

Czech Technical University in Prague

Faculty of Electrical Engineering
Department of Computer Science



Learning Inference Guidance in Automated Theorem Proving

Doctoral Thesis

Ing. Zarathustra Amadeus Goertzel

Prague, June 2023

Ph.D. Programme: Electrical Engineering and Information Technology
Branch of study: Artificial Intelligence and Biocybernetics

Supervisor: Mgr. Josef Urban, Ph.D.
Co-supervisor: Mgr. Jan Jakubův, Ph.D.

Abstract

Automated theorem proving is a subfield of Artificial Intelligence (AI) that goes back to its origins with Simon and Newell’s Logic Theory Machine. Symbolic AI has often been contrasted with statistical machine learning (ML). This thesis addresses the task of integrating statistical ML into symbolic AI in the form of automated theorem provers, specifically the E prover. E performs a refutation-complete search for a proof of a given conjecture. ENIGMA is a system that uses ML (linear regression, gradient-boosted decision trees, and graph neural networks) to select which clause to process next in a proof search via weight functions. The ML models weigh, prioritize, and filter the clauses to be used by E, thus guiding the proof search.

Three methods are developed and experimented with: ProofWatch, ENIGMAWatch, and Parental Guidance. The six papers on these methods are included. ProofWatch uses hints from many other proofs to guide the current proof search by using proof vectors that count how much of each proof on the watchlist is matched. ENIGMAWatch combines ENIGMA with ProofWatch by using the evolving proof vectors as a description of the proof state to be used by the ML systems. Parental Guidance filters which clauses get generated by their parental features to simplify the decision landscape for clause selection ENIGMA models. The Parental Guidance method can be combined with other ENIGMA models to create the 3-phase ENIGMA system, the strongest so far. The “Make E Smart Again” short paper disables two features essential for an effective and complete refutational proof search, term ordering and literal selection, to see how self-sufficient the ML-based clause selection models are. The results in the “The Isabelle ENIGMA” paper demonstrate that ENIGMA and Parental Guidance are effective on the Isabelle Sledgehammer Problems.

All of the methods in this thesis improve upon the state-of-the-art performance over the Mizar Mathematical Library at the time of publication. The best single strategy with ProofWatch performed 26.5% better than the baseline strategy, and the best ensemble of five strategies performed 7% better than the best ensemble of baseline strategies. ENIGMAWatch proved 8.8% more than ENIGMA in the first training loop with a faster training time. The experiments in “Make E Smart Again” indicated that ENIGMA attains a 256% increase over a bare-bones version of E and can surpass the performance of E with two strong strategies. Parental Guidance improves the performance of standalone ENIGMA by 11.7%. The 3-phase ENIGMA improves upon the best previous result by 17.4% and upon E’s auto-schedule by 60%. The ENIGMA clause selection and Parental Guidance methods transfer to one of the largest corpora of Isabelle Sledgehammer problems and perform on par with the state-of-the-art prover CVC5. ENIGMA and CVC5 prove a significant number of distinct problems, with ENIGMA outperforming E’s auto-schedule by 25.5%.

Keywords: Automated Theorem Proving, Automated Reasoning, Machine Learning, Proof Search Guidance, Mizar Mathematical Library, Isabelle

Abstrakt

Automatické dokazování vět je oborem umělé inteligence (AI), který sahá až k počátkům AI, například v “Logic Theory Machine” od Simona a Newella. Symbolická umělá inteligence je často srovnávána se statistickým strojovým učením (ML). Tato práce se zabývá úkolem integrace statistického ML do symbolické AI v kontextu automatických dokazovačů vět, konkrétně dokazovače E. E provádí úplné vyhledávání důkazu dané domněnky. ENIGMA je systém, který používá ML (lineární regrese, rozhodovací stromy s gradientem a grafové neuronové sítě) k výběru, kterou klauzuli zpracovat jako další během hledání důkazu. Modely ML zvažují, upřednostňují a filtrují klauzule, které má E použít, a řídí tak průběh vyhledání důkazu.

Byly vyvinuty a experimentálně vyhodnoceny tři metody: ProofWatch, ENIGMAWatch a Parental Guidance. Práce zahrnuje šest článků o těchto metodách. ProofWatch využívá informace z mnoha předchozích důkazů k řízení aktuálního vyhledání pomocí vektorů, které vyjadřují, jak moc se současné prohledávání shoduje s předchozími. ENIGMAWatch kombinuje systémy ENIGMA a ProofWatch pomocí dynamických důkazových vektorů popisujících stav důkazu, které jsou využity systémy ML. Parental Guidance filtruje, které klauzule jsou generovány pomocí vlastností rodičovských klauzulí, aby se zjednodušilo rozhodovací prostředí ENIGMA modelů pro výběr klauzulí. Metodu Parental Guidance lze kombinovat s dalšími modely ENIGMA a vytvořit tak třífázový systém ENIGMA, zatím nejsilnější v praxi. Krátký dokument “Make E Smart Again” vypíná dvě funkce nezbytné pro efektivní a úplné vyhledávání důkazů, konkrétně uspořádání termů a výběr literálů, aby bylo vidět, jak soběstačné jsou modely výběru klauzulí založené na ML. Výsledky v dokumentu „The Isabelle ENIGMA“ ukazují, že ENIGMA a Parental Guidance jsou účinné na problémy ze systému Isabelle Sledgehammer.

