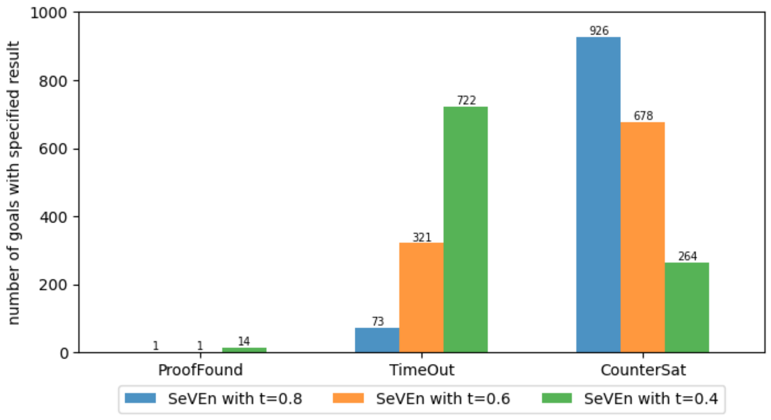

Fig. 8.5: Results of the evaluation of SeVEn with thresholds on 1000 goals that E, without selection, could not solve. For each test a different threshold t was used.

In Figure 8.5, we see that the SeVEn selection with thresholds does not produce satisfactory results. We assume that decreasing the threshold t even further would not improve the selection significantly. At some point, the threshold would no longer be restrictive, and the selection would no longer be truly selective because nearly all axioms would be chosen.

To understand this poor performance, we take a look at the number of axioms selected using this approach. When selecting with a threshold of 0.8, we find that on average only 1.404 axioms are selected. Table 8.1 shows similar results for the other threshold-values.

t	**max**	**min**	**mean**
0.8	23	0	1.404
0.6	200	0	16.024
0.4	2334	1	284.695

Table 8.1: Minimal, maximal and average number of axioms selected for different threshold values.

These results can also be visualised as a histogram, with the x-axis representing the number of axioms selected and the y-axis showing how frequently a particular number of axioms was selected. That further visualizes the problems when selecting with thresholds. Such a histogram is depicted in Figure 8.6 for every evaluated threshold-value.

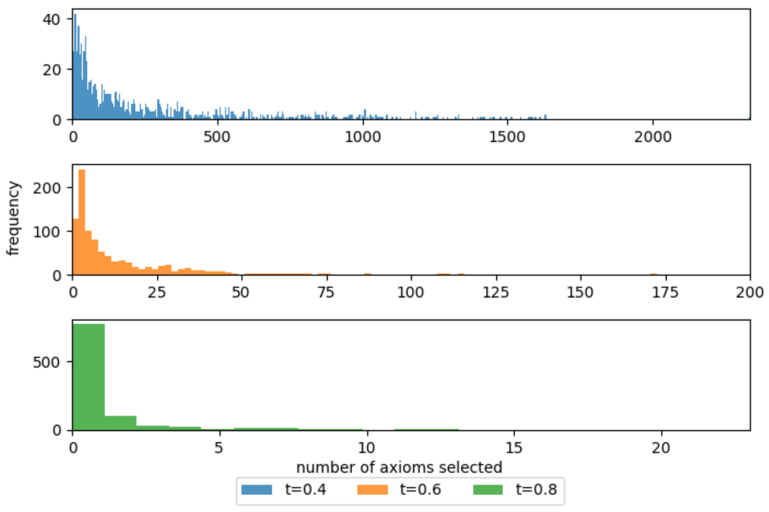

Fig. 8.6: Histograms depicting the number of axioms selected with SeVEn using different threshold values. Each sub-plot describes a different threshold. Note that the scaling of the axes varies amongst sub-plots.

These findings clearly indicate that the SeVEn selection with thresholds is very unstable and will often select too few axioms, if any at all. This renders it impossible for the ATP to find a proof for most of the goals.

