A Deep Reinforcement Learning Approach to First-Order Logic Theorem Proving

Maxwell Crouse[*1], **Ibrahim Abdelaziz**[*2], **Bassem Makni**[2], **Spencer Whitehead**[4], **Cristina Cornelio**[3], **Pavan Kapanipathi**[2], **Kavitha Srinivas**[2], **Veronika Thost**[5], **Michael Witbrock**[6], **Achille Fokoue**[2]

[1]Qualitative Reasoning Group, Northwestern University
[2]IBM Research, IBM T.J. Watson Research Center
[3]IBM Research, Zürich Research Center
[4]University of Illinois at Urbana-Champaign, Department of Computer Science
[5]MIT-IBM Watson AI Lab, IBM Research
[6]The University of Auckland, School of Computer Science

Abstract

Automated theorem provers have traditionally relied on manually tuned heuristics to guide how they perform proof search. Deep reinforcement learning has been proposed as a way to obviate the need for such heuristics, however, its deployment in automated theorem proving remains a challenge. In this paper we introduce TRAIL, a system that applies deep reinforcement learning to saturation-based theorem proving. TRAIL leverages (a) a novel neural representation of the state of a theorem prover and (b) a novel characterization of the inference selection process in terms of an attention-based action policy. We show through systematic analysis that these mechanisms allow TRAIL to significantly outperform previous reinforcement-learning-based theorem provers on two benchmark datasets for first-order logic automated theorem proving (proving around 15% more theorems).

1 Introduction

Automated theorem provers (ATPs) have established themselves as useful tools for solving problems that are expressible in a variety of knowledge representation formalisms. Such problems are commonplace in areas core to computer science (e.g., compilers (Curzon and Curzon 1991; Leroy 2009), operating systems (Klein 2009), and even distributed systems (Hawblitzel et al. 2015; Garland and Lynch 1998)), where ATPs are used to prove that a system satisfies some formal design specification. Unfortunately, while the formalisms that underlie such problems have been (more or less) fixed, the strategies needed to solve them have been anything but. With each new domain added to the purview of automated theorem proving, there has been a need for the development of new heuristics and strategies that restrict or order how an ATP searches for proofs. This process of guiding a theorem prover during proof search is referred to as *proof guidance*. Though proof guidance heuristics have been shown to have a drastic impact on theorem proving performance (Schulz and Möhrmann 2016), the specifics of when and why to use a particular strategy are still often hard to define (Schulz 2017).

Many state-of-the-art ATPs use machine learning to automatically determine heuristics for assisting with proof guidance. Generally, the features considered by such learned heuristics have been manually designed (Kaliszyk, Urban, and Vyskočil 2015; Jakubuv and Urban 2017), though more recently they have been learned through deep learning (Loos et al. 2017; Chvalovský et al. 2019; Paliwal et al. 2019), which has the appeal of lessening the amount of expert knowledge needed compared with handcrafting new heuristics. These neural approaches have just begun to yield impressive results, e.g. Enigma-NG (Jakubuv and Urban 2019) showed that purely neural proof guidance could be integrated into E (Schulz 2002) to improve its performance over manually designed proof-search strategies. However, in order to achieve competitive performance with state-of-the-art ATPs, neural methods (as they have been applied thus far) have critically relied on being seeded with proofs from an existing state-of-the-art reasoner (which itself will use a strong manually designed proof-search strategy). Thus, such approaches are still subject to the biases inherent to the theorem-proving strategies used in their initialization.

Reinforcement learning *a la* AlphaGo Zero (Silver et al. 2017b) has been explored as a natural solution to this problem, where the system automatically learns proof guidance strategies from scratch. More generally, reinforcement learning has been successfully applied to theorem proving with first-order logic (Kaliszyk et al. 2018; Piotrowski and Urban 2019; Zombori et al. 2019; Zombori, Urban, and Brown 2020), higher-order logic (Bansal et al. 2019a), and also with logics less expressive than first-order logic (Kusumoto, Yahata, and Sakai 2018; Lederman, Rabe, and Seshia 2018; Chen and Tian 2018).

In this work, we target theorem proving for first-order logic, where *tabula rasa* reinforcement learning (i.e., learning

[*]Denotes equal contribution, correspondence to Maxwell Crouse <mvcrouse@u.northwestern.edu>, Ibrahim Abdelaziz <ibrahim.abdelaziz1@ibm.com>, and Achille Fokoue <achille@us.ibm.com>

from scratch) has been integrated into tableau-based theorem provers (Kaliszyk et al. 2018; Zombori et al. 2019; Zombori, Urban, and Brown 2020). Connection tableau theorem proving is an appealing setting for machine learning research because tableau calculi are straightforward and simple, allowing for concise implementations that can easily be extended with learning-based techniques. However, the best performing, most widely-used theorem provers to date are saturation theorem provers that implement either the resolution or superposition calculi (Kovács and Voronkov 2013; Schulz 2002). These provers are capable of much finer-grained management of the proof search space; however, this added power comes at the cost of increased complexity in terms of both the underlying calculi and the theorem provers themselves. For neural-based proof guidance to yield any improvements when integrated with highly optimized, hand-tuned saturation-based provers, it must offset the added cost of neural network evaluation with more intelligent proof search. To date, this has not been possible when these neural approaches have been trained from scratch, i.e. when they are *not* bootstrapped with proofs from a state-of-the-art ATP.

In this paper, we introduce TRAIL (Trial Reasoner for AI that Learns), a theorem proving approach that applies deep reinforcement learning to saturation-based theorem proving to learn proof guidance strategies completely from scratch. Key to TRAIL's design is a novel neural representation of the state of a theorem-prover in terms of inferences and clauses, and a novel characterization of the inference selection process in terms of an attention-based action policy. The neural representations for clauses and actions that constitute TRAIL's internal state were based on a careful study of candidate representations, which we describe in this paper.

We demonstrate the performance of TRAIL on two standard benchmarks drawn from the Mizar dataset (Kaliszyk and Urban 2015): M2k (Kaliszyk et al. 2018) and MPTP2078 (Urban 2006), where we show that TRAIL, when trained from scratch, outperforms all prior reinforcement-learning approaches on these two datasets and approaches the performance of a state-of-the-art ATP on MPTP2078 dataset.

2 Background

We assume the reader has knowledge of basic first-order logic and automated theorem proving terminology and thus will only briefly describe the terms commonly seen throughout this paper. For readers interested in learning more about logical formalisms and techniques see (Bergmann, Moor, and Nelson 2013; Enderton and Enderton 2001).

In this work, we focus on first-order logic (FOL) with equality. In the standard FOL problem-solving setting, an ATP is given a *conjecture* (i.e., a formula to be proved true or false), *axioms* (i.e., formulas known to be true), and *inference rules* (i.e., rules that, based on given true formulas, allow for the derivation of new true formulas). From these inputs, the ATP performs a *proof search*, which can be characterized as the successive application of inference rules to axioms and derived formulas until a sequence of derived formulas is found that represents a *proof* of the given conjecture. All formulas considered by TRAIL are in *conjunctive normal form*. That is, they are conjunctions of *clauses*, which are themselves disjunctions of literals. Literals are (possibly negated) formulas that otherwise have no inner logical connectives. In addition, all variables are implicitly universally quantified.