Všechny metody v této práci vylepšují nejmodernější výkon na matematické knihovně Mizar v době publikace. Nejlepší jednotlivá strategie s ProofWatch má o 26,5% lepší výsledky než základní strategie a nejlepší soubor pěti strategií má o 7% lepší výsledky než nejlepší soubor základních strategií. ENIGMAWatch dokázal o 8,8% více než ENIGMA v první tréninkové smyčce a dosahuje rychlejších tréninkových časů. Experimenty v “Make E Smart Again” ukázaly, že ENIGMA dosahuje 256% nárůstu oproti holé verzi E a může překonat výkon E pomocí dvou silných strategií. Parental Guidance zlepšuje výkon samostatné ENIGMY o 11,7%. Třífázová ENIGMA zlepšuje nejlepší předchozí výsledek o 17,4% a automatický mód E o 60%. Metody výběru klauzulí ENIGMA a Parental Guidance jsou přenositelné na jeden z největších korpusů matematických problémů Isabelle Sledgehammer a fungují na stejné úrovni jako nejmodernější dokazovač CVC5. ENIGMA a CVC5 prokázaly značný počet odlišných problémů, přičemž ENIGMA překonala automatický mód E o 25,5%.

Klíčová-slova: Automatizované dokazování teorémů, Automatizované uvažování, Strojové učení, Návod na hledání důkazů, Mizar Mathematical Library, Isabelle

Acknowledgements

I would like to express gratitude to everyone and every part of the universe that has supported the research and writing of this thesis. My supervisor, Josef Urban, provided invaluable guidance in formulating bitesize research topics that not only advance the field but can be implemented and tested in a reasonable time frame for publication. His skill at writing and editing significantly improved the quality of the papers published. I am thankful to my co-supervisor, Jan Jakubův, for his cooperation in many experimental runs and his crucial work on ENIGMA’s implementation in E, without which my own implementations could not have been done. My supervisors also provided valuable feedback on the thesis, even as my comprehensive nature led to the intro materials being longer than expected. Stephan Schulz’s implementation of the watchlist features in E served as the starting point for my research. I must also express my gratitude to Bob Veroff, whose use of hints in Prover9 on the AIM problem inspired us to aim to automate his role.

I extend gratitude to my colleagues at the automated research group in Prague for the supportive environment. I also appreciate the assistance of certain administrative staff at the Czech Technical University in Prague, who helped me navigate the bureaucracy. My work has been financially supported by the *AI4REASON* ERC Consolidator grant number 649043, the Czech project *AI&Reasoning* CZ.02.1.01/0.0/0.0/15 003/0000466, the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902, the European Regional Development Fund, a CTU tuition waiver, and, for a brief period, a CTU stipend offered to doctoral students. Without the Mizar Mathematical Library and Isabelle Sledgehammer problems to test our systems on, developing machine learning models for automated reasoning would be much more difficult. The open-source software community allows us to research and use machine learning toolkits without much concern about ownership, which I’m sure has accelerated the progress of the field.

I am deeply grateful to my loved ones, friends, and family for their emotional support and encouragement throughout this extended process. The city of Prague and its inhabitants provided a bountiful environment and culture in which to enjoy life while researching. Special mention goes to my father, Ben Goertzel, who let me know about the Ph.D. position and engaged in numerous discussions on research topics and related fields. He also onboarded me to life on Earth with ample exposure to math and the sciences.

Contents

Contents	0
1 Introduction	1
2 Thesis Structure	4
3 Foundational Material	5
3.1 Logic	5
3.2 Resolution and Superposition: In Search of Saturation	11
3.3 Term Ordering and Literal Selection	13
3.4 Saturation-Based ATPs: The Given Clause Loop	14
3.5 ATP Search Strategies in E	14
3.6 Strategy Invention and Selection	15
3.7 Watchlists	16
3.8 Datasets	16
3.9 ENIGMA: Machine Learning for Given Clause Selection	18
3.10 ENIGMA: Parental Guidance	21
4 Related Research	22
4.1 ENIGMA: Leapfrogging and Reasoning Components	22
4.2 Premise Selection	22
4.3 Hammers	23
4.4 Tactical Theorem Provers	24
4.5 Machine Learning for the leanCoP Family	25
4.6 Automated Formalization and Neural Theorem Provers	26
5 Contributed Research Papers	29
5.1 ProofWatch	31
5.2 ENIGMAWatch: Proof of Concept	50
5.3 ENIGMAWatch	58
5.4 Make E Smart Again	73
5.5 Parental Guidance	81
5.6 The Isabelle ENIGMA	100
6 Conclusion and Contributions	121
6.1 Contributions	121
6.2 Concluding Remarks	122
A Author Publications	124
A.1 Citations of Author’s Publications	126
References	133

1 Introduction

Combinations of machine learning and automated reasoning are studied with increasing interest in the AI community today. This thesis documents one path from the era when machine learning could not be effectively applied to guide automated theorem provers to the current era when the application of machine learning to theorem proving is becoming common.