Still, this approach may be useful when combined with another selection strategy, such as in SeVEn-union-SInE. This combination has the considerate advantage that even if SeVEn does not select any axioms, the SInE component of the hybrid approach will still be able to select some.

Again we follow the same steps as in Section 8.3 to evaluate the effectiveness of the hybrid approach when using a threshold.

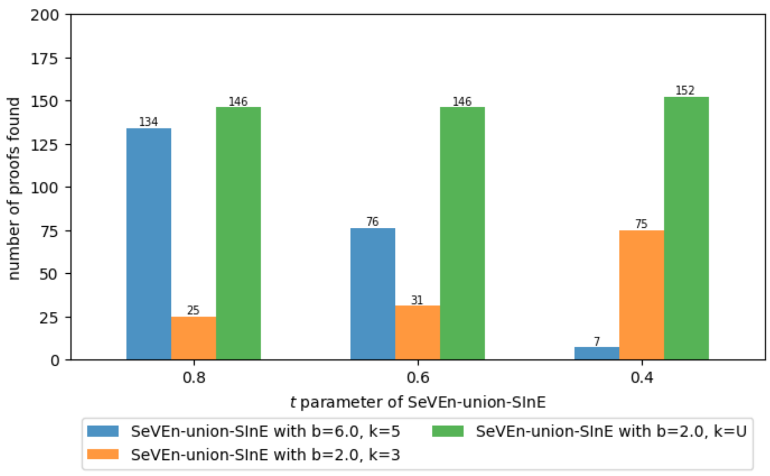

Fig. 8.7: Results of the evaluation of SeVEn-union-SInE with thresholds on 1000 goals that E, without selection, could not solve. Each test uses a different SInE configurations with different values for the benevolence parameter b and the recursion depth k. A value of `U` denotes that this parameter is unlimited.

Figure 8.7 shows that it is possible, with the right SInE configuration, to find proofs for 146 to 152 goals. We even achieve results comparable to the standard SeVEn-union-SInE, evaluated in Section 8.3, where we were able to find 154 to 166 proofs. Remarkably, both variations employ the same SInE configuration.

9 Conclusion and Future Work

In this thesis, we developed selection strategies based on a new vector representation of axioms. These approaches show great potential, but due to the limited scope of this work, we were not able to fully evaluate every aspect of them.

While SeVEn can produce competitive results, we still see two possible modifications that could potentially further increase its performance. Firstly, the translation process could be improved by either optimizing the current translator or changing the approach completely and using, for example, a language model that is trained to translate formulas into sentences. Furthermore, the language model used to encode the sentences could be modified or fine-tuned to better fit the use case.

The hybrid approach, SeVEn-union-SInE, seems even more promising, producing the best results out of all evaluated selection strategies. Since this approach is a combination of two different selection strategies, it also has a higher number of parameters that can be modified, which in turn means an even greater number of different configurations. However, we were unable to comprehensively test every viable combination of parameters, leaving open the possibility that some combinations may perform even better.

We also explored a variant of the vector-based approach that includes a threshold in the selection process. Unfortunately, this approach proved ineffective in our experiments and we do not believe that this idea alone warrants further research. But when combining it with other selection strategies, like in the hybrid approach SeVEn-union-SInE, we achieved far better results. This approach seems to counteract at least some of the drawbacks associated with using thresholds in the selection and we managed to produce results that are comparable to the hybrid approach without a threshold.

In future works, we want to focus on the hybrid approach, seeking to unlock its full potential by thoroughly testing different configurations, both with and without thresholds. In doing so, we hope to gain a deeper understanding of how individual parameters influence the overall performance. We also intend to conduct further research into the translation process of SeVEn and determine whether a more refined or authentic translation leads to an improvement in the selection process.

References

ÁHLR19. Álvez, Javier, Montserrat Hermo, Paqui Lucio and German Rigau: *Automatic white-box testing of first-order logic ontologies.* J. Log. Comput., 29(5):723–751, 2019.

