An Evaluation of Strategies to Train More Efficient Backward-Chaining Reasoners

Yue-Bo Jia
JiaY25@hsc.edu
Hampden-Sydney College
Hampden-Sydney, Virginia, USA

Gavin Johnson
gjohnson25@murraystate.edu
Murray State University
Murray, Kentucky, USA

Alex Arnold
alexarnold2024@u.northwestern.edu
Northwestern University
Evanston, Illinois, USA

Jeff Heflin
heflin@cse.lehigh.edu
Lehigh University
Bethlehem, Pennsylvania, USA

ABSTRACT

Knowledge bases traditionally require manual optimization to ensure reasonable performance when answering queries. We build on previous work on training a deep learning model to learn heuristics for answering queries by comparing different representations of the sentences contained in knowledge bases. We decompose the problem into issues of representation, training, and control and propose solutions for each subproblem. We evaluate different configurations on three synthetic knowledge bases. In particular we compare a novel representation approach based on learning to maximize similarity of logical atoms that unify and minimize similarity of atoms that do not unify, to two vectorization strategies taken from the automated theorem proving literature: a chain-based and a 3-term-walk strategy. We also evaluate the efficacy of pruning the search by ignoring rules with scores below a threshold.

KEYWORDS

backward chaining, efficient queries, knowledge bases, meta-reasoning, machine learning, neurosymbolic AI

ACM Reference Format:
Yue-Bo Jia, Gavin Johnson, Alex Arnold, and Jeff Heflin. 2023. An Evaluation of Strategies to Train More Efficient Backward-Chaining Reasoners. In *Knowledge Capture Conference 2023 (K-CAP '23), December 5–7, 2023, Pensacola, FL, USA.* ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/3587259.3627564

1 INTRODUCTION

Despite recent developments in machine learning—in particular, the advent of large language models—deep learning methods nevertheless continue to suffer from the perennial complaint that there is no explanation of *how* they know or accomplish their given task. At the same time, although symbolic methods can produce human-readable proofs, these methods nevertheless have difficulty with both performance and scale. The goal of neuro-symbolic AI [7] is to combine the strengths of these two approaches while avoiding the weaknesses of both.

This paper expands on the research of Arnold and Heflin to improve the performance of backward-chaining reasoning on knowledge bases using machine learning [1]. A knowledge base (KB) is a collection of facts and rules with which we make inferences; backward chaining is a traditional algorithm used to answer queries about what is true given the information stored in a KB. Backward-chaining, like inference procedures in general, is a type of search. In theory, machine learning can be used to train a model that can guide this search, thereby minimizing fruitless paths and backtracking.

Our contributions are to provide a framework for analyzing such problems, use this framework to propose solutions, and evaluate those solutions across three different synthetic KBs. In particular, we compare a unification-based approach for embedding logical atoms to vectorization strategies used for proof guidance in automated theorem proving (ATP). We also evaluate the use of a pruning strategy based on the predicted scores made by a trained neural model.

2 BACKGROUND

2.1 Neuro-Symbolic AI

As mentioned above, neuro-symbolic approaches attempt to combine the strengths of both machine learning and symbolic approaches. This approach is partly inspired by human cognition, which can both handle large amounts of input data and also use planning and reasoning to decipher and infer using that raw data [10]. Work such as Deep Reinforcement Learning with Symbolic Logics exemplifies the utility of this approach, in which the rigidity and transparency of logical reasoning is used in training a neural network to improve the ultimate safety and reliability of an autonomous driving system [8]. In essence, the idea is to use symbolic approaches to improve a machine learning system; previous work on using neuro-symbolic AI to conduct logical reasoning also uses this approach. For example, the approach used in "First Order Logical Neural Networks" attempts to train a neural network to directly perform reasoning in place of a traditional symbolic reasoner [5]; while NELLIE attempts to emulate an expert system using neural NLP models [12].

K-CAP '23, December 5–7, 2023, Pensacola, FL, USA

ACM ISBN 979-8-4007-0141-2/23/12…$15.00
https://doi.org/10.1145/3587259.3627564

Additionally, large language models (LLMs) allow machine learning models to traverse KBs consisting of facts, rules, and queries that are composed in the manner of human language. In doing so, we may implement both forward-chaining reasoners and backward-chaining reasoners within the context of our written statements in human-readable language(s) via LLMs. For instance, Google developed an LLM that implements backward-chaining in order to determine proofs of statements from initially given premises [4].

2.2 Proof Guidance

Unlike the neuro-symbolic approaches describe above, which attempt to train a neural network to perform like a symbolic reasoner, we aim to use machine learning to improve the performance and scalability of a symbolic reasoning system. Our work more closely resembles proof guidance in the automated theorem proving (ATP) literature. This approach retains two import aspects of symbolic AI: human readability and ensured safety. The "black box" nature of deep learning methods make it harder to provide guardrails for high impact/high risk decisions, but it is much easier to create human-readable decisions and safety measures in symbolic systems [9]. The worst case scenario in these kinds of frameworks is a less efficient reasoner, not one with dangerous high-risk mistakes.