I come from the philosophical background of the artificial general intelligence (AGI) field, which aims to develop AI systems capable of effectively operating in diverse environments. One approach in the field of general AI is to work with universal search procedures that are guaranteed to find a program solving a problem if one exists (in exponential time relative to the length of the solution program). The historically famous one is called Levin search [159, 160]. The automated theorem provers I work with use refutation-complete search procedures such that if there is a proof by contradiction to a given conjecture, this proof will be found (if one is sufficiently patient). Program synthesis and proof search are both semi-decidable because there exist terminating search procedures in the case that solutions exist, and the search procedures may not terminate when solutions do not exist. There are some well-known theoretical schemata for creating an “optimally” generally intelligent reinforcement learning system on top of universal search. One of these is the Gödel Machine [218] by Schmidhuber, which starts with a simple universal search procedure while simultaneously searching for provably superior search procedures. Hutter’s AIXI [127] is a reinforcement learning agent that takes the action that maximizes the reward over all computable environments matching the observed history weighted by the complexity of the program generating the environment. Neither of these is practical. So this approach to general AI is to work on approximations of these theoretical ideals. One proof of concept in this research direction is the Monte-Carlo Tree Search AIXI approximation that learned to play Pac-Man [245]. In this light, I view work on guiding automated theorem provers as one approach to learning how to guide a universal search procedure effectively.

Automated reasoning is one approach to general problem-solving because mathematics is the language in which problems are specified. Thus the capacity to solve mathematical problems covers problem-solving in any domain. Quaife famously argued for this point in his Ph.D. dissertation [201]. I would like to honor and affirm Quaife’s will to devote work in automated reasoning “to the pursuit and realization of unlimited pleasure”.

The field of automated reasoning [212] is broadly split into Automated Theorem Provers (ATPs) and Interactive Theorem Provers (ITPs). ATPs are fully-automated in that they prove conjectures from premises entirely on their own. ITP systems are also called proof assistants and aim to support mathematicians in formalizing their mathematical theories. Until recently, the ATPs have tended to use first-order logic (FOL) because it is easier to devise search procedures for first-order logic, and the ITPs have used type theory, higher-order logic (HOL), or set theory. The recent large datasets for developing ATPs usually come from the ITP system libraries translated into a first-order

logic format. Furthermore, the ATPs can be integrated into ITP systems as *hammers* [37] to help discharge goals, which makes life easier for formal mathematicians.

Two major mathematical successes for ITP systems are the Flyspeck project [111] (proving Kepler’s conjecture) and the Four-Color theorem [101]. Mathematicians had difficulty ascertaining the validity of the proofs of the Four-Color theorem due to their length and use of computer programs, which could be buggy and thus also need to be verified.

The Kepler conjecture states that the densest packings of equally sized spheres in a three-dimensional Euclidean space are the cubic and hexagonal close packing arrangements. In simple terms, the standard way of packing oranges in a box is optimal. Hales announced a proof in 1998 that included 3 gigabytes of data detailing solutions to linear programming problems for various configurations of spheres. In 2005, the *Annals of Mathematics* published the proof with “99% certainty” that it is correct [110]. The Flyspeck project developed a formal proof in the HOL Light proof assistant [115] with support from the Isabelle proof assistant [185] to engender higher confidence, which was completed in 2014 and accepted in 2017 [109, 112].

The Four-Color theorem states that any map can be colored with four colors so that no two adjacent regions have the same color. There is a long history of attempted proofs of the Four-Color theorem, most notably Kempe’s in 1879, which have been found to be flawed [253]. In 1976 Appel and Haken solved the problem by a reduction to 1,834 (and later 1,482) configurations that were checked by computer programs [12, 13]. The ‘computer proof’ triggered philosophical discussions as to whether proofs not checkable by hand are valid, and the proof was regarded with suspicion. In 1994, Robertson et al. [211] published a proof using a quadratic-time algorithm that only required checking 644 reducible configurations. Finally, in 2005, Gonthier completed a formal proof [103] in the Coq proof assistant [59].