ÁLR12. Álvez, Javier, Paqui Lucio and German Rigau: *Adimen-SUMO: Reengineering an Ontology for First-Order Reasoning.* Int. J. Semantic Web Inf. Syst., 8(4):80–116, 2012.

CYK^{+}18. Cer, Daniel, Yinfei Yang, Sheng-yi Kong, Nan Hua, Nicole Limtiaco, Rhomni St. John, Noah Constant, Mario Guajardo-Cespedes, Steve Yuan, Chris Tar, Yun-Hsuan Sung, Brian Strope and Ray Kurzweil: *Universal Sentence Encoder*, 2018.

DCLT19. Devlin, Jacob, Ming-Wei Chang, Kenton Lee and Kristina Toutanova: *BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.* In Burstein, Jill, Christy Doran and Thamar Solorio (editors): *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, pages 4171–4186. Association for Computational Linguistics, 2019.

dRBB^{+}16. Rooij, Steven de, Wouter Beek, Peter Bloem, Frank van Harmelen and Stefan Schlobach: *Are Names Meaningful? Quantifying Social Meaning on the Semantic Web.* In Groth, Paul, Elena Simperl, Alasdair J. G. Gray, Marta Sabou, Markus Krötzsch, Freddy Lécué, Fabian Flöck and Yolanda Gil (editors): *The Semantic Web - ISWC 2016 - 15th International Semantic Web Conference, Kobe, Japan, October 17-21, 2016, Proceedings, Part I*, volume 9981 of *Lecture Notes in Computer Science*, pages 184–199, 2016.

FKS19. Furbach, Ulrich, Teresa Krämer and Claudia Schon: *Names Are Not Just Sound and Smoke: Word Embeddings for Axiom Selection.* In Fontaine, Pascal (editor): *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil,* *August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, pages 250–268. Springer, 2019.

HR04. Huth, Michael and Mark Dermot Ryan: *Logic in computer science - modelling and reasoning about systems (2. ed.).* Cambridge University Press, 2004.

HV11. Hoder, Krystof and Andrei Voronkov: *Sine Qua Non for Large Theory Reasoning.* In Bjørner, Nikolaj S. and Viorica Sofronie-Stokkermans (editors): *Automated Deduction - CADE-23 - 23rd International Conference on Automated Deduction, Wroclaw, Poland, July 31 - August 5, 2011. Proceedings*, volume 6803 of *Lecture Notes in Computer Science*, pages 299–314. Springer, 2011.

KZS$^+$15. Kiros, Ryan, Yukun Zhu, Ruslan Salakhutdinov, Richard S. Zemel, Raquel Urtasun, Antonio Torralba and Sanja Fidler: *Skip-Thought Vectors.* In Cortes, Corinna, Neil D. Lawrence, Daniel D. Lee, Masashi Sugiyama and Roman Garnett (editors): *Advances in Neural Information Processing Systems 28: Annual Conference on Neural Information Processing Systems 2015, December 7-12, 2015, Montreal, Quebec, Canada*, pages 3294–3302, 2015.

LWWS20. Liu, Qinghua, Zishi Wu, Zihao Wang and Geoff Sutcliffe: *Evaluation of Axiom Selection Techniques.* In Fontaine, Pascal, Konstantin Korovin, Ilias S. Kotsireas, Philipp Rümmer and Sophie Tourret (editors): *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual)*, volume 2752 of *CEUR Workshop Proceedings*, pages 63–75. CEUR-WS.org, 2020.

MC91. Miller, George A. and Walter G. Charles: *Contextual correlates of semantic similarity.* Language and Cognitive Processes, 6(1):1–28, 1991.

MP09. Meng, Jia and Lawrence C. Paulson: *Lightweight relevance filtering for machine-generated resolution problems.* J. Appl. Log., 7(1):41–57, 2009.

