EPiC Series in Computing

Volume 94, 2023, Pages 96–111

Proceedings of 24th International Conference on Logic for Programming, Artificial Intelligence and Reasoning

How Much Should This Symbol Weigh? A GNN-Advised Clause Selection

Filip Bártek[1,2] and Martin Suda[1]

[1] Czech Institute of Informatics, Robotics and Cybernetics
[2] Faculty of Electrical Engineering
Czech Technical University in Prague, Czech Republic
{filip.bartek,martin.suda}@cvut.cz

Abstract

Clause selection plays a crucial role in modern saturation-based automatic theorem provers. A commonly used heuristic suggests prioritizing small clauses, i.e., clauses with few symbol occurrences. More generally, we can give preference to clauses with a low *weighted symbol occurrence count*, where each symbol's occurrence count is multiplied by a respective symbol weight. Traditionally, a human domain expert would supply the symbol weights.

In this paper, we propose a system based on a graph neural network that learns to predict symbol weights with the aim of improving clause selection for arbitrary first-order logic problems. Our experiments demonstrate that by advising the automatic theorem prover Vampire on the first-order fragment of TPTP using a trained neural network, the prover's problem solving capability improves by 6.6 % compared to uniformly weighting symbols and by 2.1 % compared to a goal-directed variant of the uniformly weighting strategy.

1 Introduction

Clause selection is a major choice point in modern saturation-based provers for first-order logic. It amounts to deciding, at each iteration of the saturation loop, which clause should be the next to activate, i.e., to participate in generating inferences with the previously activated clauses. Since the generating inferences form the backbone of the sought proof, a perfect clause selection oracle would completely alleviate the need for a search. Practical clause selection heuristics in the tightly optimized state-of-the-art provers, however, also need to be efficient to compute. The most widely used general-purpose ones are based on the first-in/first-out principle and simple symbol counting [23].

A promising line of research is to automatically derive new clause selection heuristics from prover successes using machine learning [21, 6]. In recent years, learning systems such as ENIGMA [11, 4, 13] or Deepire [28] have been able, for example, to substantially improve the performance of the base prover on problems exported from a large mathematical corpus [14]. As with classical heuristics, ensuring that the learned part is not too expensive to evaluate was key to success [12, 27].

A natural generalization of the symbol counting heuristic, mentioned, for example, in the E prover [24] manual [22] under the name `FunWeight`, allows each symbol to have a user-specified weight to act in the heuristic as a multiplier for the respective symbol's occurrence count. While the extra degrees of freedom make the heuristic, in theory, strictly more powerful than its simple variant, only a true expert on a particular problem domain or on a specific encoding could possibly possess the knowledge to capitalize on the additional power easily. The main reason for this is that the proof search is typically quite fragile [29], making it very hard to draw conclusions about cause and effect. Additionally, the idea of using the extra generality to develop new proving strategies comes with a complication: The symbol weights depend on the problem's signature, so there is no immediate general way of specifying them in advance.

In this paper, we design and implement a learning system that automatically recommends good symbol weights for clause selection for any given problem. The system is based on a graph neural network (NN) that operates on the structure of the input problem's clause normal form. Similarly to ENIGMA or Deepire, we use previous prover successes to train the system, but rely on a novel scheme for balancing positive (coming from a proof) and negative (not coming from proof) training clauses. Because the GNN only computes in a one-time preprocessing phase to recommend the weights, there is no noticeable performance penalty during actual saturation.

We train and evaluate our system on first-order problems from Thousands of Problems for Theorem Provers (TPTP) [30], finding it to substantially improve performance over a baseline strategy with uniform symbol weights. TPTP is a very diverse problem collection, which means that the knowledge the system extracts through training must be, to a large degree, general-purpose and readily useful elsewhere. We attempt to shed some light on what was learned by having a closer look at the recommended weights on several problems that were only solved with our system's help.

The remainder of the paper is organized as follows. Section 2 reviews the basic concepts used throughout the paper. Sections 3 and 4 describe the main task we tackle and our solution—a GNN-based symbol weight recommender. In Section 5, we describe experiments we performed to evaluate our recommender and analyze the results, and in Section 6, we continue the analysis in a case-based manner. We discuss possible modifications and enhancements of our recommender in Section 7 before concluding in Section 8.

2 Background

We assume that the reader is familiar with the basic concepts related to automatic theorem proving. In this section, we provide a brief overview of the terminology relevant to our research with references to detailed explanations.

Formulas in first-order logic (FOL) are built using variables, function and predicate symbols, the equality symbol (which we denote $\approx$), logical connectives, and quantifiers [1, 5]. A FOL problem consists of FOL formulas: a conjecture and zero or more axioms. Solving the problem amounts to finding a proof that the conjecture necessarily follows from the axioms. In refutational proving, we look for a proof by contradiction—we negate the conjecture and proceed to infer a trivial contradiction. A common approach is to convert the problem to an equisatisfiable clause normal form (CNF) problem and to perform the inferences in the clause space, ultimately inferring a trivial contradiction in the form of the empty clause.

A saturation-based automatic theorem prover (ATP), such as Vampire [15] or E [24], relies on a given-clause procedure [18, 23] to orchestrate the processing of inferences. It maintains two sets of clauses: active and passive. At the beginning of a proof search, the passive set is populated with the input clauses. Then, iteratively, a clause is always selected from the passive

C	$W(C)$
$p(X_1, c, X_2) \vee q(X_1)$	$3w(X) + w(p) + w(q) + w(c)$
$g(X_1, h(X_2)) \approx f(g(X_1, X_2), X_1)$	$5w(X) + w(\approx) + w(f) + 2w(g) + w(h)$
$\neg(h(X_1) \approx h(X_2)) \vee X_1 \approx X_2$	$4w(X) + 2w(\approx) + 2w(h)$

Table 1: Examples of clauses and their symbol-counting weights

set and put into the active set. It is made to participate in inferences with all the active clauses, and the newly generated clauses are added to the passive set. The proof search is completed as soon as the empty clause is inferred.