The QED manifesto [1] proposes to formalize all mathematical knowledge in the same manner as the Kepler conjecture and the four-color theorem. I believe that the future of math involves full formalization. Wiedijk keeps track of the list of the “top 100” mathematical theorems and which ITP systems they have been formalized in,¹ including Gödel’s Incompleteness Theorem [191], the Pythagorean Theorem [155], the Odd Order theorem [102], and the forcing technique to show the independence of the Continuum Hypothesis [114]. At present, 99 have been formalized across all ITP systems, and the top two, HOL Light and Isabelle, contain formalizations of 87 of the theorems. The remaining theorem is Fermat’s Last Theorem,² which will require a vast amount of background theory to be formalized first.

There are many challenges to the goal of formalizing all mathematical knowledge. The formal languages for proof assistants still need to be made easier for humans to learn, and users still need to choose from among the many libraries

¹<https://www.cs.ru.nl/~freek/100/>

²There do not exist positive integers a , b , and c such that $a^n + b^n = c^n$ for an integer $n \geq 2$.

available. The Dedukti team is developing the Dedukti proof assistant [15] to help reduce the need to choose the right proof assistant. The goal is for Dedukti to provide a universal logical language and framework in which one can express the theories corresponding to the logical systems used by the proof assistants and work with translations between them [38, 68]. The Naproche project [60] (Natural language Proof Checking) aims to develop a controlled natural language in which mathematical statements can be syntactically checked for correctness, hopefully making formalization easier for humans. Effective proof automation is needed to allow mathematicians to gloss over technical minutiae. The *MPTP* (Mizar Problems for Theorem Proving) system [237] for the Mizar Mathematical Library [139] and the *Sledgehammer* system [39] for Isabelle were among the first to demonstrate that ATPs can help with the formalization of mathematics in ITPs.

The contemporary automated theorem proving ecosystem consists of various interacting components, each of which is open to improvement by algorithmic innovation and the application of machine learning. The automated theorem prover's search process will eventually find a proof by contradiction for valid conjectures, provided the necessary *premises* are *selected* from among the available background theory. In order to accelerate the proof search, it helps to provide no more premises than necessary and to choose *strategies* with a higher likelihood of *selecting* the right *clauses* sooner than later. My research focuses on developing learning-based guidance of the search process for the E theorem prover [222, 223, 248].

I wrote a series of nine blog posts covering the content of this thesis and some additional research for an educated lay reader, which may introduce the field more gently.³

³The blog posts can be found at <https://gardenofminds.art/category/research/>

2 Thesis Structure

This thesis aims to develop and test novel methods for the learning-based guidance of automated theorem provers, particularly for the E theorem prover. The methods focus on integrating additional semantic information into the learning systems and increasing the integration of learning-based guidance into the search process. The hypothesis is that machine learning can be applied to each choice point that is not algorithmically solved.

The thesis consists of an introductory text and six research publications. The rest of the thesis is organized as follows. Section 3 provides an overview of the field background materials. Section 4 discusses related research topics. Section 5 contains the six research publications of the thesis. The following lists the papers and provides their brief overview.

- “ProofWatch”: guiding the E theorem prover’s search with previously completed proofs (Paper 1 [89]).
- “ENIGMAWatch: First Experiments”: a proof of concept result combining ProofWatch with the ENIGMA machine learning system’s guidance of E’s proof search (Short paper 1 [93]).
- “ENIGMAWatch”: full results of the system combining statistical learning (ENIGMA) and symbolic learning (ProofWatch) based guidance of E (Paper 2 [91]).
- “Make E Smart Again”: ablation studies demonstrating that the ENIGMA machine learning can function without term ordering, literal selection, or strong strategies (Short paper 2 [96]).
- “Parental Guidance”: a new system to filter generated clauses based on their parent clauses and the 2 and 3-phase ENIGMA systems combining Parental Guidance with ENIGMA guidance (Paper 3 [97]).
- “The Isabelle ENIGMA”: the application of a 2-phase ENIGMA with Parental Guidance to Isabelle Sledgehammer problems (Paper 4 [98]).

Section 5 begins with more detailed descriptions, followed by the unmodified published papers. Section 6 contains the concluding remarks and a discussion of the authors’ specific contributions to the research publications.

3 Foundational Material

This section provides a brief overview of the automated theorem proving field leading up to the research in this thesis. The individual papers contain additional overviews of the field. The following section covers related research in the application of machine learning to automated theorem proving. Statistical machine learning and symbolic learning, including theorem proving, have classically been viewed as opposing approaches to artificial intelligence. Their romantic reunion is the main topic of this thesis.

For an additional high-level overview of approaches to learning from previous proof experience in 1999, see [65]. The reader may also be interested in checking out the article by Harrison, Urban, and Wiedijk covering the history of interactive theorem proving up to 2014 [116]. The Handbook of Automated Reasoning [212] contains in-depth coverage of many topics in the field.