Let F be a set of formulas and $\mathcal{I}$ be a set of inference rules. We write that F is *saturated* with respect to $\mathcal{I}$ if every inference that can be made using axioms from $\mathcal{I}$ and premises from F is also a member of F, i.e. F is closed under inferences from $\mathcal{I}$. Saturation-based theorem provers aim to saturate a set of formulas with respect to their inference rules. To do this, they maintain two sets of clauses, referred to as the *processed* and *unprocessed* sets of clauses. These two sets correspond to the clauses that have and have not been yet selected for inference. The actions that saturation-based theorem provers take are referred to as *inferences*. Inferences involve an inference rule (e.g. resolution, factoring) and a non-empty set of clauses, considered to be the *premises* of the rule. At each step in proof search, the ATP selects an inference with premises in the unprocessed set (some premises may be part of the processed set) and executes it. This generates a new set of clauses, each of which is added to the unprocessed set. The clauses in the premises that are members of the unprocessed set are then added to the processed set. This iteration continues until a clause is generated (typically the empty clause for refutation theorem proving) that signals a proof has been found, the set of clauses is saturated, or a timeout is reached. For more details on saturation (Robinson 1965) and saturation-calculi, we refer the reader to (Bachmair and Ganzinger 1998).

3 TRAIL

We first describe our overall approach to defining the proof guidance problem in terms of reinforcement learning. For this, we detail (a) a *sparse vectorization process* which represents all clauses and actions in a way compatible with subsequent neural components, (b) a *neural proof state* which concisely captures the neural representations of clauses and actions within a proof state, and (c) an *attention-based policy network* that learns the interactions between clauses and actions to select the next action. Last, we describe how TRAIL learns from scratch, beginning with a random initial policy.

3.1 Reinforcement Learning for Proof Guidance

We formalize the proof guidance as a reinforcement learning (RL) problem where the reasoner provides the environment in which the learning agent operates. Figure 1 shows how an ATP problem is solved in our framework. Given a conjecture and a set of axioms, TRAIL iteratively performs reasoning steps until a proof is found (within a provided time limit). The reasoner tracks the proof state, s_t, which encapsulates the clauses that have been derived or used in the derivation so far and the actions that can be taken by the reasoner at the current step. At each step, this state is passed to the learning agent - an attention-based model (Luong, Pham, and Manning 2015) that predicts a distribution over the actions it uses to sample a corresponding action, $a_{t,i}$. This action is given to the reasoner, which executes it and updates the proof state.

Formally, a state, $s_t = (\mathcal{C}_t, \mathcal{A}_t)$, consists of two components. The first is the set of processed clauses, $\mathcal{C}_t = \{c_{t,j}\}_{j=1}^{N}$,

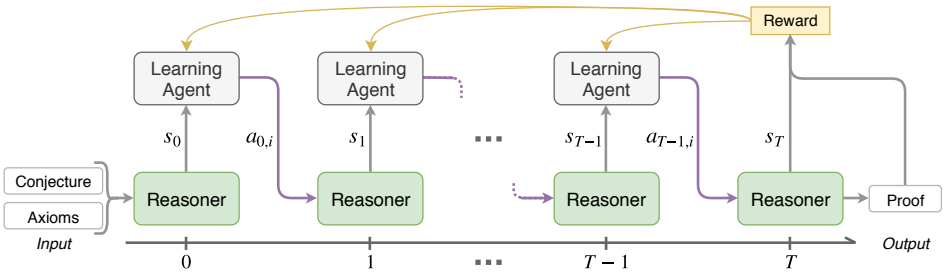

Figure 1: Formulation of automated theorem proving as a RL problem

(i.e., all clauses selected by the agent up to step t); where $\mathcal{C}_0 = \emptyset$. The second is the set of all available actions that the reasoner could execute at step t, $\mathcal{A}_t = \{a_{t,i}\}_{i=1}^{M}$; where $\mathcal{A}_0$ is the cross product of the set of all inference rules (denoted by $\mathcal{I}$) and the set of all axioms and the negated conjecture. An action, $a_{t,i} = (z_{t,i}, \hat{c}_{t,i})$, is a pair comprising an inference rule, $z_{t,i}$, and a clause from the unprocessed set, $\hat{c}_{t,i}$.

At step t, given a state s_t (provided by the reasoner), the learning agent computes a probability distribution over the set of available actions $\mathcal{A}_t$, denoted by $P_\theta(a_{t,i}|s_t)$ (where θ is the set of parameters for the learning agent), and samples an action $a_{t,i} \in \mathcal{A}_t$. The sampled action $a_{t,i} = (z_{t,i}, \hat{c}_{t,i})$ is executed by the reasoner by applying $z_{t,i}$ to $\hat{c}_{t,i}$ (which may involve processed clauses). This yields a set of new derived clauses, $\bar{\mathcal{C}}_t$, and a new state, $s_{t+1} = (\mathcal{C}_{t+1}, \mathcal{A}_{t+1})$, where $\mathcal{C}_{t+1} = \mathcal{C}_t \cup \{\hat{c}_{t,i}\}$ and $\mathcal{A}_{t+1} = (\mathcal{A}_t - \{a_{t,i}\}) \cup (\mathcal{I} \times \bar{\mathcal{C}}_t)$.

Upon completion of a proof attempt, TRAIL computes a loss and issues a reward that encourages the agent to optimize for decisions leading to a successful proof in the shortest time possible. Specifically, for an unsuccessful proof attempt (i.e., the underlying reasoner fails to derive a contradiction within the time limit), each step t in the attempt is assigned a reward $r_t = 0$. For a successful proof attempt, in the final step, the underlying reasoner produces a refutation proof $\mathcal{P}$ containing only the actions that generated derived facts directly or indirectly involved in the final contradiction. At step t of a successful proof attempt where the action $a_{t,i}$ is selected, the reward r_t is 0 if $a_{t,i}$ is not part of the refutation proof $\mathcal{P}$; otherwise r_t is inversely proportional to the time spent proving the conjecture.

The final loss consists of the standard policy gradient loss (Sutton and Barto 1998) and an entropy regularization term to avoid collapse onto a sub-optimal deterministic policy and to promote exploration.

$$\mathcal{L}(\theta) = -\mathbb{E}\big[r_t \log(P_\theta(a_t|s_t))\big] - \lambda\mathbb{E}\big[\sum_{i=1}^{|\mathcal{A}_t|} -P_\theta(a_{t,i}|s_t)\log(P_\theta(a_{t,i}|s_t))\big]$$

where a_t is the action taken at step t and λ is the entropy regularization hyper-parameter. We use a normalized reward to improve training stability, since the intrinsic difficulty of problems can vary widely in our problem dataset. We explored (i) normalization by the inverse of the time spent by a traditional reasoner, (ii) normalization by the best reward obtained in repeated attempts to solve the same problem, and (iii) no normalization; the normalization strategy was a hyper-parameter. This loss has the effect of giving actions that contributed to the most direct proofs for a given problem higher rewards, while dampening actions that contributed to more time consuming proofs for the same problem.

3.2 Sparse Vectorization Process

For compatibility with subsequent neural components, TRAIL transforms the formulas internal to the proof state into real-valued vectors. To do this, TRAIL utilizes a set of M vectorization modules, $\mathcal{M} = \{m_1, \ldots, m_M\}$, that each characterize some important aspect of the clauses and actions under consideration.