There are several examples of proof guidance in ATP. Wang et al. [11] proposed to use Graph Convolutional Networks to identify which mathematical statements were relevant to a given conjecture. Jakubův and Urban's ENIGMA is a learning-based method to train a classification model to identify "*useful* and *un-useful*" clauses for proof search [3]. Crouse et al. combine various embedding strategies, including from ENIGMA, with a deep reinforcement learning approach to proof search [2]. Our work follows from a similar philosophy, but we note that query answering over KBs does have significant differences from ATP. In particular, KBs are usually described with less expressive logics, have much larger vocabularies of symbols, and have an abundance of facts compared to rules. Additionally, rather than just determining that a statement is entailed, we are usually interested in a set of variable bindings that constitute the answer to a query. Finally, KBs can be quite large. When associated with an ontology of axioms, knowledge graphs, can be seen as examples of large, real-world KBs containing billions of statements [6].

2.3 Horn Logic and Backward Chaining

In Horn logic, a subset of first-order logic, all propositions are either atoms—atomic propositions, or propositions without connectives—or clauses, which take the form of a conditional with exactly one atom as the consequent and the conjunction of arbitrarily many atoms as the antecedent; the consequent is called the *head*, whereas the antecedent is called the *body*. Following on Arnold and Heflin [1], we use Datalog's syntax,[1] in which the head is written on the left and ":-" is used for implication. Likewise, we will use uppercase letters to indicate variables and lowercase letters to indicate constants.

The backward-chaining reasoning algorithm is founded on the concept of unification. Two atoms are said to unify if one could apply some substitution of terms to the variables such that the two atoms would be made identical; for example, the propositions $p(X, Y)$ and $p(a, b)$ unify (with the substitution X/a and Y/b), but the propositions $p(a, X)$ and $p(b, X)$ do not, as different constants cannot be substituted for one another.

Given a query that consists of one or more subgoals —each an atom—we can find an answer by successively performing Selective Linear Definite (SLD) resolution with a subgoal in the query, and a rule in the KB whose head unifies with the subgoal. The answer can either be the query's truth value if the query has no variables, or a substitution for the query's variables that would make it true. The details of SLD resolution are not salient here; it suffices to know that when a rule matches a subgoal, a new goal is created by replacing the subgoal with the body of the rule, and the choice of a subgoal and rule changes the ultimate outcome of this algorithm,. Badly-chosen goal-rule pairs can result in failure to find an answer or infinite loops. In Prolog, subgoals are typically evaluated from left to right and rules are selected from top to bottom. With such a guarantee, KB designers are expected to optimize their logical statements. Such optimization can be difficult, however, as the range of possible queries increase, and could create a significant workload if two or more KBs must be integrated.

Arnold and Heflin's approach therefore seeks to learn heuristics for the choice of goal-rule pairs [1], training a model to classify such pairs according to whether they are likely to lead to an answer or not. They show that a guided reasoner using this strategy could often achieve improvements of orders of magnitude on most queries using a small, randomly-generated KB.

3 APPROACH

3.1 Overview

We have identified three main issues that must be addressed in order to train a more efficient reasoner: representation, training, and control. *Representation* is the determination of how to express symbolic information in a form that can be input to a neural architecture. Such architectures require inputs to be vectors of numbers. *Training* is the mechanism by which the reasoner learns knowledge to help guide decisions for future queries. For example, the system might be trained by supervised learning or reinforcement learning. *Control* determines how the knowledge is utilized. In particular, the control mechanism dictates which choices the algorithm will make, and when these choices will be made.

We now characterize Arnold and Heflin's approach to training a meta-reasoner [1],

(1) **Representation:** Use triplet loss to learn an embedding for logical atoms that respects unification.
(2) **Training:** Use supervised learning to train a system that can evaluate goal/rule pairs.
   (a) Use forward-chaining to get candidate queries in the KB for training.
   (b) Use backward-chaining to answer the queries. For each step, record the goal being attempted and rule applied. Goal/rule pairs that eventually lead to a solution are assigned a *positive* label, while all other pairs are assigned a *negative* label.

[1] This syntax is based on Prolog.

(c) Use supervised learning and a simple neural network to learn a prediction (between 0 and 1, inclusive) that an unseen goal/rule pair will lead to a solution.

(3) **Control:** Modify the standard backward-chaining algorithm so that at each step, the valid goal/rule pairs are sorted in descending order of their predictions. Rules are applied to subgoals in this order (instead of left-to-right, top-to-bottom).

This paper looks at alternatives for each of these steps. First, we consider two alternative representation mechanisms; these mechanisms were introduced for proof guidance in automated theorem proving. Second, we consider an alternative way to produce goal/rule scoring data that leads to more positive examples. Finally, we evaluate a change to the control policy that introduces a minimum threshold score in order for a goal/rule pair to be applied.