3.1 Logic

An overview of automated theorem proving (ATP) must begin with the notion of a formal proof: the notion that a theorem statement can be justified via sequences of inferences from known statements. A *proof* is a directed acyclic graph where the nodes are statements, and the edges are logical inferences connecting them. The root node is the theorem to be proven, the leaves are axioms assumed to be true, and the intermediary nodes are derived statements. Some proof theorists like to view a proof as a finite sequence of statements, each of which is an axiom or a statement derived from previous statements via a logical inference rule. An automated theorem prover is given a target conjecture along with some theory axioms and asked to search for a proof. The proof should be easily verifiable by checking each inference rule’s correctness.

Logic is the field that studies which inference rules work in which formal languages. More generally, logic is the study of rational thought and coherent chains of reasoning: logic deals with determining when components fit together coherently, whether conclusions follow from premises, and how to identify contradictions and fallacies of reasoning. In essence, how do we know when a proof or argument is correct?

The study of logic usually begins with propositional logic [10, Chapter 1], also called zeroth-order logic. Classical propositional logic consists of *propositional formulas*⁴ that represent logically true or false statements. Logical operators are used to inductively create *compound propositions* from *atomic propositions*. Atomic propositions consist of *propositional variables* that represent assertions of logical truth or falsity. The standard *logical operators* are \wedge (“and”), \vee (“or”), \neg (“not”), \rightarrow (“implication”), and \leftrightarrow (“equivalence”). The first check for the

⁴Well-formed formulas of propositional logic can simply be called *propositions*. The term *statement* can also be used synonymously. Sometimes philosophers wish to draw a distinction between the syntactic statement and the abstract entity that is stated. For further discussion, see Chalmers [50, Third Excursus] or the Stanford Encyclopedia of Philosophy [171].

correctness of a proof is to ensure that all the statements are syntactically *well-formed formulas* according to the formal language.

An *interpretation* assigns meaning to the symbols of a formal language. The standard truth-value semantics stipulates that propositions are either true or false, which can be represented with the symbols \top (“truth”) and \perp (“falsity”). Thus interpretations are boolean functions that bestow truth values upon propositional variables. The logical connectives have standard interpretations such that the truth of compound propositions can be recursively checked, e.g., $(P \wedge Q) \rightarrow R$ evaluates to true if R is true or if $P \wedge Q$ is false, which is the case if at least one of P and Q is false.

To apply propositional logic to natural language statements, one assigns each statement to a proposition variable and specifies the logical relations via logical operators. For example, consider the classic example of modus ponens, an inference rule stating that: “if P is true and P implies Q , then Q is true”, which is written, $P, (P \rightarrow Q) \vdash Q$. Let P represent “Socrates is a man” and Q represent “Socrates is an animal”. Then, $P \rightarrow Q$ represents the implication, “If Socrates is a man, then Socrates is an animal”. It is common knowledge that Socrates is a man and that humans are technically animals; thus, we can consider P and $P \rightarrow Q$ to be axioms, which results in the conclusion that Q is true: “Socrates is an animal”.⁵

If there exists an interpretation assigning truth values to the variables such that a formula is true, then one says that the formula is *satisfiable*. In this case, the interpretation is said to be a *model* of the formula. For example, $P \rightarrow Q$ is satisfiable (with $P := \perp$ and $Q := \top$), but $P \wedge \neg P$ is not satisfiable for any interpretation of the variables. A logical *validity* or *tautology* is a formula that is true no matter what the truth values of its variables are. The principle of excluded middle, $P \vee \neg P$, is a valid proposition under the classic truth-value semantics.

Deciding the satisfiability of a propositional formula is, in general, an NP-complete problem, and the field of study is called SAT [24]. The problem can be solved by creating a truth table containing a row for each of the 2^n interpretations where n is the number of variables. A popular approach used in many SAT solvers is the DPLL algorithm [63] [24, Chapter 3.5], which is a complete backtracking-based algorithm for solving the propositional satisfiability problem. Modern SAT solvers often use the conflict-driven clause learning (CDCL) algorithm [24, 27, 167, 168], which extends the DPLL algorithm with clause learning and non-chronological backtracking.

First-order logic [10, 70, 174] allows one to employ predicates and to quantify over variables with respect to a universe of discourse, which is usually a set denoted U . Predicates allow one to reason about properties of entities and relations among entities. One view is that *predicates* are symbols that are intended to be interpreted as functions from the universe to propositions, which are true or false. Predicates are usually modeled as relations,⁶ and predicates with

⁵Please forgive the use of present tense for simplicity. Socrates was a man.