Each module $m_k \in \mathcal{M}$ follows the same general design: given an input clause or action, m_k produces a discrete, bag-of-words style vector in $\mathbb{Z}^{n_k}$, where n_k is a pre-specified dimensionality specific to module m_k. As an example, consider a module intended to capture the symbols of a formula. It would map each symbol present in its input to an index ranging from $1, \ldots, n_k$. The vector representation would be created by assigning to each dimension of a sparse n_k-dimensional vector the number of times a symbol corresponding to that dimension appeared in the input (i.e., a bag-of-words representation). Letting $m_k(i)$ be the sparse vector for an input i from module $m_k \in \mathcal{M}$, the final vector representation v_i is then the concatenation of the outputs from each module. As actions are pairs of clauses and inference rules, TRAIL represents the clause in each action pair using the process described above and the inference rule as a one-hot encoding of size $|\mathcal{I}|$, where $\mathcal{I}$ is the set of inference rules.

This vectorization process allows TRAIL to trivially incorporate techniques from the large body of research on vectorization strategies for machine learning guided theorem proving (Bridge, Holden, and Paulson 2014; Kaliszyk, Urban, and Vyskočil 2015). For instance, TRAIL uses Enigma (Jakubuv and Urban 2017) modules which characterize a clause in terms of fixed-length term walks (with separate modules for term walks of length $l \in \{1, 2, 3\}$).

In addition to term walks and chain-based vectorization (described next), TRAIL uses general purpose modules for representing clause age (the timestep at which a clause was derived), weight (the number of symbols in the clause), literal count, and set-of-support (whether or not the clause has a negated conjecture clause as an ancestor in its proof tree).

Chain-Based Vectorization We refer to the vectorization method introduced here as *chain-based vectorization*. Within TRAIL, it functions as one of the available vectorization modules. Its development was inspired by the way inference rules operate over clauses. Consider the resolution inference rule. Two clauses resolve if one contains a positive literal whose constituent atom is unifiable with the constituent atom of a negative literal in the other clause. Hence, vector representations of clauses should both capture the relationship between literals and their negations and reflect structural similarities between literals that are indicative of unifiability.

Our approach captures these features by deconstructing input clauses into sets of patterns. We define a *pattern* to be a linear chain that begins from a predicate symbol and includes one argument (and its argument position) at each depth until it ends at a constant or variable. The set of all patterns for a clause is then the set of all linear paths between each predicate and the constants and variables they bottom out with. Since the names of variables are arbitrary, they are replaced with a wild-card symbol, "$*$". Argument position is also indicated with the use of wild-card symbols.

We obtain a d-dimensional representation of a clause by hashing the linearization of each pattern p using MD5 hashes (Rivest 1992) to compute a hash value v, and setting the element at index $v \bmod d$ to the number of occurrences of the pattern p in the clause. We also explicitly encode the difference between patterns and their negations by doubling the representation size and hashing them separately, where the first d elements encode the positive patterns and the second d elements encode the negated patterns. Chain vectorization is intended to produce feature vectors useful for estimates of structural similarity. Figure 2 shows how chain patterns would be extracted from a clause along with term walks.

The use of hashing (provided that d is small enough) is partially intended to discourage the learning of symbol or pattern-specific interactions. That is, if hash collisions are sufficiently frequent, we would expect them to act as a form of regularization that inhibits the network from relying on the presence or absence of specific patterns within a clause. Ideally (though certainly not guaranteed), the network would then instead learn to rely on more holistic features like the overall structural similarity between pairs of clauses (given by the dot product of pattern feature vectors).

An important design decision for TRAIL was to rely on

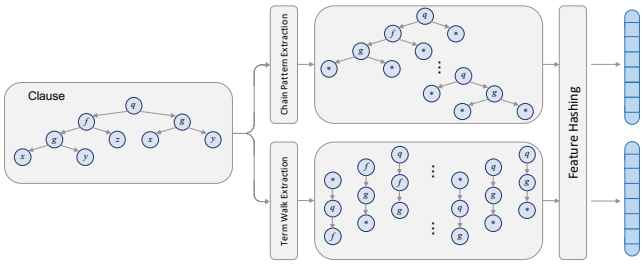

Figure 2: Overview of chain and term walk vectorizers operating on the clause $q(f(g(x,y),z),g(x,y))$

a combination of simple and efficient vectorizers like the pattern-based vectorizers described above, as opposed to more complex graph neural network (GNN) approaches. Though GNN-based approaches are powerful, they introduce a noticeable additional run-time overhead (i.e., more time spent looking for a better search strategy and less time left to execute it). We contrast our approach with a graph neural network based approach in Section 4.

3.3 Neural Representation of Proof State

Recall that the proof state consists of the sets of processed clauses $\mathcal{C}_t$ and actions $\mathcal{A}_t$. We write $\mathbf{c}_{t,i}$ as the sparse vector representation for processed clause c_i at time t and $(\mathbf{z}_{t,i}, \hat{\mathbf{c}}_{t,i})$ as the vector representations for the inference rule and clause pairing for action $a_{t,i}$. To produce dense representations for the elements of $\mathcal{C}_t$ and $\mathcal{A}_t$, TRAIL first transforms the sparse clause representations (excluding inference rules for actions) into dense representations by passing them through k fully-connected layers. This yields sets $\{\mathbf{h}^{(p)}_{t,1}, ..., \mathbf{h}^{(p)}_{t,N}\}$ and $\{\mathbf{h}^{(a)}_{t,1}, ..., \mathbf{h}^{(a)}_{t,M}\}$ of dense representations for the processed and action clauses. TRAIL also collects the dense representations for the negated conjecture clauses as $\{\mathbf{h}^{(c)}_1, \dots \mathbf{h}^{(c)}_k\}$.

For each action pair, TRAIL concatenates the clause representation $\mathbf{h}^{(a)}_{t,i}$ with the corresponding inference representation $\mathbf{z}_{t,i}$ to form the new action $\mathbf{a}_{t,i} = [\mathbf{h}^{(a)}_{t,i}, \mathbf{z}_{t,i}]$ and joins each such action into a matrix $\mathbf{A}$. To construct the processed clause matrix, TRAIL first produces a dense representation of the conjecture as the element-wise mean of dense negated conjecture clause representations

$$\mathbf{h}^{(c)} = \frac{1}{k}\sum_{i=1}^{k} \mathbf{h}^{(c)}_i$$

where k is the number of conjecture clauses. New processed clause representations are produced by combining the original dense representations with the pooled conjecture. For a processed clause embedding $\mathbf{h}^{(p)}_i$, its new value would be

$$\hat{\mathbf{h}}^{(p)}_i = \mathbf{h}^{(p)}_i + \mathbf{h}^{(c)} + F(\mathbf{h}^{(p)}_i \,||\, \mathbf{h}^{(c)})$$

where F is a feed-forward network, $||$ denotes the concatenation operation, and the original inputs are included with skip connections (He et al. 2016). The new processed clause embeddings are then joined into a matrix $\mathbf{C}$.

The two resulting matrices $\mathbf{C}$ and $\mathbf{A}$ can be considered the neural forms of $\mathcal{C}_t$ and $\mathcal{A}_t$. Thus, they concisely capture the notion of a *neural proof state*, where each column of either matrix corresponds to an element from the formal proof state. Following the construction of $\mathbf{C}$ and $\mathbf{A}$, this neural proof state is fed into the policy network to select the next inference.