In a typical proof search, selecting inauspicious clauses quickly clutters the active set with useless clauses, slowing the proof search considerably and thus reducing the chance of finding a proof in a reasonable time. On the contrary, accurately selecting proof clauses leads to quick success. Thus, the quality of the clause selection heuristic is crucial for the performance of the prover [23]. Most provers interleave several priority queues of passive clauses, each prioritized by a distinct criterion. Commonly utilized criteria include clause age, which favors early-inferred clauses, and symbol count, often referred to as clause weight. These queues are commonly interleaved at a predetermined ratio. For example, in its default configuration, Vampire interleaves age- and weight-based selection at a ratio known as the age-weight ratio [15].

While the proof search may also conclude by depleting the passive set and thus demonstrating the satisfiability of the input clause set, in this work, we are only concerned with refutation proofs and only consider a proof search successful if it infers the empty clause.

3 Task Definition

3.1 Symbol-Counting Clause Weight

Extended signature. Consider a problem P defined over the signature Σ_P (the set of predicate and function symbols in P). We define the *extended signature* Σ_P^+ by extending Σ_P with two fresh symbols: $\approx$ (for the equality predicate) and X (a generic variable). In the following text, we use the term "symbol" in the broader sense of any member of Σ_P^+.

Symbol occurrence count. Given a clause C over Σ_P, we define $S_C : \Sigma_P^+ \to \mathbb{N}$ as the *symbol occurrence count operator* of C: The values of $S_C(\approx)$ and $S_C(s)$ for $s \in \Sigma_P$ are to be computed in an obvious way and $S_C(X)$ is the total number of positions in C at which a variable occurs.

Symbol and clause weight. Any mapping $w : \Sigma_P^+ \to \mathbb{R}^{\geq 1}$ is a *symbol weight assignment* for P. By extension, $W(C) = \sum_{s \in \Sigma_P^+} S_C(s) \cdot w(s)$ is the *symbol-counting weight* of clause C induced by the symbol weight assignment w.

See examples of symbol-counting clause weights in Table 1.

Fairness. When W is used as a clause selection heuristic in an ATP, a weight-based selection chooses the clause that minimizes W among the passive clauses. While we could, in principle, allow symbol weights smaller than 1 (or even negative), constraining the symbol weights with a lower bound greater than 0 makes the induced clause selection heuristic *fair* [23]: For each

$W' \in \mathbb{R}$, there is only a finite number of clauses (up to variable renaming) that weigh at most W', so no clause inferred during the proof search is denied selection indefinitely.[1]

The standard weight criterion used by Vampire assigns the weight 1 to each symbol [15].

3.2 Main Task

We assume a fixed distribution of FOL problems (the target distribution) and a fixed FOL ATP (the target prover) with a fixed time limit (specified in wall-clock time or CPU instructions). Furthermore, we assume that the target prover is a saturation-based prover that selects at least some clauses by a symbol-counting clause weight W instantiated by a configurable symbol weight assignment w.

Our main goal is to create a system that, when presented with a FOL problem P sampled from the target distribution, produces a symbol weight assignment w such that the target prover is likely to solve P within the time limit when using w.

4 Recommender

To solve the main task, we propose a system based on a neural network (NN) that produces a symbol weight assignment. The system is trained on traces of successful proof searches of the target prover on the target problem distribution.

4.1 Input Problem Format

Given a FOL problem, we first convert it to an equisatisfiable CNF problem [17]. The conversion may introduce new function symbols by Skolemizing the existential quantifiers and new predicate symbols by naming some subformulas. In the remainder of this text, we assume that the input problem is specified in CNF.

In addition to the CNF specification of the problem, some metadata are considered part of the input. Specifically, the recommender is given the role of each clause (axiom, assumption, or negated conjecture), and, for each symbol, whether the symbol was introduced during preprocessing (Skolemization or subformula naming) and whether it appears in the conjecture.

4.2 Prediction

4.2.1 Graph Representation of Problems

Given a problem in CNF, we convert it to a graph. Each node in the graph represents a syntactic element of the input problem, such as a clause, an atom, a term, a predicate or function symbol, or a variable. The edges represent direct relations of the syntactic elements; for example, a clause is connected to all the atoms it contains and an atom is connected to a predicate symbol and zero or more arguments. The nodes and edges are typed according to their meaning: The type of a node denotes the role of the corresponding syntactic element (for example, “clause” or “atom”), and the type of an edge denotes the relation it represents (such as “clause contains atom”) and is typically fully specified by the types of the adjacent nodes.

[1] Strictly speaking, interleaving weight-based selection with age-based selection suffices to ensure fairness. However, in practice, we observed that allowing negative symbol weights may have undesirable effects even when age-based selection is present; see Section 5.5.3.

4.2.2 Graph Neural Network

The graph neural network (NN) that constitutes the predictive core of the recommender is a relational graph convolutional network (GCN) [20]. It maintains an embedding (a numeric vector), in our case of length 16, for each node in the input graph.

Initially, the embedding of a node is a trainable vector common to all nodes of one type. In the clause and symbol nodes, the initial embedding is augmented with the metadata extracted from the input problem.[2]

Four message-passing layers follow: In each of these layers, each node embedding is updated by aggregating messages incoming from the node's neighbors. Each message is the output of a trainable affine transformation of the embedding of the source node. The trainable coefficients of the transformation are shared by all edges of the same type in the same layer.[3]

After all four layers are evaluated, the final node embeddings are available. To obtain an output weight $w(s)$ of symbol $s \in \Sigma_P$, the final embedding of the node that represents s is transformed using an affine transformation, the trainable coefficients of which are shared by all symbols, followed by an output activation function. To obtain an output variable weight $w(X)$, the embeddings of all variable nodes are aggregated by summation and transformed by a trainable affine transformation followed by an output activation function. Similarly, the output weight of equality $w(\approx)$ is calculated by aggregating the embeddings of all equality atom nodes.