⁶I will underline predicate names to denote the relation interpreting the predicate.

no arguments are the same as propositional variables. For example, let $\text{Man}(\cdot)$ be a predicate of arity 1 that can be interpreted as the set of all men, so $\underline{\text{Man}}(x)$ will be true if the variable x is a man. *Quantifiers* are operators that bind variables to a context. Variables that a quantifier has not *bound* are called *free variables*. A statement with no free variables is also called a *sentence* and is similar to a proposition. Given a predicate $P(\cdot)$, the *existential quantification*, $\exists_x P(x)$, is intended to be true if and only if there is an existential witness a in U such that $\underline{P}(a)$ is true. For example, if $\underline{\text{Man}}(\text{socrates})$ is true, then socrates is an existential witness for $\exists_x \text{Man}(x)$. Given a predicate $P(\cdot)$, the *universal quantification*, $\forall_x P(x)$, is intended to be true if and only if $\underline{P}(a)$ is true for every element a in U . For example, instead of writing “Socrates is a man” \rightarrow “Socrates is an animal” as in propositional logic, one can use the universal quantifier to say that if an entity x in U is a man, then x is an animal: $\forall_x (\text{Man}(x) \rightarrow \text{Animal}(x))$.

The addition of a universe of discourse allows the introduction of *function* symbols that are intended to be interpreted as n -ary functions from the universes of discourse to itself, which includes constants when n is zero. *Terms* are inductively defined to be variables or functions whose arguments are terms. The equality relation, $=$, applied to terms, is often included in the definition of the language. The *atomic formulas* consist of the equality and predicate symbols applied to the appropriate number of terms. *Well-formed formulas* are inductively defined to be atomic formulas, quantifiers over well-formed formulas, and compound formulas made up of propositional logical connectives applied to well-formed formulas.⁷

Adding universes, function symbols, and predicate symbols to the logic requires interpretations of first-order logic languages to exhibit additional structure. A *structure* \mathcal{M} consists of a universe, a signature, and an interpretation. The contemporary standard is to use set-theoretic semantics in which the universes⁸ are sets. First, the *signature* of the language specifies the constant, function, and predicate symbols. The number of arguments, the arity, of function and predicate symbols is also specified. The rest of the language is similar to the language of propositional logic: logical symbols, the equality relation, and a potentially infinite set of variable symbols. An interpretation assigns to each constant symbol an element of the universe U . The interpretation associates every n -ary function symbol with a function from U^n to U and every n -ary predicate symbol with an n -ary relation on U , that is, with a subset of U^n . This interpretation completes the structure and determines the truth value of arbitrary sentences.

To determine the truth value of formulas with free variables, one needs a *variable assignment* μ associating each variable with an element of the universe. A formula F is *satisfiable* in \mathcal{M} if there is a variable assignment μ such that F is true. A formula F is *valid* in \mathcal{M} if F is true under all variable assignments and one writes $\mathcal{M} \models F$. And a formula F is *logically valid* if it is true in every interpretation. A theory T usually refers to a set of sentences of a particular

⁷It's common to use the term *formulas* to refer to *well-formed formulas* for convenience.

⁸The universe of discourse is also called the *domain* of discourse.

signature, such as the axioms of Peano arithmetic for the natural numbers. A structure \mathcal{M} is a *model* of a theory T if it satisfies all of the sentences in T .

First-order logic admits many proof systems, some of which are sound and semantically complete [70, Chapter 5]. There are two logical consequence relations. Let Γ be a set of formulas denoting the context. The *syntactic consequence* relation denotes *provability* in a given formal proof system FS , and one writes $\Gamma \vdash_{FS} F$ to denote that the formula F is provable from Γ . The *semantic consequence* relation does not depend on the formal proof system, and one writes $\Gamma \models F$ to denote that every interpretation that models Γ also models F . When Γ is empty, $\vdash_{FS} F$ denotes that F is a tautology in FS , and $\models F$ denotes that F is a logical validity. A formal proof system FS is *sound* (or *correct*) if provability implies semantic consequence: for all Γ and F , if $\Gamma \vdash_{FS} F$, then $\Gamma \models F$. A formal proof system is *semantically complete* if semantic consequence implies provability: for all Γ and F , if $\Gamma \models F$, then $\Gamma \vdash_{FS} F$. Gödel's completeness theorem [85] demonstrated semantic consequence for a particular first-order proof system, a Hilbert-Ackermann proof system [120]. The combination of soundness and semantic completeness means that a proof system's notions of provability and semantic truth are aligned.

First-order logic has many other pleasant properties. Semantic completeness is a proof-theoretic notion that is intimately related to the compactness property [70, Chapter 6], which states that a set of first-order sentences has a model if and only if every finite subset has a model. First-order logic admits proof systems that allow for the computable enumeration of all logical consequences of a computably enumerable set of axioms. Thus logical validity is semi-decidable, similar to the halting problem in computer science: if a sentence is valid (given a set of axioms), then there is a semi-decision procedure that will determine this. However, if the sentence is invalid, the procedure may never terminate. First-order satisfiability is fully undecidable: there does not exist a computable procedure that will, in general, determine whether, for a given formula in a first-order language, there exists an interpretation under which it is True.