3.4 Attention-Based Policy Network

Figure 3 shows how sparse representations are transformed into the neural proof state and passed to the policy network. Throughout the reasoning process, the policy network must produce a distribution over the actions relative to the clauses that have been selected up to the current step, where both the

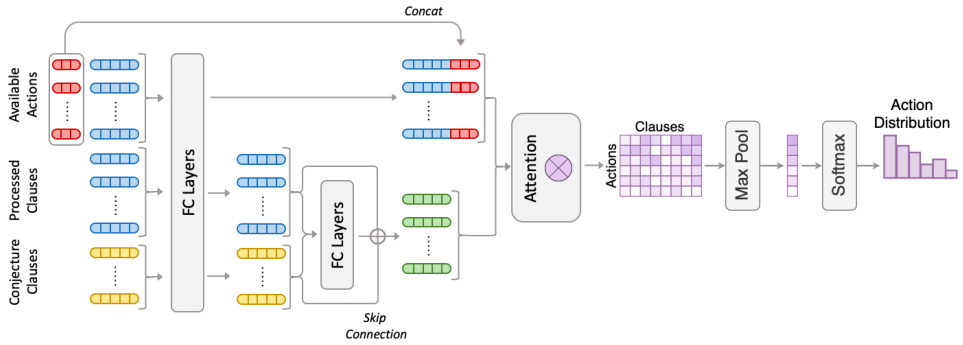

Figure 3: Flow from sparse vectorization through the policy network

actions and clauses are sets of variable length. This setting is analogous to ones seen in attention-based approaches to problems like machine translation (Luong, Pham, and Manning 2015; Vaswani et al. 2017) and video captioning (Yu et al. 2016; Whitehead et al. 2018), in which the model must score each encoder state with respect to a decoder state or other encoder states. To score each action relative to each clause, we compute a multiplicative attention (Luong, Pham, and Manning 2015) as $\mathbf{H} = \mathbf{A}^\top \mathbf{W}_a \mathbf{C}$, where $\mathbf{W}_a \in \mathbb{R}^{(2d+|\mathcal{I}|)\times 2d}$ is a learned parameter and the resulting matrix, $\mathbf{H} \in \mathbb{R}^{M\times N}$, is a heat map of interaction scores between processed clauses and available actions. TRAIL then performs max pooling over the columns (i.e., clauses) of $\mathbf{H}$ to produce unnormalized action values $\widehat{P}_\theta(a_{t,i}|s_t)$

Prior work integrating deep learning with saturation-based ATPs would use a neural network to score the unprocessed clauses with respect to *only* the conjecture and *not* the processed clauses (Loos et al. 2017; Jakubuv and Urban 2019). TRAIL's attention mechanism can be viewed as a natural generalization of this, where inference selection takes into account both the processed clauses and conjecture.

3.5 Learning From Scratch

TRAIL begins learning through random exploration of the search space as done in AlphaZero (Silver et al. 2017a) to establish performance when the system is started from a *tabula rasa* state (i.e., a randomly initialized policy network P_θ). At training, at an early step t (i.e., $t < \tau_0$, where τ_0, the temperature threshold, is a hyper-parameter that indicates the depth in the reasoning process at which training exploration stops), we sample the action $a_{t,i}$ in the set of available actions $\mathcal{A}_t$, according to the following probability distribution P_θ derived from the policy network's output $\widehat{P}_\theta$:

$$P_\theta(a_{t,i}|s_t) = \frac{\widehat{P}_\theta(a_{t,i}|s_t)^{1/\tau}}{\sum_{a_{t,j}\in\mathcal{A}_t} \widehat{P}_\theta(a_{t,j}|s_t)^{1/\tau}}$$

where τ, the temperature, is a hyperparameter that controls the exploration-exploitation trade-off and decays over the iterations (a higher temperature promotes more exploration). When the number of steps already performed is above the temperature threshold (i.e., $t \geq \tau_0$), an action, $a_{t,i}$, with the highest probability from the policy network, is selected, i.e.

$$a_{t,i} = \arg\max_{a_{t,j}\in\mathcal{A}_t} P_\theta(a_{t,j}|s_t) \ .$$

At the end of training iteration k, the newly collected examples and those collected in the previous w iterations (w is the example buffer hyperparameter) are used to train, in a supervised manner, the policy network using the reward structure and loss function defined in Section 3.1.

4 Experiments and Results

In this section, we are trying to answer the following questions: (a) Is TRAIL effective at generalized proof guidance? (b) What is the best vectorization strategy and how well does it generalize / transfer to unseen problems?

Datasets We evaluated TRAIL on two datasets: *M2k* (Kaliszyk et al. 2018) and *MPTP2078* (Alama et al. 2014a). Both datasets are exports of parts of Mizar[1] (Grabowski, Kornilowicz, and Naumowicz 2010) into the TPTP (Sutcliffe 2017) format. The M2k dataset contains 2003 problems selected randomly from the subset of Mizar that is known to be provable by existing ATPs, while MPTP2078 contains 2078 problems selected regardless of whether or not they could be solved by an ATP system.

4.1 Effectiveness of TRAIL

For traditional ATP systems, we compare TRAIL to: 1) E (Schulz 2002) in auto mode, a state-of-the-art saturation-based ATP system that has been under development for over two decades, 2) Beagle (Baumgartner, Bax, and Waldmann 2015), a newer saturation-based theorem prover that has achieved promising results in recent ATP competitions, and 3) mlCop (Kaliszyk, Urban, and Vyskočil 2015), an OCaml reimplementation of leanCop (Otten and Bibel 2003), which is a tableau-based theorem prover that was applied to M2k in (Kaliszyk et al. 2018) and MPTP2078 in (Zombori, Urban, and Brown 2020). For learning-based approaches, we compare against two recent RL-based approaches: rlCop (Kaliszyk et al. 2018) and plCop (Zombori, Urban, and

[1] https://github.com/JUrban/deepmath/

		M2k	MPTP2078
Traditional	E	**1922**	**998**
	Beagle	1543	742
	mlCop	1034	502
Reinforcement Learning-Based	rlCop	1235	733
	plCop	1359	782
	TRAIL	**1561**	**910**

Table 1: Number of problems solved in M2k and MPTP2078. Best two approaches in **bold**

Brown 2020), both of which are connection tableau-based theorem provers that build off mlCop and leanCop, respectively. All results for mlCop, rlCop, plCop, and E are those reported in (Kaliszyk et al. 2018; Zombori, Urban, and Brown 2020), except for E on M2k which we ran ourselves. We also replicated plCop and rlCop numbers under our exact hardware and time constraints and found comparable results.

Following (Kaliszyk et al. 2018; Zombori, Urban, and Brown 2020), we report the *best iteration completion performance*. In this setting, TRAIL starts from scratch and is applied to M2k and MPTP2078 for 10 iterations, with learning from solved problems occurring after each iteration completes. The reported number of problems solved is then the number solved in the best performing iteration.

Table 1 shows the performance of TRAIL against both traditional ATP systems and recent learning-based approaches. Following (Zombori, Urban, and Brown 2020), we limit TRAIL to a maximum of 2,000 steps per problem with a hard limit of 100 seconds. As Table 1 shows, while TRAIL outperformed Beagle, state-of-the-art saturation-based ATP systems are still superior to RL-based systems, with E solving the most problems on both datasets. Among learning-based approaches, TRAIL solved 202 more problems on M2k compared to plCop (Zombori, Urban, and Brown 2020), and on MPTP2078, TRAIL solved 128 more problems. On both datasets TRAIL outperformed its underlying reasoner, Beagle, with a 22% relative improvement on MPTP2078. These results show that TRAIL (when trained from a random initial policy) is a competitive theorem proving approach, as it outperformed all other systems except for E, with significant gains over prior RL-based approaches. Notably, TRAIL substantially narrows the gap between state-of-the-art traditional and from-scratch RL approaches on MPTP2078.