We use the output activation function $a(x) = 1+\mathrm{softplus}(x)$, where $\mathrm{softplus}(x) = \log(e^x+1)$. This ensures that every output weight is greater than 1, which ensures fairness of the clause selection, as described in Section 3.1. The empirical experiments described in Section 5.5.3 confirm the efficacy of this activation function.

In the way just described, GNN transforms an input problem graph into a symbol weight assignment w. The quality of the assignment can be empirically assessed by trying to solve the input problem using the target prover with a symbol-counting clause selection heuristic induced by w.

4.3 Training

Next, we explain how the NN is trained. Our approach exploits the fact that the symbol weight assignment w predicted by the NN can be used outside of the prover to compute the weight $W(C)$ of an arbitrary clause C and that $W(C)$ is differentiable with respect to the trainable parameters of the NN.

4.3.1 Training Data Collection

We train the NN on traces of successful proof searches. To collect training data, we run the target prover on the training problem set. We use a fixed clause selection heuristic, namely one that interleaves an age-based queue with symbol-counting with uniform symbol weights at the ratio 1:5. We enforce a time limit to curb impractically long proof searches.

From each successful proof search produced this way, we extract all the selected clauses. We divide the selected clauses into two sets: proof clauses $\mathcal{C}_+$ and nonproof clauses $\mathcal{C}_-$, depending on whether or not the clause contributed to the proof. We limit the size of each of these sets to 10 000 clauses by subsampling if necessary.

[2] More precisely, the initial embedding of a node is the concatenation of a node-specific metadata vector (empty in all nodes except clause and symbol nodes; see Section 4.1), and a trainable vector common to all nodes of one type.

[3] The architecture and hyperparameters of the GNN are the same as those used in our earlier work on recommending symbol precedences [2] which we refer the reader to for additional details.

4.3.2 Training Example

We construct a training example from each pair of a proof clause C_+ and a nonproof clause C_- from the Cartesian product $\mathcal{C}_+ \times \mathcal{C}_-$. We interpret such an example as "C_+ should be selected before C_-". Since we know the occurrence counts of variables and symbols in each of these clauses, we can use the weights predicted by the NN to predict clause weights $W(C_+)$ and $W(C_-)$.

We define the predicted probability of C_+ being preferable to C_- according to the NN as $p(C_+, C_-) = \text{sigmoid}(W(C_-) - W(C_+))$, where $\text{sigmoid}(x) = \frac{1}{1+e^{-x}}$ is a standard activation function that maps real numbers to the interval $[0, 1]$.

4.3.3 Accuracy

We consider the example (C_+, C_-) classified correctly if and only if $p(C_+, C_-) > 0.5$. Since this occurs if and only if $W(C_-) > W(C_+)$, this ultimately corresponds to the prover selecting C_+ before C_-.

Given a set of examples, the prediction *accuracy* of the NN is defined as the number of correct classifications divided by the total number of examples. While the main task of the NN remains to achieve a strong performance of the target prover, we train the NN on this proxy task of achieving high accuracy on the training data. The proxy task has two crucial practical advantages:

1. Proxy performance evaluation is relatively inexpensive, so we can quickly assess a trained NN using its proxy performance.

2. Since p is differentiable with respect to the trainable parameters of the NN, we can use gradient descent [8] to optimize the proxy performance of the NN.

4.3.4 Loss Function

To conclude the exposition of the training process, we define the loss on training example (C_+, C_-) as the negative log-likelihood:

$$\ell = -\log p(C_+, C_-) = -\log \text{sigmoid}(W(C_-) - W(C_+)) = \log(1 + e^{W(C_+)-W(C_-)})$$

This is a standard choice for binary classification tasks [10].

4.3.5 Training Procedure

We use minibatch stochastic gradient descent [8] to minimize the total loss on the training set. We use a weighted sum to aggregate the loss values of multiple proof searches of different sizes to reflect our intuition that every successful proof search has the same value. Training proceeds in epochs, each using each training example exactly once.

When optimizing the proxy performance, we are effectively training a clause ranking function W based on training examples of the form "$W(C_+)$ should be smaller than $W(C_-)$". The general task of training a NN to rank arbitrary objects has been explored in RankNet [3]. In our case, unlike RankNet, W is a mere proxy metric, not a final goal, so we need to ensure that W represents our main task sufficiently. This representation is realized by the decomposition of W into the symbol-wise components w.

4.4 Evaluation

The quality of the NN can be assessed by measuring proxy performance or empirical performance. *Proxy performance* is the mean prediction accuracy on a set of proof searches (see Section 4.3.3). To determine the *empirical performance* on a set of problems, we run the target prover using symbol weight assignments predicted by the NN for the respective problems. Empirical performance is measured as the number or proportion of proofs found within a fixed proof search instruction limit.

For the experiments reported in the next section, we implemented a prototype recommender. We did not attempt to optimize the recommender to reduce the computational overhead introduced in addition to the proof search. In the version of the recommender we used in the experiments[4], this overhead for solving a single input problem is of the same order as the time limit we used for empirical evaluation (5×10^{10} CPU instructions). When solving a batch of problems, the overhead can be amortized to a great extent, so the recommender can already be useful in such a setting. Further optimization is necessary to make the recommender practically useful for solving isolated problems with time limits in the order of 10^{10} instructions[5] or less.