3.2 Representation

The problem of representation is a necessary condition for any solution to the overall problem; in order to apply machine learning, we must be able to generate appropriate vector representations, or embeddings, of logical propositions. Using ASCII, for example, would not do: since the letters a and b are represented by adjacent numbers, a neural network would then find these two letters to be closer to one another than, say, a and z, whereas in reality this relation is merely nominal.

3.2.1 Unification Embeddings. Arnold and Heflin [1] approach this problem as follows: An atomic proposition consists of a predicate and its arguments, each of which can be represented using a one-hot encoding. The atomic proposition can then be represented by concatenating the one-hot encoding for the predicate and each argument; because the vectors must all be the same size, representations of atomic propositions with fewer than a specified maximum number of arguments are padded with zeros. A neural network is then trained to create "embeddings that respect unification," maximizing the cosine similarity of embeddings of atoms that unify and minimizing the cosine similarity of embeddings of atoms that do not unify. The embedding for a rule is created by combining the embeddings of its constituent atoms: the embeddings of the body atoms are summed, and this is concatenated with the embedding of the head atom. This guarantees that every rule has an embedding of the same dimensions, regardless of how many antecedents it has.

It is not necessary to train a new embedding for a new KB. Instead, we can rely on the fact that one can apply a renaming to a KB without changing its semantics. Specifically, an embedding that has been trained on a vocabulary of a specific number of constants, variables, and predicates of varying arities can be reused to reason with any KB that does not exceed the counts of any of these symbol categories. For example, a vocabulary of a hundred variables, hundreds of predicates, and hundreds of thousands of constants should be suitable for many (though not all) KBs.

In their work, Arnold and Heflin show that this embedding strategy makes a significant difference in comparison to using an autoencoder to create embeddings from the one-hot vectors. Using the same set of training examples, the guided reasoner trained using autoencoder embeddings performs much worse than the guided reasoner trained using their approach. This result, however, naturally invites further comparison. Other work, particularly in the theorem-proving literature, has also attempted to develop methods for representing logical statements as vectors.

3.2.2 Alternative Vectorization Strategies. We identify two embedding strategies for comparison with the unification embedding strategy. Jakubův and Urban use an approach based on term walks of length three [3]. They begin by representing a logical proposition as a digraph in which edges exist between symbols adjacent in the sentence; for example, given the proposition $p(x, y)$, there would be an edge from p to x and p to y, although x and y would not themselves have an edge between them, and the parentheses are ignored. They then enumerate every 3-term-walk in such a graph; construct a vector of size $|\Sigma|^3$, where Σ is the set of symbols; and increment a determinate position in the vector by one for each occurrence of a given triple of symbols.

We made three additional adjustments to (1) maintain fidelity to the original application of this embedding strategy and (2) adapt this strategy to our own problem. Because this embedding approach varies with respect to the syntax used to express a proposition, we chose to translate sentences from Datalog syntax to a disjunction of clauses, which Jakubův and Urban uses in their work, and because this approach also treats both negation *and assertion* as symbols in their own right, we express implications as their equivalent disjunctive statements. Thus, if a KB were to contain a sentence such as $p(a, X)$:- $q(X, a)$, we would instead represent this sentence as $+(\neg q(X, a) \vee +p(a, X))$ before applying Jakubův and Urban's approach. Figure 1 below depicts the digraph that would be created from this proposition. Some examples of 3-term-walks are "$+ \vee \neg$" and "$\neg\ q\ a$".

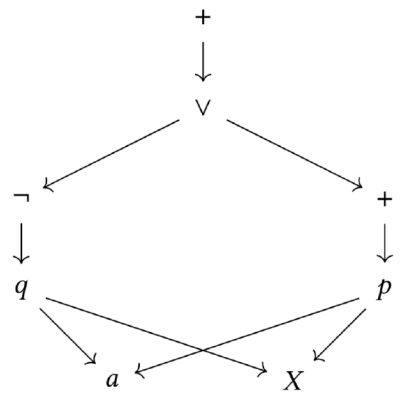

Figure 1: Digraph of the sentence $+(\neg q(X, a) \vee +p(a, X))$

Unfortunately, Jakubův and Urban's embedding approach results in embedding sizes that are cubic in the size of the vocabulary. The vector for an axiom has a position for each possible three-term walk, and the value at this position is the number of times that sequence occurs in the axiom. As mentioned above, for a vocabulary defined by the set Σ, the vector must have $|\Sigma|^3$ dimensions. Automated theorem proving often requires relatively few symbols, whereas capturing knowledge of even relatively constrained domains would require unwieldy symbol sets. Representing a college directory, for instance, would require at least one constant for each faculty member and student; for a tiny college of only a hundred total

people, this requirement would already result in an embedding size of over one million, without even considering predicates and variables. Thus, we reduce the size of the embedding by considering the constraints of Datalog. In particular, 3-term-walks of Datalog clauses expressed in the format we discuss above never begin with a predicate, constant, or variable, all of which together account for the majority of our symbol set; instead, it must always begin with either +, ∨, or ¬. Thus, instead of a vector of size $|\Sigma|^3$, we can instead construct a vector of size $3 * |\Sigma|^2$.