Value of Attention Over Processed Clauses As refuting the conjecture is the end goal of proof search, incorporating the negated conjecture into action selection has clear value. However, we would expect processed clauses to be useful as well, since most inferences will involve premises from both the unprocessed and processed clauses. To determine the utility of including processed clauses in action selection, we performed an ablation experiment that restricted TRAIL to utilizing *only* the conjecture's neural representation in the processed clause matrix (i.e., the embedded representation for each processed clause became $\hat{\mathbf{h}}_i^{(p)} = \mathbf{h}^{(c)}$). This turned the attention mechanism into a measure of how aligned an action was to solely the conjecture representation.

When using only the conjecture representations, TRAIL solved 1,561 problems on M2k and 854 problems on MPTP. Recall that TRAIL when incorporating processed clause and conjecture representations solved 1,561 problems on M2k and 910 problems on MPTP. Though there was no difference for M2k, it is significant that TRAIL could solve 56 more problems ($\sim$6% relative improvement) on MPTP when incorporating processed clauses.

4.2 Effects of Vectorization Choice

The term walk and chain-based vectorization modules used by TRAIL are intended to quickly extract structural features from their inputs. Though they are not tailored to any particular domain within FOL, they are clearly feature engineering. Graph neural network (GNN) approaches have been proposed as a general alternative that can lessen the need for expert knowledge by having a neural network automatically learn salient features from the underlying graph representations of logical formulas. Though GNN approaches have quickly gained traction in this domain due to the graph-centric nature of automated reasoning (Paliwal et al. 2019; Wang et al. 2017; Olšák, Kaliszyk, and Urban 2019), they have not yet clearly demonstrated that the value they provide in terms of formula representation offsets their greater computational cost when deployed in a comparison with a state-of-the-art FOL ATP.

We view TRAIL as providing a framework for further testing this, as GNN-based vectorizers can be trivially substituted into TRAIL's operation as one of its initial vectorization modules (see Section 3.2) that feed into the computation of the neural proof state. Consequently, we performed an experiment where we varied the vectorization modules available to TRAIL, comparing four different ablated versions: 1) TRAIL using the chain-based vectorizer, 2) TRAIL using the term walk vectorizer (Jakubuv and Urban 2017), 3) TRAIL using a GCN-based vectorizer (Kipf and Welling 2016), and 4) combinations of these vectorizers. For all ablations, TRAIL had access to the general-purpose vocabulary-independent features mentioned in Section 3.2. The GCN is the standard GCN described in (Kipf and Welling 2016) operating over the shared subexpression forms of clauses (Wang et al. 2017).

We report TRAIL's performance using both the prior *best iteration performance* metric (where the train and test sets are the same) and a more standard evaluation where the train and test sets are distinct. For the latter evaluation, we use one dataset (i.e., M2k or MPTP) to train and the other to test (see next Section). Table 2 shows the performance of TRAIL for these two metrics on both datasets, with the distinction between the sparse vectorization modules, the GNN-based module, and all combinations of each explicitly indicated.

Performance Within Datasets Examining Table 2, we see that while chain-based vectorization is weaker than term walk vectorization, the combination of the two achieves significantly better results as a whole. This suggests that they are capturing non-redundant feature sets. Additionally, both the term-walk vectorization and the GCN are very strong on M2k, but they do not do well in every case (i.e., there seems to be something important they are not capturing). Notably, while the GCN alone provides quite reasonable performance, com-

		No Transfer		Transfer	
		M2k	MPTP	MPTP → M2k	M2k → MPTP
Sparse Only	Chains	1473	653	1463	418
	Term Walks	1552	737	1425	570
	Chains + Term Walks	**1561**	**910**	**1510**	**812**
Neural Only	GCN	1523	750	1394	712
Sparse & Neural	GCN + Chains	1495	734	1395	652
	GCN + Term Walks	1549	660	1394	611
	GCN + Chains + Term Walks	1555	706	1435	690

Table 2: TRAIL's performance with different vectorization methods in terms of number of proved theorems. Best results in **bold**

bining it with other vectorization modules leads to a decrease in performance; which we suspect is due to the more expensive computational cost. However, that the GCN provides a strong baseline is promising, as it required no domain-expert knowledge. Our takeaway is that cheap vectorization methods are currently still quite useful; however, the gap between those techniques and GNN-based approaches is closing.

Generalization Across Datasets Table 2 also shows the performance of TRAIL when trained on M2k and tested on MPTP dataset, and vice versa. In terms of transfer, the combination of chain-based and term walk vectorization produces the best results. This provides evidence that these features are reasonably domain-general (though, we again emphasize that, while M2k and MPTP are different datasets and may require different reasoning strategies, they are both generated from Mizar). Notably, there was only a 11% loss of performance when training on MPTP and testing M2K, and a 3% loss when training on M2k and testing MPTP2078. From this experiment we conclude that cheap structural features, though hand-engineered, are relatively domain-general, which again will make them tough for more complex approaches to beat.

5 Related Work

Several approaches have focused on the sub-problem of premise selection (i.e., finding the axioms relevant to proving the considered problem) (Alama et al. 2014b; Blanchette et al. 2016; Alemi et al. 2016; Wang et al. 2017). As is often the case with automated theorem proving, most early approaches were based on manual heuristics (Hoder and Voronkov 2011; Roederer, Puzis, and Sutcliffe 2009) and traditional machine learning (Alama et al. 2014b); though some recent works have used neural models (Alemi et al. 2016; Wang et al. 2017; Rawson and Reger 2020; Crouse et al. 2019; Piotrowski and Urban 2020). Additional research has used learning to support interactive theorem proving (Blanchette et al. 2016; Bancerek et al. 2018).

Some early research applied (deep) RL for guiding inference (Taylor et al. 2007), planning, and machine learning techniques for inference in relational domains (van Otterlo 2005). Several papers have considered propositional logic or other decidable FOL fragments, which are much less expressive compared to TRAIL. Closer to TRAIL are the approaches described in (Kaliszyk et al. 2018; Zombori, Urban, and Brown 2020) where RL is combined with Monte-Carlo tree search for theorem proving in FOL. However they have some limitations: 1) Their approaches are specific to tableau-based reasoners and thus not suitable for theories with many equality axioms, which are better handled in the superposition calculus (Bachmair, Ganzinger, and Waldmann 1994), and 2) They rely upon simple linear learners and gradient boosting as policy and value predictors.

Our work also aligns well with the recent proposal of an API for deep-RL-based interactive theorem proving in HOL Light, using imitation learning from human proofs (Bansal et al. 2019b). That paper also describes an ATP as a proof-of-concept. However, their ATP is intended as a baseline and lacks more advanced features like our exploratory learning.