5 Experiments

To assess the strength of the architecture and training procedure described in Section 4, we performed a collection of experiments. In the experiments, we targeted the first-order fragment of the Thousands of Problems for Theorem Provers (TPTP) problem library as our target problem distribution and the ATP Vampire as our target prover.

5.1 System

We performed our experiments on a computer with a CPU Intel Xeon Gold 6140 (72 cores @ 2.30 GHz) and 502 GiB random access memory (RAM).

5.2 Target Prover

Vampire [15] is a powerful ATP that has shown success especially in proving FOL theorems. We modified Vampire

- to allow exporting the signature and the graph representation of the input problem,
- to print all the clauses selected in the proof search, and
- to allow the user to override the symbol weight assignment for clause selection.[6]

We used Vampire in its default configuration with the following exceptions: the Discount saturation algorithm, age-weight ratio 1:5, no AVATAR [31], and a limit of 5×10^{10} CPU instructions[7]. Using the Discount saturation algorithm, as opposed to the default limited resource strategy [18], removes a prominent source of nondeterminism from the proof search. With Discount, the

[4] Source code of the recommender: https://github.com/filipbartek/vampire-ml/tree/lpar2023
[5] The order of seconds on a contemporary CPU
[6] Source code of the modified Vampire: https://github.com/filipbartek/vampire/tree/lpar2023
[7] In our experimental setup, 5×10^{10} instructions were executed in approximately 16 seconds of wall-clock time.

age-weight ratio 1:5 is stronger than the default 1:1 and increases the significance of weight-based clause selection. Disabling AVATAR is a convenience that facilitates the parsing of the selected clauses.

5.3 Problem Sets

We first collected all the 17 436 problems annotated as clause normal form (CNF) or first-order form (FOF) in TPTP [30] v8.1.2. We set aside 3487 (approximately 20 %) randomly sampled problems as the test set for the final evaluation. We divided the remaining 13 949 problems in a ratio of approximately 80 to 20 into 11 159 training problems and 2202 validation problems.

The validation set was used to measure the performance of the trained NN during the training to compensate for the possible overfitting of the NN to the training set. Analogously, performing the final evaluation on a separate test set ensures that the observed success is not due to overfitting to the validation set.

Due to practical concerns, we restricted our attention to problems whose graph representation has at most 100 000 nodes. Therefore, the training, validation, and test sets have 10 016, 2492, and 3149 problems, respectively.

5.4 Training

First, we ran Vampire on the training and validation problem sets. The proofs found in this way form the training and validation data. Each proof search trace was subsampled to at most 10 000 proof clauses and at most 10 000 nonproof clauses.

We trained a GNN as described in Section 4. Each training epoch was followed by a proxy evaluation on the training and validation problem sets. Every 10 epochs, we performed an empirical evaluation. After approximately 60 hours (160 epochs) of training, we chose the final model based on the performance observed on the validation set.

Since the accuracy (both training and validation) grew smoothly within the first 80 epochs and turned noticeably noisy afterward, we chose the final version of the GNN among the first 80 epochs to avoid accidental overfitting to the validation problem set. The final version has the highest observed number of proofs found on the validation set.

5.5 Results

5.5.1 Empirical Performance

We evaluated the empirical performance of the final trained NN and the standard symbol weight assignment that assigns the weight 1 to each symbol (denoted as baseline). Table 2 and Figure 1 show the results of the evaluation. The GNN improves the performance by approximately 6.6 % over the baseline. Furthermore, Figure 1 shows that the performance of the trained GNN generalizes to instruction limits up to 4 times as large as the limit used to generate the training data (5×10^{10} instructions).

To get a better sense of the scale of this result, we evaluated the baseline with AVATAR enabled. AVATAR is a powerful and sophisticated Vampire feature that aids saturation-based theorem proving using a propositional SAT solver [31]. When we enabled AVATAR under the same conditions as the baseline, we observed an improvement of 5.9 % over the baseline, which is of a magnitude similar to the improvement brought about by our trained GNN.

Configuration	Proofs found		Compared to B		
	/3149	%	+	−	%
Trained GNN	1494	47.4 %	+141	−49	+6.6 %
Baseline (B)	1402	44.5 %	+0	−0	+0.0 %
B + AVATAR	1485	47.2 %			+5.9 %
B + Goal-directed	1463	46.5 %			+4.4 %

Table 2: Results of the final empirical evaluation. The reported performance is the number of proofs found on the test set (3149 problems) within 5×10^{10} CPU instructions per proof search.