Crouse et al. proposed an alternative "chain-based" embedding approach [2]. Similarly to Jakubuv and Urban, they begin by representing a logical proposition as a graph, as we depict in Figure 1, albeit replacing every variable with the symbol '*'. Instead of enumerating term walks, they enumerate each "pattern," or path of symbols that begins at a predicate and bottoms out at a constant or variable. An example of a pattern from Figure 1 would be "$p*$." In order to create a d-dimensional representation, they then obtain the MD5 hash v of each pattern and set the element of the d-dimensional vector at the position $v \mod d$ to the number of occurrences of a pattern. Crouse et al. also distinguish between patterns and their negations by doubling the size of the representation; a separate d-dimensional vector is constructed to count the occurrences of pattern negations. Since our KB is expressed in Datalog which does not allow functions, all of our patterns contain one predicate and one constant or variable. Finally, because Crouse et al. do not specify a particular embedding size, for the sake of comparison we choose d such that the input size for the meta-reasoning neural network will be the same as with the unification embedding approach.

We identify two potential issues with the chain-based vectorization strategy: (1) since all variables are replaced with the same symbol, the representation does not distinguish between rules which can be applied in fewer situations due to having the same variable repeated, and rules which are more general; and (2) the hashing of patterns means that completely unrelated patterns could have the same representation.

3.3 Training

We have chosen to use a supervised learning approach to train our models. To be consistent with Arnold and Heflin [1], we used a simple neural model with one hidden layer. This hidden layer has 30 units and uses a sigmoid activation function. The output layer is a single unit with a sigmoid activation function. A Binary Cross-Entropy function (also known as Binary Log Loss) computes the loss for the model. In this paper, our main concern is how to produce suitable training data for this network.

Arnold and Heflin [1] proposed to generate positive and negative examples of goal/rule pairs. Given a set of training queries, they executed a backward-chaining search and classified examples based on the outcome of each path: if the path led to a solution, then for each node (except the root) in the path, they created a training example $(g, r, 1)$ for the subgoal g and rule r that led to the node. If the path failed to find a solution, then a training example $(g, r, 0)$ was created for each node. To ensure the search was not dependent on a particular order of rules or rule bodies, these were randomized with each choice. To ensure the search eventually terminated, a depth limit of 15 was set. All duplicates were removed and then oversampling of the smaller class and downsampling of the larger class were used to achieve a balanced set of examples.

Algorithm 1 BChainGuided($goals, \theta, depth$)

```
1:  if goals = ∅ then
2:    return θ
3:  else if depth > MaxDepth then
4:    return fail
5:  else
6:    opts ← {(g, r) | g ∈ goals and Unify(g, head(r))}
7:    Sort opts by descend score(g, r) for each (g, r) ∈ opts
8:    for all (g, r) ∈ opts do
9:      if score(g, r) < MinScore then
10:       return fail
11:     end if
12:     θ′ ←Unify(g, head(r))
13:     newg ← Subst(body(r) ∪ goals − {g}, θ′)
14:     θ′ ← Compose(θ, θ′)
15:     if depth < FallbackDepth then
16:       ans ← BChainGuided(newg, θ′, depth + 1)
17:     else
18:       ans ← BChainStd(newg, θ′, depth + 1)
19:     end if
20:     if ans ≠ fail then
21:       return ans
22:     end if
23:   end for
24: end if
25: return fail
```

We observed that the strategy above leads to sub-optimal scoring. In particular, a path that eventually fails may include several successful subgoals, but the strategy will assign all such goal/rule pairs a score of 0, since they were used on a path that was ultimately unsuccessful due to a completely unrelated goal. Thus we propose a new strategy for producing training examples: if subgoal g is eventually proven after applying rule r, then example $(g, r, 1)$ is created, otherwise, $(g, r, 0)$ is created. We note that identifying these examples while backward-chaining is non-trivial, and requires additional bookkeeping.

We note that for the random KBs we created a branching factor of 3 to 4 was typical, and a cutoff depth of 15 could lead to searches that took several minutes to complete. To support faster generation of training data we changed the depth limit to 5 after 50,000 nodes were expanded. Since the algorithm randomly selects which subgoal to prove and which rule to prove it, these cutoffs could often prevent the search from completing successfully. Therefore, when the search fails to find even one successful path, we restart up to two times.

3.4 Control

Our modified backward-chaining algorithm is shown in Algorithm 1. Like the standard algorithm, it performs a depth-first search by selecting a goal and a rule to achieve. In particular, it considers all subgoals (as opposed to just the first) and all rules with heads that unify with these subgoals. Each such (g, r) pair of goal g and

unifying rule r is then scored by the meta-reasoning model and the options are sorted by descending score (line 7). Thus, the highest scoring (g, r) is used to continue the search, and if backtracking is required, the next highest will be attempted. A new set of subogals is formed by removing g from the original goals and preprending the body of r. The algorithm is then called recursively. To avoid infinite search, we use a cutoff limit $MaxDepth$; in our experiments this was set to 10.