Non-RL-based approaches using deep-learning to guide proof search include (Chvalovský et al. 2019; Loos et al. 2017; Paliwal et al. 2019). These approaches differ from ours in that they seed the training of their networks with proofs from an existing reasoner. In addition, they use neural networks during proof guidance to score and select available clauses with respect *only* to the conjecture. Recent works have focused on addressing these two strategies. For instance, (Piotrowski and Urban 2019) explored incorporating more than just the conjecture when selecting which inference to make with an RNN-based encoding scheme for embedding entire proof branches in a tableau-based reasoner. However, it is unclear how to extend this method to saturation-based theorem provers, where a proof state may include thousands of irrelevant clauses. Additionally, (Aygün et al. 2020) investigated whether synthetic theorems could be used to bootstrap a neural reasoner without relying on existing proofs. Though their evaluation showed promising results, it was limited to a subset of the TPTP (Sutcliffe 2017) that excluded equality. It is well known that the equality predicate requires much more elaborate inference systems than resolution (Bachmair and Ganzinger 1998), thus it is uncertain as to whether their approach would be extensible to full equational reasoning.

6 Conclusions

In this work we introduced TRAIL, a system using deep reinforcement learning to learn effective proof guidance in a saturation-based theorem prover. TRAIL outperformed all prior RL-based approaches on two standard benchmark datasets and approached the performance of a state-of-the-art traditional theorem prover on one of the two benchmarks.

References

Alama, J.; Heskes, T.; Kühlwein, D.; Tsivtsivadze, E.; and Urban, J. 2014a. Premise selection for mathematics by corpus analysis and kernel methods. *Journal of Automated Reasoning* 52(2): 191–213.

Alama, J.; Heskes, T.; Kühlwein, D.; Tsivtsivadze, E.; and Urban, J. 2014b. Premise Selection for Mathematics by Corpus Analysis and Kernel Methods. *J. Autom. Reasoning* 52(2): 191–213. doi:10.1007/s10817-013-9286-5. URL https://doi.org/10.1007/s10817-013-9286-5.

Alemi, A. A.; Chollet, F.; Een, N.; Irving, G.; Szegedy, C.; and Urban, J. 2016. DeepMath - Deep Sequence Models for Premise Selection. In *Proceedings of the 30th International Conference on Neural Information Processing Systems*, NIPS'16. URL http://dl.acm.org/citation.cfm?id=3157096.3157347.

Aygün, E.; Ahmed, Z.; Anand, A.; Firoiu, V.; Glorot, X.; Orseau, L.; Precup, D.; and Mourad, S. 2020. Learning to Prove from Synthetic Theorems. *arXiv preprint arXiv:2006.11259* .

Bachmair, L.; and Ganzinger, H. 1998. Equational reasoning in saturation-based theorem proving. *Automated deduction—a basis for applications* 1: 353–397.

Bachmair, L.; Ganzinger, H.; and Waldmann, U. 1994. Refutational theorem proving for hierarchic first-order theories. *Applicable Algebra in Engineering, Communication and Computing* 5(3-4).

Bancerek, G.; Byliński, C.; Grabowski, A.; Korniłowicz, A.; Matuszewski, R.; Naumowicz, A.; and Pąk, K. 2018. The Role of the Mizar Mathematical Library for Interactive Proof Development in Mizar. *Journal of Automated Reasoning* 61(1): 9–32. ISSN 1573-0670. doi:10.1007/s10817-017-9440-6. URL https://doi.org/10.1007/s10817-017-9440-6.

Bansal, K.; Loos, S. M.; Rabe, M. N.; and Szegedy, C. 2019a. Learning to Reason in Large Theories without Imitation. *CoRR* abs/1905.10501. URL http://arxiv.org/abs/1905.10501.

Bansal, K.; Loos, S. M.; Rabe, M. N.; Szegedy, C.; and Wilcox, S. 2019b. HOList: An Environment for Machine Learning of Higher-Order Theorem Proving (extended version). *CoRR* abs/1904.03241. URL http://arxiv.org/abs/1904.03241.

Baumgartner, P.; Bax, J.; and Waldmann, U. 2015. Beagle – A Hierarchic Superposition Theorem Prover. In Felty, A. P.; and Middeldorp, A., eds., *CADE-25 – 25th International Conference on Automated Deduction*, volume 9195 of *LNAI*, 367–377.

Bergmann, M.; Moor, J.; and Nelson, J. 2013. *The Logic Book*. McGraw-Hill Higher Education. ISBN 9780077578336. URL https://books.google.com/books?id=-SYiAAAAQBAJ.

Blanchette, J. C.; Greenaway, D.; Kaliszyk, C.; Kühlwein, D.; and Urban, J. 2016. A Learning-Based Fact Selector for Isabelle/HOL. *J. Autom. Reason.* 57(3): 219–244. ISSN 0168-7433. doi:10.1007/s10817-016-9362-8. URL http://dx.doi.org/10.1007/s10817-016-9362-8.

Bridge, J. P.; Holden, S. B.; and Paulson, L. C. 2014. Machine learning for first-order theorem proving. *Journal of automated reasoning* 53(2): 141–172.

Chen, X.; and Tian, Y. 2018. Learning to Progressively Plan. *CoRR* abs/1810.00337. URL http://arxiv.org/abs/1810.00337.

Chvalovský, K.; Jakubuv, J.; Suda, M.; and Urban, J. 2019. ENIGMA-NG: Efficient Neural and Gradient-Boosted Inference Guidance for E. *CoRR* abs/1903.03182. URL http://arxiv.org/abs/1903.03182.

Crouse, M.; Abdelaziz, I.; Cornelio, C.; Thost, V.; Wu, L.; Forbus, K.; and Fokoue, A. 2019. Improving graph neural network representations of logical formulae with subgraph pooling. *arXiv preprint arXiv:1911.06904* .

Curzon, P.; and Curzon, P. 1991. A Verified Compiler for a Structured Assembly Language. In *TPHOLs*, 253–262.

Enderton, H.; and Enderton, H. 2001. *A Mathematical Introduction to Logic*. Elsevier Science. ISBN 9780080496467. URL https://books.google.com/books?id=dVncCl_EtUkC.

Garland, S. J.; and Lynch, N. A. 1998. The IOA language and toolset: Support for designing, analyzing, and building distributed systems. Technical report, Technical Report MIT/LCS/TR-762, Laboratory for Computer Science.

Grabowski, A.; Kornilowicz, A.; and Naumowicz, A. 2010. Mizar in a nutshell. *Journal of Formalized Reasoning* 3(2): 153–245.

Hawblitzel, C.; Howell, J.; Kapritsos, M.; Lorch, J. R.; Parno, B.; Roberts, M. L.; Setty, S.; and Zill, B. 2015. IronFleet: proving practical distributed systems correct. In *Proceedings of the 25th Symposium on Operating Systems Principles*, 1–17.

He, K.; Zhang, X.; Ren, S.; and Sun, J. 2016. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, 770–778.

Hoder, K.; and Voronkov, A. 2011. Sine qua non for large theory reasoning. In *International Conference on Automated Deduction*, 299–314. Springer.

Jakubuv, J.; and Urban, J. 2017. ENIGMA: Efficient Learning-Based Inference Guiding Machine. In *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Proceedings*, 292–302. doi:10.1007/978-3-319-62075-6_20. URL https://doi.org/10.1007/978-3-319-62075-6_20.

Jakubuv, J.; and Urban, J. 2019. Hammering Mizar by Learning Clause Guidance. *CoRR* abs/1904.01677. URL http://arxiv.org/abs/1904.01677.

Kaliszyk, C.; and Urban, J. 2015. MizAR 40 for Mizar 40. *J. Autom. Reasoning* 55(3): 245–256. doi:10.1007/s10817-015-9330-8. URL https://doi.org/10.1007/s10817-015-9330-8.

Kaliszyk, C.; Urban, J.; Michalewski, H.; and Olsák, M. 2018. Reinforcement Learning of Theorem Proving. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018*, 8836–8847.