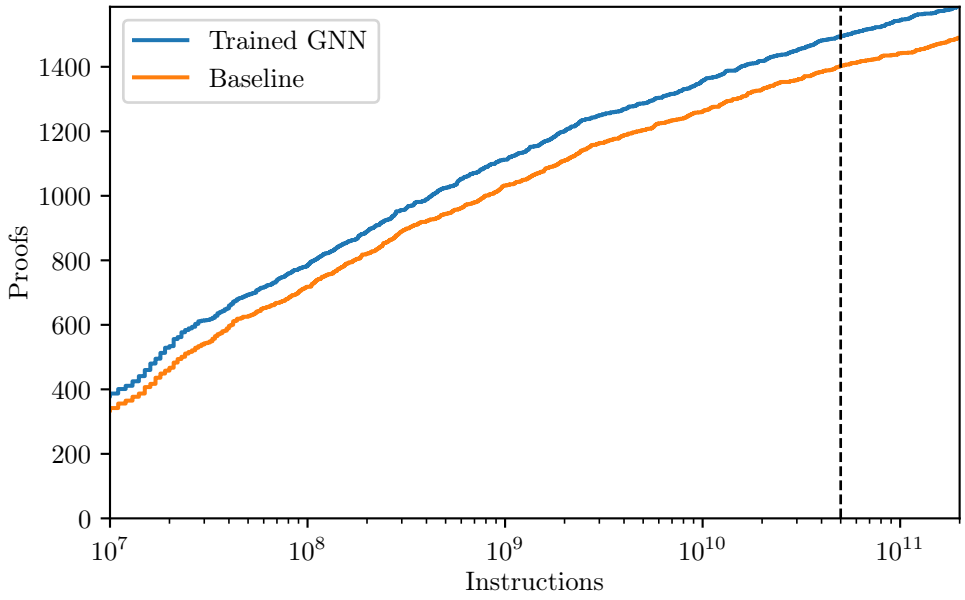

Figure 1: Total number of proofs found on the test set (3149 problems) in a given number of CPU instructions per proof search. The dashed line shows the default evaluation limit of 5×10^{10} instructions that was used to generate the training data. The configurations were evaluated to up to 2×10^{11} instructions.

5.5.2 Iteration Efficiency

Figure 2 shows that the trained GNN is relatively efficient in terms of iterations of the saturation loop. This is an expected outcome: The training indirectly incentivizes the prover to select proof clauses in relatively early iterations, making the proof search conclude in relatively few iterations. There were 1453 test problems that both the GNN and the baseline solved within 2×10^{11} instructions. Of these, 828 were solved in fewer iterations by the GNN, while only 471 were solved in fewer iterations by the baseline.

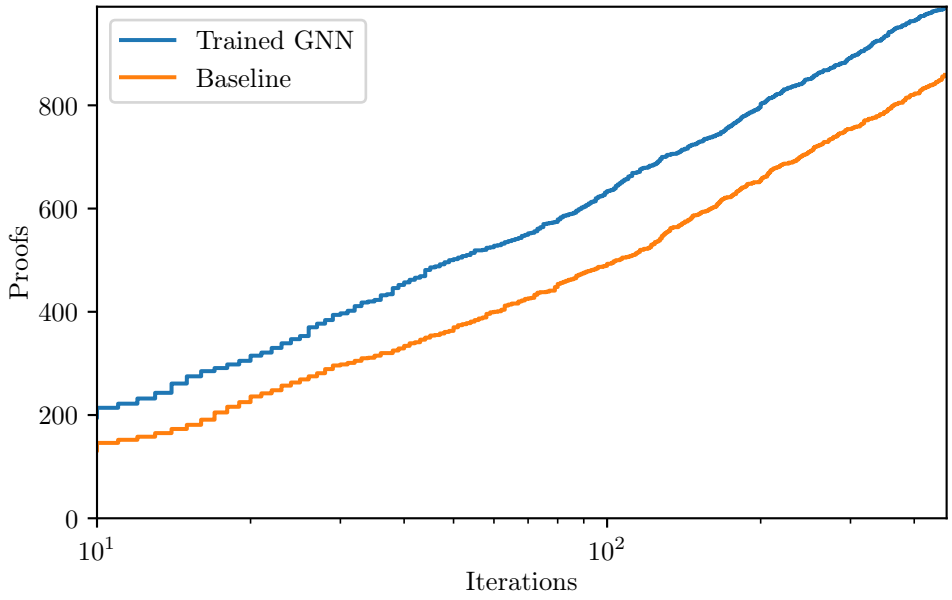

Figure 2: Total number of proofs found on the test set (3149 problems) in a given number of iterations of the saturation loop per proof search. Only proofs found within 2×10^{11} CPU instructions and 463 iterations are included. Proof searches with more than 463 iterations are not shown to ensure that no more than 10 % proofs are obscured from the plot by the instruction limit.

5.5.3 Activation Function Comparison

As described in Section 4.2.2, we forced the output symbol weights to have values greater than 1. To understand what the effect of allowing weights smaller than 1 would be, we trained the GNN for 24 hours with various final activation functions: $a_1(x) = 1 + \text{softplus}(x)$ (default), $a_2(x) = 0.1 + \text{softplus}(x)$ (greater than 0.1), $a_3(x) = \text{softplus}(x)$ (greater than 0), and $a_4(x) = x$ (unconstrained). In almost all observed epochs, the empirical performance on the validation set decreased from a_1 to a_4. Manual inspection of several failed proof searches performed with the weights found with the activation function a_4 confirmed our suspicion that allowing negative symbol weights can often lead to infinite derivation chains of increasingly long clauses with decreasing negative weights.

6 Interpreting the Learned Advice

The details behind the computation performed by a trained network are notoriously hard to interpret. Although this holds for our GNN, too, we at least have the possibility to study its end result—the recommended symbol weights—and to attempt to discover some important features of the recommendations that lead to an efficient proof search. In this section, we report our findings from manually analyzing the GNN symbol weight recommendations and

the corresponding Vampire runs on several TPTP problems. To have the highest chance of noticing impactful features, we selected the problems so that Vampire with recommendations solves each of them fast while the baseline configuration fails (under the limit of 5×10^{10} CPU instructions).

Small conjecture symbols. For most of the problems we examined, symbols appearing in the conjecture were assigned weights very close to the smallest allowed value, 1. At the same time, other symbols were assigned weights ranging from 1 to approximately 10. Having the conjecture symbols weigh less than the rest is a way to make the search goal-directed, a well-known, generally successful strategy [e.g., 23].[8]

We find the fact that the GNN "reinvents" this strategy from the training data quite encouraging. However, it also raises the question of whether this is not all that has been learned. To check this possibility, we reran the baseline configuration in a goal-directed mode.[9] It solved 1463 (46.5 %) problems in the test set, compared to 1494 (47.4 %) solved by the trained GNN (cf. Table 2). In other words, the trained GNN solved 2.1 % more problems than the goal-directed strategy. In summary, while goal-directedness seems to play an important role, it does not explain all the gains obtained through the GNN symbol recommendation.