Our control strategy has additional points of interest. First, because our strategy can consider any current subgoal at any point in the tree, the potential branching factor is much larger than that of standard backward-chaining. Furthermore, if rules with more than one body atom are regularly applied, this branching factor can continue to grow at deeper levels. In the rare case where the model is unable to choose the right rules early on, the search space can expand rapidly. To limit this, we have instituted a $FallbackDepth$, such that the algorithm will switch over to a standard backward-chaining search once the depth is reached (line 15). In our experiments, this depth is set to 5.

We have also experimented with pruning (g, r) pairs that appear to be very unlikely to be useful. If $Score(g, r) < MinScore$ then the path is immediately terminated (line 9). Note, since we are processing (g, r) pairs in descending order of score, once we encounter one pair that is below the threshold, all remaining pairs will also be below the threshold. When we wish to compare this strategy to one without minimum rule scoring, we simply set the $MinScore$ threshold to 0; this will consider all matching (g, r) pairs.

4 EVALUATION

We conduct experiments to compare the different vectorization strategies and to compare the impact of minimum scoring on performance. We evaluate the different system configurations on three different sizes of randomly generated KBs: 100 statements, 150 statements and 200 statements. For each configuration, we run 100 queries that are distinct from the queries used to generate training data.

When generating random KBs, we use a probability distribution to determine the form of each statement: there is a 80% chance of a fact, and 20% chance of a rule. The body of a rule can have one to four atoms, where shorter bodies are more likely. We use the same randomly generated vocabulary for all three KBs: 10 predicates with arities from one to four, 10 variables, and 100 constants. The properties of our random KBs are given in Table 1. Since our training and test queries are extracted from the facts the can be derived from the KB, we report the number of inferred facts, as determined by a forward-chaining process to compute the deductive closure.

Statements	Non-inferred Facts	Inferred Facts
100	82	11
150	122	718
200	161	468

Table 1: Properties of the three KBs evaluated

We trained the unification embeddings using the vocabulary of the 150 statement KB and randomly generated atoms. Since we used the same vocabulary for the other two KBs, this embedding can be applied to the other KBs without having to map them into the vocabulary.

Name	**Embedding**	***MinScore***
Standard	n/a	n/a
Unification	Unification	0.00
3-term-walk	3-term-walk	0.00
Chain-based	Chain-based	0.00
Unification-Min-Score	Unification	0.01
Chain-based-Min-Score	Chain-based	0.01

Table 2: The five configurations evaluated

We generated training data and trained each meta-reasoner as described in Section 3.3. The number of goal/rule examples prior to balancing for each KB was 639, 5181, and 3622. For consistency, we used the same simple network architecture for all representation strategies: a two-layer network with sigmoid activation functions (see Section 3.3 for details). Each representation strategy was trained for 1000 epochs on each KB. We note that the term-walk strategy in particular takes an order of magnitude longer to train than the other strategies.[2] This is due to the larger number of parameters of the model: even with our small vocabulary of 10 predicates, 10 variables and 100 constants, a rule requires a vector with 45387 dimensions![3] In contrast, the unification and chain-based strategies can represent the same rule in 40 dimensions. The different models achieve different training losses: for example, with the 100 statement KB, the term-walk representation gets to a training loss of below 0.1 in 200 epochs, and eventually reaches below 0.05. We believe the significantly larger number of parameters is what allows it to achieve this, but we also suspect that the model is severely overfitting (more on that issue later). The chain-based and unification approaches are only able to achieve training losses of near 0.25. This suggests that improvements can be achieved by using network architectures with larger layers and/or more layers.

In order to test each configuration, we ran 100 test queries using the algorithm described in Section 3.4. We evaluated each representation with and without a $MinScore$ cutoff of 0.1, We used $MaxDepth = 10$ and $FallbackDepth = 5$ for all configurations. The standard reasoner is a backward-chaining reasoner that always evaluates subgoals from left-to-right and matches rules from top to bottom (i.e., the same evaluation order that Prolog uses). Of course, this configuration did not have a fallback depth.

Our first observation is that both chain-based and 3-term-walk failed on many queries when the $MinScore$ cutoff was used. When this occurred they often terminated search with much fewer nodes explored than otherwise, but given that they would fail to correctly answer a query, we excluded these configurations from further consideration. We hypothesize that these failures are the result of overfitting. The queries failed because useful paths received scores under the $MinScore$ value of 0.01, meaning the model was (mistakenly) very confident that they would not lead to solutions,

[2]It takes nearly 7 hours on a laptop with a 2.10 GHz Intel processor, an NVIDIA GeoForce MX550 GPU and 32GB of memory to train a 3-term-walk model on our simple network for 1000 epochs.

[3]The calculation accounts for three logical symbols, in addition to predicates, variables, and constants for a total vocabulary size of 123.