First-order logic can not, however, uniquely describe structures with infinite domains (such as the natural numbers or the real continuum): the Löwenheim-Skolem theorem [70] shows that if a first-order theory has a model of an infinite cardinality, then it has models of every infinite cardinality greater than or equal to the cardinality of its signature. Lindström's theorem [70] states that first-order logic is the strongest logical system satisfying certain properties, including the compactness theorem and the Löwenheim-Skolem theorem.

Classification: Clausal Normal Form

In the ATP and SAT fields, problems are often transformed into the *clausal normal form* (CNF), also called the *conjunctive normal form*. Atomic formulas and their negations are called *literals*, and a *clause* is a disjunction of literals, e.g., $\text{Man}(x) \vee \neg \text{Ape}(x)$. The conjunctive normal form is when a formula consists of a conjunction of clauses, e.g., $(\neg \text{Man}(x) \vee \text{Ape}(x)) \wedge (\neg \text{Ape}(\text{child}(x)) \vee \text{Animal}(\text{child}(x)))$. Automated theorem provers usually aim to do proofs by

contradiction, which means that to succeed, the ATP must show that a set of sentences is unsatisfiable. Thus, for theory T and conjecture C , one transforms the problem $T \vdash C$ into $T \wedge \neg C \vdash \perp$. The clausification process is applied to the theory and negated conjecture to produce a *refutationally equivalent* set of clauses: a refutation to the clause set is derivable if and only if the original negated conjecture can be disproved.

The clausification process typically first converts formulas into negation normal form by removing implications and moving negations inward so that the negation operator is only applied to atomic formulas, and the only logical operators are conjunction and disjunction. For propositional logic and SAT problems, one only needs to distribute the \forall s over the \wedge s to create a clause. The size of a formula in CNF can blow up exponentially, so there are many techniques for more effective clausification. Clauses are often represented as multi-sets of literals, which means a formula in CNF can be represented as a multi-set of multi-sets.

For first-order problems, one needs to deal with quantified variables. From negation normal form, the next step is to standardize variable names to avoid conflicts and to move quantifiers to the outermost scope. Then the process of *Skolemization* replaces existential quantifiers with *Skolem symbols* (constants or functions) that provide the existential witness. This step preserves unsatisfiability because if there can be no interpretation of the Skolem symbol that provides the right element, then there can be no existential witness for the original formula. Now that existential quantifiers have been lifted to the level of the language and all universal quantifiers have been moved to the outermost scope, one can assume that all variables are implicitly universally quantified and drop the universal quantifiers. Finally, the \forall s are distributed over the \wedge s to create first-order clauses.

3.1.1 The Logic Zoo

Some other logical specimens are worthy of a brief mention. This thesis primarily works with classical first-order logic. In The Isabelle ENIGMA paper, many-sorted first-order logic is successfully used. The author hopes that the machine learning techniques developed in this thesis can also be effectively applied to other logical systems.

1. In *many-sorted first-order logic* [70,166], one can have several universes of discourse over which variables can be quantified. For sorts $\{s_1, \dots, s_n\}$, a many sorted language has distinct variable sets V_1, \dots, V_n , and a many-sorted structure has universes $\{U_1, \dots, U_n\}$. The sorts of the arguments of predicate and function symbols can be specified by a rank function. The scope of quantification also needs to be specified so that variables only range over the appropriate sorts. For example, let s_i denote the sort of animals in “not all animals have a father”: $\neg \forall_{x:s_i} \exists_{y:s_i} \text{Father}^{(s_i,s_i)}(x, y)$.
2. In *Second-order logic* [70,249], one can instantiate variables with predicates instead of only elements of the universe. Quantifying over predicates is

equivalent to quantifying over subsets of elements of the universe because predicates are represented as relations. Second-order logic with the standard semantics can uniquely describe structures with infinite domains and does not provide a deductive system that is sound, complete, and with a computable proof-checking algorithm.

3. *Simple type theory* [10, 76] is a standard form of *higher-order logic* that is built in terms of function abstraction and application. The base types are the *type of individuals*, ι , and the *type of truth values*, $*$. The *function types*, $(\alpha \rightarrow \beta)$ for types α and β , are defined inductively. Predicates and sets can be defined as types $(\iota \rightarrow *)$, and the usual logical connectives and quantifiers are defined in terms of function application, function abstraction, and equality. Quantified variables are restricted by type, which includes higher-order types that correspond to “sets of sets of sets” and so on.

The set-theoretic model theory works similarly to how it does in first-order logic. Let D_α be the set of all values of type α . The set D_ι would correspond with the first-order universe U . *Standard models* require function domains, $D_{\alpha \rightarrow \beta}$, to contain all total functions from D_α to D_β . *General models* only require function domains to contain some nonempty set of total functions from D_α to D_β . When using standard models, one can uniquely describe structures with infinite domains, such as the natural numbers, and there is no sound and complete proof system. When using general models, simple type theory is semantically equivalent to first-order logic due to the compactness and Löwenheim-Skolem theorems holding, and it admits a sound and complete deductive system.