Bumped-up "Tseitin variables". The TPTP problem `ALG066+1` contains a ground description of a finite algebra with one binary operation `op` on five elements `e0,...,e4` (one of which is a `unit`). Its conjecture is a formula with a complex Boolean structure, for which efficient clausification Vampire introduces 130 new predicate symbols $\texttt{sP}_i$ as names for subformulas [17].

The baseline run on this problem does not finish in a reasonable time, activating thousands of clauses such as `sP126 | sP125 | sP107 | sP88 | sP58`, where the resolution rule systematically "distributes out" parts of the conjecture's Boolean structure (previously abstracted by new names during preprocessing). However, the GNN-advised Vampire finds a proof in 0.25 s and approx. 2500 iterations of the main loop. The search proceeds differently because the GNN assigns high weights (between 2 and 4) to the new predicate symbols $\texttt{sP}_i$.

It seems that the rule discovered here is that it may pay off to postpone working on clauses that are heavy by containing too many subformula names. We note, however, that `ALG066+1`, being ground, becomes trivial also for the baseline when equipped with AVATAR [31], as all the Boolean structure gets then efficiently dealt with by the SAT solver.

Variable weight. We noticed that the trained GNN almost always assigns a low weight (a value very close to 1) to variables. (Recall that in our setting, there is only one global variable occurrence weight for each problem.) This could be interpreted as "Other things being equal, a general clause is more likely to help finish a proof than a specific one.", which is actually obvious when thinking about logical strength and the ultimate goal of finding a contradiction. At the same time, some caution should be in order, as we also know that a clause with more variables provides more opportunities for inferences, which often leads to a more explosive proof search.

Curious to see whether the learned tendency to have variables weigh less than most other symbols was not stopped early by our fairness constraint (stating that any recommended weight must be ≥ 1, cf. Section 4.2.2), we reran Vampire with the GNN symbol weights but overriding

[8] An interesting variant of this idea in the context of equational proving has been given by Smallbone [25].

[9] Specifically, by setting Vampire's *non-goal weight coefficient* option [15] to 5. This makes the search goal-directed by globally multiplying the weight of every clause not derived (directly or indirectly) from the conjecture by 5. The concrete value 5 has been found to lead to good performance in previous experiments on TPTP [26].

variable weight to 0.5. However, this resulted in a decline in overall performance, and the number of problems uniquely solved by this modified configuration was not too high either.

This makes us conclude here that the variable weight of 1 is probably working as a reference value for the GNN recommendations, in the sense that the weights assigned to other symbols are appropriately scaled in relation to this unit, and, in its internal reckoning, they are neither too low nor too high.

Rarely useful axioms? The problem `SET016+1` formalizes the set theory lemma that the first components of equal ordered pairs are equal. The conjecture is stated in the context of 43 common set theory axioms and definitions included from the file `SET005+0.ax`.

Vampire's clausification produces 10 Skolem functions. The last four of these correspond to the universally quantified variables of the conjecture and get recommended weights close to 1 by our GNN. At the other end of the scale, there are the Skolems `sK1` of arity 0 with weight 5.1 and `sK4` of arity 1 with weight 4.1. These correspond to the witnesses from the axioms of infinity and regularity, respectively. We speculate that these relatively "exotic" axioms were rarely, if at all, useful in the proofs used for training (there were 41 training problems that included the axiom file `SET005+0.ax`) and, at the same time, that the GNN was able to recognize them just from their structure and how they relate to the surrounding symbols and other axioms. If these assumptions are correct, assigning the corresponding (uniquely determined) Skolems large weights is a good way to steer the search away from exploring possible inferences with such axioms. However, we did not find a way to easily confirm this rigorously.

7 Future Work

This paper has shown that generalizing the standard clause selection heuristic and automatically determining the symbol weights improves the performance of the prover. Arguably, the clause weight scheme could be generalized further with a negligible computational overhead by a conservative augmentation of the set of clause features used to compute the clause weight, for example, by the number of positive literals[10], the number of distinct variables, or the depth of the clause. Further generalization of the recursively implemented (via summation) symbol-counting clause weight heuristic would yield a recursive neural network (NN) operating on the term structure. An RNN-based clause selection heuristic has been trained by Chvalovský et al. [4] on the proxy task of proof clause classification.

Although allowing clause feature weights smaller than 1 has proven detrimental to the performance of the trained system (see Section 5.5.3), we hypothesize that this effect could be compensated by enriching the training data using failed proof searches from the empirical evaluations.

Using additional or different Vampire base configurations for training data collection and final evaluation, for example, one that enables AVATAR, could yield an even stronger trained prover.

Finally, using constrained logistic regression allows one to find nearly optimal symbol weights (in terms of proxy performance) for one or more successful proof searches with a common signature. It would be interesting to see if we could use such a method to find nearly optimal weights for some interesting domain with a shared signature, such as the Mizar Mathematical Library [9]. In a more general setting, we could collapse the problem-specific symbol counts into a small number of common symbol-counting features such as the number of literals (the sum of

[10] Cf. parameter `pos_mult` of the generic weight functions in E [22].

all predicate and equality occurrence counts), the total number of non-Skolem function symbol occurrences, or the total number of occurrences of predicates introduced in preprocessing. Another way to combine logistic regression with misaligned signatures could involve decomposing the weight of a symbol into common symbol features (such as the arity or an indication of being introduced in preprocessing) and learning, for each symbol category (such as predicate symbols or function symbols), the weights of these symbol features from the data.