The fact In the case of 3-term-walk, this hypothesis is supported by the fact that the model has many more parameters and that the training loss reached diminishing returns far more quickly than the other representations.

In what follows, we generally compare five configurations: Standard, Unification, 3-term-walk, Chain-based, and Unification-Min-Score; we include the Chain-based-Min-Score configuration for the the 150 statement KB, as it did not fail on any answerable query. These configurations are summarized in Table 2.

An overall summary of the results is displayed in Figure 2. This chart plots the average number of nodes explored by each configuration on each KB. The error bars indicate standard deviations over all of the queries. Test queries that the standard reasoner failed to answer due to the maximum depth limit were excluded from these results; there are 95, 100, and 94 successful queries for the 100, 150, and 200 statement KBs, respectively. We additionally note that for the 200 statement KB, both the chain-based and the 3-term-walk approaches timed out on one query—although it was a different query in each case—that the unification and standard reasoner successfully answered. There are a few general observations from this graph: first all meta-reasoning approaches are able to do much better than the standard reasoner for the 100 statement KB. The larger KBs require much larger searches for all reasoners, and when just averages are considered, the meta-reasoners appear to perform worse than the standard reasoner. We also see large standard deviations, however, and when we look more closely at the data, a different picture emerges.

Figure 3 shows the median number of nodes explored for each configuration on each KB. Here it is clearly shown that for typical queries, all of the trained meta-reasoners are able to reduce the search by several orders of magnitude. This means that the large averages are due to outliers. For this reason, we now analyze the data by looking at detailed comparisons across each KB.

First, consider the 100 statement KB. Table 3 shows the mean nodes expanded, median nodes expanded, number of queries better than the standard reasoner and number of queries worse than the standard reasoner. All four meta-reasoning system explore on average ≈ 2.8 nodes, which is $4x$ fewer than the number of nodes explored by the standard reasoner. Chain-based and 3-term-walk performed slightly better than the unification strategy (both with and without minimum scoring): there were three queries where they searched four nodes instead of six. Each of the meta-reasoning system performed better than the baseline on 28 (out of 100) queries. None of them performed worse on any queries. We note that this KB only had 11 additional facts in its deductive closure, and as a result there could be significant similarities between the training and test queries. In such a case, all of the trained models would be able to become almost perfect reasoners, with little to no backtracking,

Now, consider the results of the more complex 150 statement KB as displayed in Table 4. This reinforces the observation from above that although the means of each the meta-reasoners was worse than the standard reasoner, the medians are significantly better. The standard reasoner had a median of $144,366$ while the other reasoners have medians of 4 or 5. This means that for most queries, the meta-reasoners were able to find solutions with very little search. However, because these systems consider many more possible paths, when the meta-reasoner makes incorrect predictions, the search can increase significantly, greatly increasing the mean. All of the meta-reasoners had at least one query that explored over 13 million nodes, while the standard reasoner's maximum was only $899,922$. We note that in terms of nodes explored, on average, without the min-score cutoff the unification approach performs best, followed by the 3-term-walk and then the chain-based method. With the min-score cutoff, however, the mean nodes explored are drastically reduced to significantly fewer than the standard reasoner; here, the chain-based approach fared better than the unification approach. When considering the number of queries better or worse than the standard reasoner, however, we observe a trade-off: although the Chain-based-Min-Score approach solves 66 queries faster compared to Unification-Min-Score's 62, the former also solves 17 queries slower in comparison to the latter's 10. These results are followed by the approaches that do not use the Min-Score cutoff: Unification with 50 queries better than the standard, 3-term-walk with 51, and and Chain-based with 45.

Finally, we look at the 200 statement KB, as displayed in Table 5, Due to the random nature of the KB, despite having more rules and facts than the 150 statement KB, it had few inferred facts (468 v. 718, see Table 1), Despite being a larger KB, queries can be answered more quickly. For example, the standard reasoner had a mean of $\approx 62,000$ nodes, while the mean was nearly $4x$ as many on the smaller 150 statement KB. We attribute this in part to random chance ordering the rules and facts in a fairly efficient way for the training queries that were selected. The most notable observation is that similar to the experiment for the 150 statement KB, all of the meta-reasoners had a significantly better number of median nodes explored than the standard: 4 vs. over 32,000. These results appear to be the least ambiguous: the unification approach appears decisively superior to the 3-term-walk and chain-based approach in terms of mean nodes explored, outperforming even the standard. It had only one query that performed worse than the standard, and with the Min-Score cutoff, the approach performed even better, with a five-fold decrease in the mean nodes explored and with *zero* queries that performed worse than the standard. By contrast, the 3-term-walk and chain-based results are more comparable to their performance on the 150 statement KB, with mean node counts in the millions and a more significant number of queries on which they performed worse than the standard.

In summary, the data shows that there are significant opportunities for meta-reasoners to make search for query answers more efficient, often reducing the number of nodes explored by several orders of magnitude. This effect is more likely with larger KBs, but there are factors other than the size of a KB that play a role as well. Unfortunately, there are occasional outliers that result in extremely poor performance, and these outliers can greatly inflate the average number of nodes explored. Future work is needed to more closely examine these outliers and design solutions that can eliminate or minimize them.