MSC$^+$13. Mikolov, Tomás, Ilya Sutskever, Kai Chen, Gregory S. Corrado and Jeffrey Dean: *Distributed Representations of Words and Phrases and their Compositionality.* In Burges, Christopher J. C., Léon Bottou, Zoubin Ghahramani and Kilian Q. Weinberger (editors): *Advances in Neural Information Processing Systems 26: 27th Annual Conference on Neural Information Processing Systems 2013. Proceedings of a meeting held December 5-8, 2013, Lake Tahoe, Nevada, United States*, pages 3111–3119, 2013.

RG19. Reimers, Nils and Iryna Gurevych: *Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks.* In *Proceedings of the 2019*

Conference on Empirical Methods in Natural Language Processing. Association for Computational Linguistics, 11 2019.

RPS09. Roederer, Alex, Yury Puzis and Geoff Sutcliffe: *Divvy: An ATP Meta-system Based on Axiom Relevance Ordering*. In Schmidt, Renate A. (editor): *Automated Deduction - CADE-22, 22nd International Conference on Automated Deduction, Montreal, Canada, August 2-7, 2009. Proceedings*, volume 5663 of *Lecture Notes in Computer Science*, pages 157–162. Springer, 2009.

Sch02. Schulz, Stephan: *E - a brainiac theorem prover*. AI Commun., 15(2-3):111–126, 2002.

Sch22. Schon, Claudia: *Selection Strategies for Commonsense Knowledge*. CoRR, abs/2202.09163, 2022.

Sch23. Schon, Claudia: *Associative Reasoning for Commonsense Knowledge*. In Seipel, Dietmar and Alexander Steen (editors): *KI 2023: Advances in Artificial Intelligence - 46th German Conference on AI, Berlin, Germany, September 26-29, 2023, Proceedings*, volume 14236 of *Lecture Notes in Computer Science*, pages 170–183. Springer, 2023.

SN12. Schuster, Mike and Kaisuke Nakajima: *Japanese and Korean voice search*. In *2012 IEEE International Conference on Acoustics, Speech and Signal Processing, ICASSP 2012, Kyoto, Japan, March 25-30, 2012*, pages 5149–5152. IEEE, 2012.

SP07. Sutcliffe, Geoff and Yury Puzis: *SRASS - A Semantic Relevance Axiom Selection System*. In Pfenning, Frank (editor): *Automated Deduction - CADE-21, 21st International Conference on Automated Deduction, Bremen, Germany, July 17-20, 2007, Proceedings*, volume 4603 of *Lecture Notes in Computer Science*, pages 295–310. Springer, 2007.

Sut17. Sutcliffe, G.: *The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0*. Journal of Automated Reasoning, 59(4):483–502, 2017.

WSC$^+$16. Wu, Yonghui, Mike Schuster, Zhifeng Chen, Quoc V. Le, Mohammad Norouzi, Wolfgang Macherey, Maxim Krikun, Yuan Cao, Qin Gao, Klaus Macherey, Jeff Klingner, Apurva Shah, Melvin Johnson, Xiaobing Liu, Lukasz Kaiser, Stephan Gouws, Yoshikiyo Kato, Taku Kudo, Hideto Kazawa, Keith Stevens, George Kurian, Nishant Patil, Wei Wang, Cliff Young, Jason Smith, Jason Riesa, Alex Rudnick, Oriol Vinyals, Greg Corrado, Macduff Hughes and Jeffrey Dean: *Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation*. CoRR, abs/1609.08144, 2016.

A

Selbstständigkeitserklärung

☐ Diese Arbeit habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

☐ Diese Arbeit wurde als Gruppenarbeit angefertigt. Meinen Anteil habe ich selbstständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel verwendet.

Namen der Mitverfasser:

Meine eigene Leistung ist:

Datum	Unterschrift der Kandidatin/des Kandidaten