8 Conclusion

We proposed and evaluated a symbol weight recommender based on a GNN. The recommender instantiates weights for the weighted symbol occurrence count, a clause selection heuristic we implemented in the FOL ATP Vampire. When trained on a set of proof search traces produced using a baseline Vampire configuration, the recommender outperforms the baseline by 6.6 % in the number of proofs found within a fixed instruction limit. This is, to the best of our knowledge, the first time a machine-learned clause selection heuristic has achieved a comparable improvement on the diverse TPTP problem library.

In this work, we make the following main observations. First, classifying proof-nonproof pairs of clauses is a good proxy task for training a clause selection heuristic. Second, a good symbol weight recommendation can boost the performance of our chosen baseline strategy to a similar degree as enabling the AVATAR architecture. Third, constraining the symbol weights with a nontrivial lower bound helps the performance of a trained recommender significantly. Finally, strong symbol weight assignments tend to assign variable occurrences a small weight.

9 Acknowledgments

We thank Cezary Kaliszyk for discussions. This work was supported by the Czech Science Foundation project no. 20-06390Y (JUNIOR grant), the European Regional Development Fund under the Czech project AI&Reasoning no. CZ.02.1.01/0.0/0.0/15_003/00004, the project RICAIP no. 857306 under the EU-H2020 programme, and the Grant Agency of the Czech Technical University in Prague, grant no. SGS20/215/OHK3/3T/37. Lastly, we thank the anonymous reviewers for valuable comments.

References

[1] Leo Bachmair and Harald Ganzinger. Resolution theorem proving. In Robinson and Voronkov [19], pages 19–99. ISBN 0-444-50813-9. doi:10.1016/b978-044450813-3/50004-7.

[2] Filip Bártek and Martin Suda. Neural precedence recommender. In Platzer and Sutcliffe [16], pages 525–542. ISBN 978-3-030-79875-8. doi:10.1007/978-3-030-79876-5_30.

[3] Christopher J. C. Burges, Tal Shaked, Erin Renshaw, Ari Lazier, Matt Deeds, Nicole Hamilton, and Gregory N. Hullender. Learning to rank using gradient descent. In Luc De Raedt and Stefan Wrobel, editors, *Machine Learning, Proceedings of the Twenty-Second International Conference (ICML 2005), Bonn, Germany, August 7-11, 2005*, volume 119 of *ACM International Conference Proceeding Series*, pages 89–96. ACM, 2005. ISBN 1-59593-180-5. doi:10.1145/1102351.1102363.

[4] Karel Chvalovský, Jan Jakubuv, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In Fontaine [7], pages 197–215. ISBN 978-3-030-29435-9. doi:10.1007/978-3-030-29436-6_12.

[5] Anatoli Degtyarev and Andrei Voronkov. Equality reasoning in sequent-based calculi. In Robinson and Voronkov [19], pages 611–706. ISBN 0-444-50813-9. doi:10.1016/b978-044450813-3/50012-6.

[6] Jörg Denzinger and Stephan Schulz. Learning domain knowledge to improve theorem proving. In Michael A. McRobbie and John K. Slaney, editors, *Automated Deduction - CADE-13, 13th International Conference on Automated Deduction, New Brunswick, NJ, USA, July 30 - August 3, 1996, Proceedings*, volume 1104 of *Lecture Notes in Computer Science*, pages 62–76. Springer, 1996. ISBN 3-540-61511-3. doi:10.1007/3-540-61511-3_69.

[7] Pascal Fontaine, editor. *Automated Deduction - CADE 27 - 27th International Conference on Automated Deduction, Natal, Brazil, August 27-30, 2019, Proceedings*, volume 11716 of *Lecture Notes in Computer Science*, 2019. Springer. ISBN 978-3-030-29435-9. doi:10.1007/978-3-030-29436-6.

[8] Ian J. Goodfellow, Yoshua Bengio, and Aaron C. Courville. *Deep Learning*. Adaptive computation and machine learning. MIT Press, 2016. ISBN 978-0-262-03561-3. URL http://www.deeplearningbook.org/.

[9] Adam Grabowski, Artur Kornilowicz, and Adam Naumowicz. Mizar in a nutshell. *J. Formaliz. Reason.*, 3(2):153–245, 2010. doi:10.6092/issn.1972-5787/1980.

[10] Trevor Hastie, Robert Tibshirani, and Jerome H. Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction, 2nd Edition*. Springer Series in Statistics. Springer, 2009. ISBN 9780387848570. doi:10.1007/978-0-387-84858-7.

[11] Jan Jakubuv and Josef Urban. ENIGMA: efficient learning-based inference guiding machine. In Herman Geuvers, Matthew England, Osman Hasan, Florian Rabe, and Olaf Teschke, editors, *Intelligent Computer Mathematics - 10th International Conference, CICM 2017, Edinburgh, UK, July 17-21, 2017, Proceedings*, volume 10383 of *Lecture Notes in Computer Science*, pages 292–302. Springer, 2017. ISBN 978-3-319-62074-9. doi:10.1007/978-3-319-62075-6_20.

[12] Jan Jakubuv and Josef Urban. Hammering mizar by learning clause guidance (short paper). In John Harrison, John O'Leary, and Andrew Tolmach, editors, *10th International Conference on Interactive Theorem Proving, ITP 2019, September 9-12, 2019, Portland, OR, USA*, volume 141 of *LIPIcs*, pages 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2019. ISBN 978-3-95977-122-1. doi:10.4230/LIPIcs.ITP.2019.34.

[13] Jan Jakubuv, Karel Chvalovský, Miroslav Olsák, Bartosz Piotrowski, Martin Suda, and Josef Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In Nicolas Peltier and Viorica Sofronie-Stokkermans, editors, *Automated Reasoning - 10th International Joint Conference, IJCAR 2020, Paris, France, July 1-4, 2020, Proceedings, Part II*, volume 12167 of *Lecture Notes in Computer Science*, pages 448–463. Springer, 2020. ISBN 978-3-030-51053-4. doi:10.1007/978-3-030-51054-1_29.