Of the approaches we evaluated, Unification-Min-Score is the most promising. It improves over standard on more queries than the other systems. 3-term-walk is worse than unification, but generally better than chain-based. However, training time is excessive and there is potential of overfitting. Furthermore, our goal is to eventually be able to answer queries over KBs with billions of statements and millions of constants. Since 3-term-walk produces vectors with

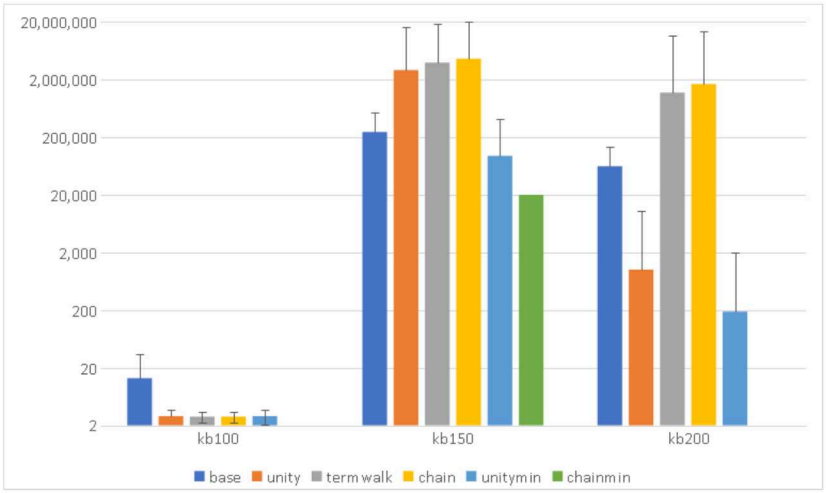

Figure 2: Mean nodes explored with standard deviation

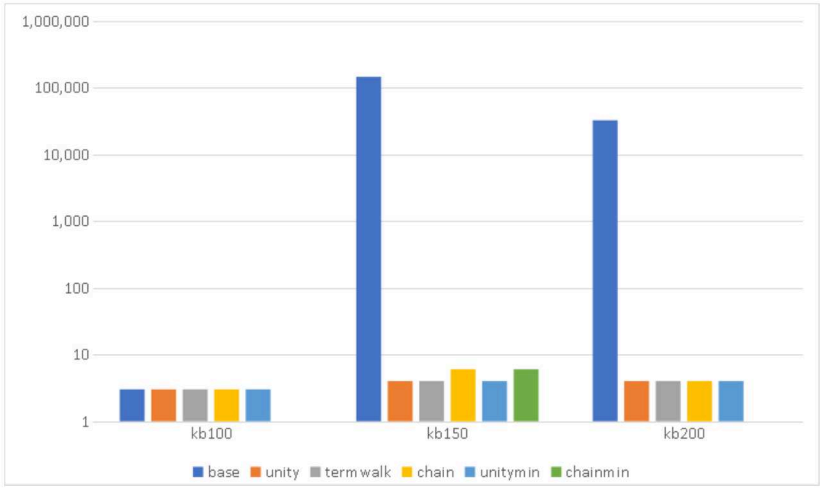

Figure 3: Median nodes explored

Embedding Method	Mean Nodes	Median Nodes	Queries Better	Queries Worse
Standard	13.1	3	N/A	N/A
Unification	2.9	3	28	0
3-term-walk	2.8	3	28	0
Chain-based	2.8	3	28	0
Unification-Min-Score	2.9	3	28	0

Table 3: Performance of the guided reasoner for the 100 rule KB

a dimensionality that is polynomial in the size of the vocabulary, it cannot scale to such purposes. It is a reasonable approach for automated theorem proving, where the vocabularies are small and the axioms are complex, but not for query-answering on large KBs. Chain-based is a reasonable alternative that seems to work well in both automated theorem proving and KB query answering, but is still out-performed by Unification embeddings when applied to Datalog KBs.

5 CONCLUSION

We have considered the problem of training reasoners to perform more effective search for query answers over medium-sized KBs.

Embedding Method	Mean Nodes	Median Nodes	Queries Better	Queries Worse
Standard	244,366.5	146368	N/A	N/A
Unification	2,895,708.2	4	50	19
3-term-walk	3,904,373.5	4	51	18
Chain-based	4,554,967.1	6	45	38
Unification-Min-Score	94,388.2	4	62	7
Chain-based-Min-Score	19,863.6	6	66	17

Table 4: Performance of the guided reasoner for the 150 rule KB

Embedding Method	Mean Nodes	Median Nodes	Queries Better	Queries Worse
Standard	62,235.8	32628.5	N/A	N/A
Unification	1,002.6	4	59	1
3-term-walk	1,177,214.1	4	57	15
Chain-based	1,658,454.2	4	58	6
Unification-Min-Score	187.1	4	60	0