Kaliszyk, C.; Urban, J.; and Vyskočil, J. 2015. Certified connection tableaux proofs for HOL Light and TPTP. In *Proceedings of the 2015 Conference on Certified Programs and Proofs*, 59–66.

Kaliszyk, C.; Urban, J.; and Vyskočil, J. 2015. Efficient Semantic Features for Automated Reasoning over Large Theories. In *Proceedings of the 24th International Conference on Artificial Intelligence*, IJCAI'15, 3084–3090. AAAI Press. ISBN 978-1-57735-738-4. URL http://dl.acm.org/citation.cfm?id=2832581.2832679.

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907* .

Klein, G. 2009. Operating system verification—an overview. *Sadhana* 34(1): 27–69.

Kovács, L.; and Voronkov, A. 2013. First-order theorem proving and Vampire. In *International Conference on Computer Aided Verification*, 1–35. Springer.

Kusumoto, M.; Yahata, K.; and Sakai, M. 2018. Automated Theorem Proving in Intuitionistic Propositional Logic by Deep Reinforcement Learning. *CoRR* abs/1811.00796. URL http://arxiv.org/abs/1811.00796.

Lederman, G.; Rabe, M. N.; and Seshia, S. A. 2018. Learning Heuristics for Automated Reasoning through Deep Reinforcement Learning. *CoRR* abs/1807.08058. URL http://arxiv.org/abs/1807.08058.

Leroy, X. 2009. Formal verification of a realistic compiler. *Communications of the ACM* 52(7): 107–115.

Loos, S. M.; Irving, G.; Szegedy, C.; and Kaliszyk, C. 2017. Deep Network Guided Proof Search. In *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, 85–105. URL http://www.easychair.org/publications/paper/340345.

Luong, T.; Pham, H.; and Manning, C. D. 2015. Effective Approaches to Attention-based Neural Machine Translation. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*, 1412–1421. Lisbon, Portugal: Association for Computational Linguistics. doi:10.18653/v1/D15-1166. URL https://www.aclweb.org/anthology/D15-1166.

Olšák, M.; Kaliszyk, C.; and Urban, J. 2019. Property Invariant Embedding for Automated Reasoning. In *24th European Conference on Artificial Intelligence*.

Otten, J.; and Bibel, W. 2003. leanCoP: lean connection-based theorem proving. *Journal of Symbolic Computation* 36(1-2).

Paliwal, A.; Loos, S.; Rabe, M.; Bansal, K.; and Szegedy, C. 2019. Graph Representations for Higher-Order Logic and Theorem Proving. *arXiv preprint arXiv:1905.10006* .

Piotrowski, B.; and Urban, J. 2019. Guiding Theorem Proving by Recurrent Neural Networks. *arXiv preprint arXiv:1905.07961* .

Piotrowski, B.; and Urban, J. 2020. Stateful Premise Selection by Recurrent Neural Networks. In *LPAR 2020: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Alicante, Spain, May 22-27, 2020*, 409–422. URL https://easychair.org/publications/paper/g38n.

Rawson, M.; and Reger, G. 2020. Directed Graph Networks for Logical Reasoning. In *Workshop on Practical Aspects of Automated Reasoning*.

Rivest, R. 1992. The MD5 message-digest algorithm. Technical report.

Robinson, J. A. 1965. A machine-oriented logic based on the resolution principle. *Journal of the ACM (JACM)* 12(1): 23–41.

Roederer, A.; Puzis, Y.; and Sutcliffe, G. 2009. Divvy: An ATP meta-system based on axiom relevance ordering. In *International Conference on Automated Deduction*, 157–162. Springer.

Schulz, S. 2002. E–a brainiac theorem prover. *AI Communications* 15(2, 3): 111–126.

Schulz, S. 2017. We know (nearly) nothing! In *1st International Workshop on Automated Reasoning: Challenges, Applications, Directions, Exemplary Achievements*.

Schulz, S.; and Möhrmann, M. 2016. Performance of clause selection heuristics for saturation-based theorem proving. In *International Joint Conference on Automated Reasoning*, 330–345. Springer.

Silver, D.; Hubert, T.; Schrittwieser, J.; Antonoglou, I.; Lai, M.; Guez, A.; Lanctot, M.; Sifre, L.; Kumaran, D.; Graepel, T.; et al. 2017a. Mastering chess and shogi by self-play with a general reinforcement learning algorithm. *arXiv preprint arXiv:1712.01815* .

Silver, D.; Schrittwieser, J.; Simonyan, K.; Antonoglou, I.; Huang, A.; Guez, A.; Hubert, T.; Baker, L.; Lai, M.; Bolton, A.; et al. 2017b. Mastering the game of go without human knowledge. *Nature* 550(7676): 354.

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

Sutton, R. S.; and Barto, A. G. 1998. Reinforcement learning: an introduction MIT Press. *Cambridge, MA* .

Taylor, M. E.; Matuszek, C.; Smith, P. R.; and Witbrock, M. J. 2007. Guiding Inference with Policy Search Reinforcement Learning. In *Proceedings of the Twentieth International Florida Artificial Intelligence Research Society Conference*, 146–151. URL http://www.aaai.org/Library/FLAIRS/2007/flairs07-027.php.

Urban, J. 2006. MPTP 0.2: Design, implementation, and initial experiments. *Journal of Automated Reasoning* 37(1-2): 21–43.

van Otterlo, M. 2005. A Survey of Reinforcement Learning in Relational Domains. Number 05-31 in CTIT Technical Report Series. Centre for Telematics and Information Technology (CTIT).

Vaswani, A.; Shazeer, N.; Parmar, N.; Uszkoreit, J.; Jones, L.; Gomez, A. N.; Kaiser, Ł.; and Polosukhin, I. 2017. Attention is all you need. In *Advances in neural information processing systems*, 5998–6008.

Wang, M.; Tang, Y.; Wang, J.; and Deng, J. 2017. Premise Selection for Theorem Proving by Deep Graph Embedding. In *Advances in Neural Information Processing Systems 30: Annual Conference on Neural Information Processing Systems 2017*, 2783–2793.

Whitehead, S.; Ji, H.; Bansal, M.; Chang, S.-F.; and Voss, C. 2018. Incorporating Background Knowledge into Video Description Generation. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, 3992–4001. Brussels, Belgium: Association for Computational Linguistics. URL https://www.aclweb.org/anthology/D18-1433.

Yu, H.; Wang, J.; Huang, Z.; Yang, Y.; and Xu, W. 2016. Video paragraph captioning using hierarchical recurrent neural networks. In *CVPR*.

Zombori, Z.; Csiszárik, A.; Michalewski, H.; Kaliszyk, C.; and Urban, J. 2019. Towards Finding Longer Proofs. *arXiv preprint arXiv:1905.13100* .

Zombori, Z.; Urban, J.; and Brown, C. E. 2020. Prolog Technology Reinforcement Learning Prover. *arXiv preprint arXiv:2004.06997* .

A Appendix

A.1 Hyperparameter Tuning and Experimental Setup

We used gradient-boosted tree search from scikit-optimize[2] to find effective hyper-parameters using 10% of the Mizar dataset. This returned the hyperparameter values in Table 3. The maximum time limit for solving a problem was 100 seconds. Hyper-parameter tuning experiments were conducted over a cluster of 19 CPU (56 x 2.0 GHz cores & 247 GB RAM) and 10 GPU machines (2 x P100 GPU, 16 x 2.0 GHz CPU cores, & 120 GB RAM) over 4 to 5 days.