[14] Cezary Kaliszyk and Josef Urban. Mizar 40 for mizar 40. *J. Autom. Reason.*, 55(3): 245–256, 2015. doi:10.1007/s10817-015-9330-8.

[15] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *lncs*, pages 1–35. Springer, 2013. ISBN 978-3-642-39798-1. doi:10.1007/978-3-642-39799-8_1.

[16] André Platzer and Geoff Sutcliffe, editors. *Automated Deduction - CADE 28 - 28th International Conference on Automated Deduction, Virtual Event, July 12-15, 2021, Proceedings*, volume 12699 of *Lecture Notes in Computer Science*, 2021. Springer. ISBN 978-3-030-79875-8. doi:10.1007/978-3-030-79876-5.

[17] Giles Reger, Martin Suda, and Andrei Voronkov. New techniques in clausal form generation. In Christoph Benzmüller, Geoff Sutcliffe, and Raúl Rojas, editors, *GCAI 2016. 2nd Global Conference on Artificial Intelligence, September 19 - October 2, 2016, Berlin, Germany*, volume 41 of *EPiC Series in Computing*, pages 11–23. EasyChair, 2016. doi:10.29007/dzfz.

[18] Alexandre Riazanov and Andrei Voronkov. Limited resource strategy in resolution theorem proving. *J. Symb. Comput.*, 36(1-2):101–115, 2003. doi:10.1016/S0747-7171(03)00040-3.

[19] John Alan Robinson and Andrei Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001. ISBN 0-444-50813-9. URL https://www.sciencedirect.com/book/9780444508133/handbook-of-automated-reasoning.

[20] Michael Sejr Schlichtkrull, Thomas N. Kipf, Peter Bloem, Rianne van den Berg, Ivan Titov, and Max Welling. Modeling relational data with graph convolutional networks. In Aldo Gangemi, Roberto Navigli, Maria-Esther Vidal, Pascal Hitzler, Raphaël Troncy, Laura Hollink, Anna Tordai, and Mehwish Alam, editors, *The Semantic Web - 15th International Conference, ESWC 2018, Heraklion, Crete, Greece, June 3-7, 2018, Proceedings*, volume 10843 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2018. ISBN 978-3-319-93416-7. doi:10.1007/978-3-319-93417-4_38.

[21] S. Schulz. Explanation Based Learning for Distributed Equational Deduction. Diplomarbeit in Informatik, Fachbereich Informatik, Universität Kaiserslautern, 1995. URL http://wwwlehre.dhbw-stuttgart.de/~sschulz/PAPERS/Sch95-diplom.ps.gz.

[22] Stephan Schulz. E 2.4 user manual. EasyChair preprint no. 2272, Manchester, 2020. URL https://easychair.org/publications/preprint/8dss.

[23] Stephan Schulz and Martin Möhrmann. Performance of clause selection heuristics for saturation-based theorem proving. In Nicola Olivetti and Ashish Tiwari, editors, *Automated Reasoning - 8th International Joint Conference, IJCAR 2016, Coimbra, Portugal, June 27 - July 2, 2016, Proceedings*, volume 9706 of *Lecture Notes in Computer Science*, pages 330–345. Springer, 2016. ISBN 978-3-319-40228-4. doi:10.1007/978-3-319-40229-1_23.

[24] Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In Fontaine [7], pages 495–507. ISBN 978-3-030-29435-9. doi:10.1007/978-3-030-29436-6_29.

[25] Nicholas Smallbone. Twee: An equational theorem prover. In Platzer and Sutcliffe [16], pages 602–613. ISBN 978-3-030-79875-8. doi:10.1007/978-3-030-79876-5_35.

[26] Martin Suda. Aiming for the goal with SInE. In Laura Kovács and Andrei Voronkov, editors, *Vampire 2018 and Vampire 2019. The 5th and 6th Vampire Workshops*, volume 71 of *EPiC Series in Computing*, pages 38–44. EasyChair, 2019. URL https://easychair.org/publications/paper/lZfv.

[27] Martin Suda. Improving enigma-style clause selection while learning from history. In Platzer and Sutcliffe [16], pages 543–561. ISBN 978-3-030-79875-8. doi:10.1007/978-3-030-79876-5_31.

[28] Martin Suda. Vampire with a brain is a good ITP hammer. In Boris Konev and Giles Reger, editors, *Frontiers of Combining Systems - 13th International Symposium, FroCoS 2021, Birmingham, UK, September 8-10, 2021, Proceedings*, volume 12941 of *Lecture Notes in Computer Science*, pages 192–209. Springer, 2021. ISBN 978-3-030-86204-6. doi:10.1007/978-3-030-86205-3_11.

[29] Martin Suda. Vampire getting noisy: Will random bits help conquer chaos? (system description). In Jasmin Blanchette, Laura Kovács, and Dirk Pattinson, editors, *Automated Reasoning - 11th International Joint Conference, IJCAR 2022, Haifa, Israel, August 8-10, 2022, Proceedings*, volume 13385 of *Lecture Notes in Computer Science*, pages 659–667. Springer, 2022. ISBN 978-3-031-10768-9. doi:10.1007/978-3-031-10769-6_38.

[30] Geoff Sutcliffe. The TPTP problem library and associated infrastructure. *Journal of Automated Reasoning*, 59(4), December 2017. ISSN 0168-7433. doi:10.1007/s10817-017-9407-7.

[31] Andrei Voronkov. AVATAR: the architecture for first-order theorem provers. In Armin Biere and Roderick Bloem, editors, *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, volume 8559 of *Lecture Notes in Computer Science*, pages 696–710. Springer, 2014. ISBN 978-3-319-08866-2. doi:10.1007/978-3-319-08867-9_46.