Table 5: Performance of the guided reasoner for the 200 rule KB

There are three sub-problems: representation, training, and control. We have evaluated three different representation strategies and considered two variations of control. By evaluating the configurations on KBs of three different sizes, we have determined that the trained meta-reasoners are often able to outperform a baseline backward-chaining reasoner, but that there are sometimes a small number of outlier queries that are several orders of magnitude worse. Using a unification approach for vectorizing symbolic statements is promising, often showing improvements on more queries than the 3-term-walk and chain-based alternatives. We suggest that this difference comes from the nature of the backward-chaining algorithm, whose success or failure is determined by whether atoms can unify with one another. Furthermore, this appears to be the only vectorization approach that can consistently benefit from using a minimum score threshold to prune some paths without search. For the other representation strategies, it can incorrectly prune useful paths, leading to failed queries.

There are several avenues to continue this work. Given the variability across KBs, it is important to evaluate our approach on more KBs, and to look not just at random KBs, but also real-world KBs. Questions remain as to how best address the outliers: can these be reduced by adding more or larger layers to the neural network, or will changes to the control strategy be needed? One factor we mentioned is the search can have a much larger branching factor than that of a standard backward-chaining reasoner because instead of considering just one goal at any step, any goal can be considered. This suggests that an improvement might be possible by first evaluating and sorting the goals. Other important topics for future work include handling queries with multiple answers and transferring learning from one or more KBs to a new KB.

ACKNOWLEDGMENTS

This work was conducted as part of an REU site supported by the National Science Foundation under Grant No. CNS- 2051037.

REFERENCES

[1] Alex Arnold and Jeff Heflin. 2022. Learning a More Efficient Backward-Chaining Reasoner. In *Tenth Annual Conference on Advances in Cognitive Systems (ACS-2022)*. Cognitive Systems Foundation, Arlington, VA, 12 pages.

[2] Maxwell Crouse, Ibrahim Abdelaziz, Bassem Makni, Spencer Whitehead, Cristina Cornelio, Pavan Kapanipathi, Kavitha Srinivas, Veronika Thost, Michael Witbrock, and Achille Fokoue. 2021. A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving. *35th AAAI Conference on Artificial Intelligence, AAAI 2021* 7 (2021), 6279–6287. arXiv:1911.02065

[3] Jan Jakubův and Josef Urban. 2017. ENIGMA: Efficient Learning-Based Inference Guiding Machine. In *Intelligent Computer Mathematics*, Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke (Eds.). Springer International Publishing, Cham, 292–302.

[4] Mehran Kazemi, Najoung Kim, Deepti Bhatia, Xin Xu, and Deepak Ramachandran. 2023. LAMBADA: Backward Chaining for Automated Reasoning in Natural Language. arXiv:2212.13894 [cs.AI]

[5] Boonserm Kijsirikul and Thanupol Lerdlamnaochai. 2016. First-Order Logical Neural Networks. *International Journal of Hybrid Intelligent Systems* 2, 4 (2016), 253–267. https://doi.org/10.3233/his-2005-2403

[6] Natasha Noy, Yuqing Gao, Anshu N. Jain, Anantha Narayanan, Alan Patterson, and Jamie Taylor. 2019. Industry-scale knowledge graphs. *Commun. ACM* 62 (2019), 36 – 43. https://api.semanticscholar.org/CorpusID:153314008

[7] Md Kamruzzaman Sarker, Lu Zhou, Aaron Eberhart, and Pascal Hitzler. 2021. Neuro-Symbolic Artificial Intelligence: Current Trends. arXiv:2105.05330 [cs.AI]

[8] Iman Sharifi, Mustafa Yildirim, and Saber Fallah. 2023. Towards Safe Autonomous Driving Policies using a Neuro-Symbolic Deep Reinforcement Learning Approach. arXiv:2307.01316 [cs.RO]

[9] Amit Sheth, Manas Gaur, Kaushik Roy, Revathy Venkataraman, and Vedant Khandelwal. 2022. Process Knowledge-Infused AI: Toward User-Level Explainability, Interpretability, and Safety. *IEEE Internet Computing* 26, 5 (2022), 76–84. https://doi.org/10.1109/MIC.2022.3182349

[10] Amit Sheth, Kaushik Roy, and Manas Gaur. 2023. Neurosymbolic AI- Why, What, and How. *arXiv preprint arXiv:2305.00813* (2023).

[11] Mingzhe Wang, Yihe Tang, Jian Wang, and Jia Deng. 2017. Premise Selection for Theorem Proving by Deep Graph Embedding. In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (Long Beach, California, USA) *(NIPS'17)*. Curran Associates Inc., Red Hook, NY, USA, 2783–2793.

[12] Nathaniel Weir and Benjamin Van Durme. 2023. Dynamic Generation of Grounded Logical Explanations in a Neuro-Symbolic Expert System. arXiv:2209.07662 [cs.CL]

Received 4 September 2023