[2]https://scikit-optimize.github.io/

Once the best hyperparameters were found, we ran TRAIL and its competitors (see Section 4.1 and Appendix A.5) on a CPU machine with 56 x 2.0 GHz cores & 247 GB RAM. At the end of each iteration, collected training examples were shipped to a dedicated GPU server (with 2 x P100 GPUs, 16 x 2.0 GHz CPU cores, & 120 GB RAM) which trains and updates TRAIL's policy network.

Parameter	Value
Chains Patterns	500
Sub-walks	2000
k layers	2
units per layer	161
dropout	0.57
λ (reg.)	0.004
$2d$ (sparse vector size)	645
τ (temp.)	1.13
τ_0 (temp. threshold)	11
embedding layers	4
dense embedding size	800
reward normalization	(i) normalized by difficulty

Table 3: Hyperparameter values

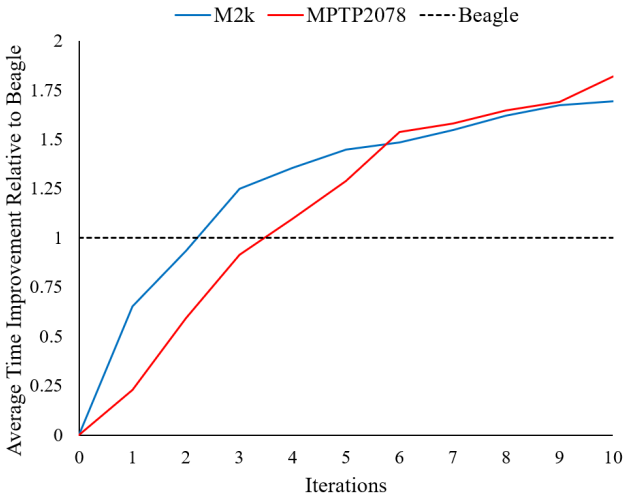

Figure 4: TRAIL's average proof time improvement relative to Beagle (i.e., Beagle's average time to find a proof divided by TRAIL's average time to find a proof)

A.2 Underlying Reasoner

The current implementation of TRAIL uses Beagle (Baumgartner, Bax, and Waldmann 2015) as its underlying inference execution system. This is purely an implementation choice, made primarily due to Beagle's easily modifiable open source code and friendly license. The purpose of Beagle in TRAIL is to execute the actions selected by the TRAIL learning agent; i.e., Beagle's proof guidance was completely disabled when embedded as a component in TRAIL and whenever Beagle reaches a decision point at the level of clause selection, it delegates the decision to TRAIL's policy to decide the next action to pick. When TRAIL passes an inference action to Beagle to execute, it does not specify any additional restrictions, thus ordering constraints and literal selection specified for Beagle's default settings are used by Beagle when it executes the action. Redundancy elimination is also allowed (e.g., proper subsumption deletion), with the exception of all backward simplification techniques.

Using an off-the-shelf reasoner (like Beagle) as a reasoning shell is to ensure that the set of inference rules available to TRAIL are both sound and complete, and that all proofs generated can be trusted. Beagle only executes the actions selected by TRAIL and can thus be replaced by any saturation-based reasoner capable of applying FOL inference rules.

A.3 TRAIL's Learning

Table 4 shows the performance of TRAIL across iterations. Compared to the first iteration, TRAIL managed to solved 547 more problems on MPTP2078 and 519 more problems on M2k. This indicates that TRAIL is learning rather quickly, beating rlCop and plCop on both datasets by the fifth iteration of learning. Interestingly, TRAIL's performance monotonically increases over the iterations, which indicates that it is not overfitting to a particular subset of the problems within either dataset.

We also show in Figure 4 the speed at which TRAIL finds a proof as compared to Beagle. As can be seen in the figure, TRAIL surpasses Beagle's speed rather quickly (at the second iteration for M2k and the fourth iteration for MPTP). One possibility for this is that TRAIL is initially solving an easier subset of problems, as evidenced by the fact that at both of those iterations, TRAIL is actually solving fewer problems than Beagle (see Table 4). However, by the time TRAIL reaches iteration 8 on M2k, it solves more problems than Beagle with a 1.6x improvement in terms of time. Similarly on MPTP2078, TRAIL iteration 5 managed to solved more than 160 problems than Beagle in 1.29x better time.

A.4 Statistical Significance Tests

Table 5 shows the performance of TRAIL compared to learning and traditional theorem provers. This table repeats the results reported in Table 1 in Section 4.1 with statistical significance tests ($p < 0.05$) relative to TRAIL. For example, state-of-the-art traditional theorem prover E outperforms all other approaches including TRAIL. E outperforms TRAIL in a statistically significant way with $z = -16.9$ on M2k and $z = -2.7$ on MPTPT2078 dataset. On the other hand, TRAIL outperforms Beagle in a non-significant way on M2k ($z = 0.6$) and in a significant way on MPTP2078 ($z = 5.3$). Furthermore, all TRAIL's improvements over mlCop, rlCop and plCop are statistically significant.

A.5 rlCop and plCop Experiments

As mentioned in Section 4.1, the numbers reported in Table 1 for plCop and rlCop are taken from their papers (Kaliszyk et al. 2018; Zombori, Urban, and Brown 2020). We also replicated their performance under our exact hardware and time

	1	2	3	4	5	6	7	8	9	10
M2k	1042	1266	1428	1490	1507	1527	1533	1549	1552	**1561**
MPTP2078	363	552	680	730	806	844	858	878	877	**910**

Table 4: TRAIL's performance across iterations

		M2k	Stat. Sig.	MPTP2078	Stat. Sig.
Traditional	E	**1922**	✓ (z=-16.9)	**998**	✓ (z=-2.7))
	Beagle	1543		742	✓ (z=5.3)
	mlCop	1034	✓ (z=17.4)	502	✓ (z=13.4)
RL-Based	rlCop	1235	✓ (z=11.2)	733	✓ (z=5.6)
	plCop	1359	✓ (z=7.2)	782	✓ (z=4.0)
	TRAIL	**1561**		**910**	

Table 5: Number of problems solved in M2k and MPTP2078, best two approaches in **bold**. Statistically significant differences ($p < .05$) *relative to TRAIL* are marked with ✓.

	M2k	MPTP2078
TRAIL	**1,561**	**910**
rlCop (w/o paramodulation)	1,148	543
rlCop (w/ paramodulation)	1,238	563
plCop (w/o paramodulation)	1,222	707
plCop (w/ paramodulation)	1,301	773

Table 6: plCop and rlCop performance using same hardware and time limit (100 seconds) as TRAIL

constraints. In particular, we used the authors' source code available at https://github.com/zsoltzombori/plcop which contains the implementation of both rlCop and plCop. We used the same default parameters from plCop's configuration files. We noticed, however, that they have two prominent configurations for each dataset (with and without paramodulation) and as a result we decided to report both configurations on each dataset. Table 6 shows the performance of TRAIL, plCop and rlCop on the same hardware with 100 seconds time limit. plCop performance is very close to what we have in Table 1 in Section 4.1 while rlCop numbers are lower. To avoid any confusion, we decided to use the best performance for both rlCop and plCop in Table 1 in Section 4.1 which is what the authors reported in their paper. This experiment is to show that the hardware used and the time limits are comparable and hence fair comparison can be made.