4. *Intuitionistic* or *constructive logic* [119, 180] is like classical logic without the principle of excluded middle or the double-negation rule (that one can prove A from $\neg\neg A$). The Coq proof assistant uses the calculus of constructions, an intuitionistic logic.

A noteworthy family of results is the double negation translations: (1) Glivenkos’s translation is such that a proposition P is provable in classic propositional logic if and only if $\neg\neg P$ is provable in intuitionistic logic. (2) Kuroda’s negative translation is such that a first-order formula F is classically provable if and only if $\neg\neg F'$ is provable in intuitionistic logic where F' is obtained from F by adding double negations beneath each universal quantifier [43].

5. *Paraconsistent logic* systems [47, 199] aim to develop non-consistency tolerant deductive systems in which the principle of explosion is invalid, that is, in which one cannot prove any proposition P from a contradiction. This way, knowledge bases and reasoning can survive contradictions. One charming result due to Canielli and Fuenmayor [48] is that in some *Logics of Formal Inconsistency*, Gödel’s Incompleteness Theorem can be seen as *Gödel’s Existence Theorem*.

6. *Fuzzy logic* systems [58, 215] allow truth values to take on values within the real interval $[0, 1]$ instead of only 0 and 1 (False and True). There are many ways to interpret the fuzzy logical operators. Standard fuzzy logic uses the following operators (denoted with the subscript s): $\vee_s := \max$, $\wedge_s := \min$, and $\neg_s(x) := 1 - x$. Standard fuzzy logic reduces to classical logic in the case that all of the fuzzy truth values are 0 or 1.
7. *Probabilistic logic* systems [64] aim to extend crisp logical entailment to uncertain or probabilistic inference. Markov logic networks [210] satisfy the property that all logical validities will have probability one, which makes them a generalization of classical logic.
8. *Linear logic* systems [67] assign a cost to doing inference so that one can reason about resources.
9. *Modal logic* systems [79] allow one to reason about various modal operators such as belief, knowledge, possibility, necessity, temporality, and obligation. The semantics of modal logic systems are often understood in terms of possible worlds and the influence of the modal operator on the accessibility relation among worlds.

3.2 Resolution and Superposition: In Search of Saturation

The resolution and superposition calculi [252] were developed to facilitate algorithmic proof search, and they form the core of most modern automated theorem provers, including the E prover. They can act as the primary inference rules for *refutation complete* search processes, provided the applications are fairly ordered. When using a refutation complete search process, a proof will eventually be found if a set of first-order clauses is unsatisfiable. This section closely follows Weidenbach from the Handbook of Automated Reasoning [212, Chapter 27] and the E 2.6 user manual.⁹ All term definitions can be found in both sources.

The *resolution* rule takes two clauses that contain complementary literals (L and $\neg L$) and produces a resolvent clause that they logically imply. Below is the propositional logic version:

$$\frac{C_1 \vee L \quad \neg L \vee C_2}{C_1 \vee C_2}$$

The clausal form of modus ponens is a special case of resolution:

$$\frac{\perp \vee A \quad \neg A \vee B}{B}$$

In the first-order logic case, one needs to introduce the concept of unification to deal with the variables.

A *substitution* σ is a finite mapping from a set of variables to a set of terms. The application of substitutions extends to terms, predicates, and clauses by

⁹http://www.lehre.dhbw-stuttgart.de/~ssschulz/WORK/E_DOWNLOAD/V_2.6/eprover.pdf.

structural induction: e.g., for a function symbol f of arity n , $f(t_1, \dots, t_n)\sigma = f(t_1\sigma, \dots, t_n\sigma)$. A *unifier* for two terms s and t is a substitution σ such that $s\sigma = t\sigma$. A unifier σ is a *most general unifier* if it can be specialized via a substitution into any other unifier, that is, if, for any unifier τ of s and t , there exists a substitution λ such that $\sigma\lambda = \tau$. Substitutions are composed from left to right. For example, $C\sigma\lambda$ is the same as $(C\sigma)\lambda$. We write $\sigma = mgu(s, t)$ to denote that σ is the most general unifier of s and t . The notation $[t/x]$ denotes the substitution of the term t for the variable x .

We say that a term s *matches* a term t if there is a substitution σ such that $s\sigma = t$. A clause C_1 is said to *subsume* a clause C_2 if there exists a substitution σ such that $C_1\sigma$ matches a subset of C_2 , that is, $C_1\sigma \subseteq C_2$. We write $C_1 \sqsubseteq C_2$ to denote that C_1 subsumes C_2 . One important fact is that subsumption implies logical implication: if $C_1 \sqsubseteq C_2$, then C_1 logically implies C_2 .

For first-order resolution, let $\sigma = mgu(L_1, L_2)$:

$$\frac{C_1 \vee L_1 \quad \neg L_2 \vee C_2}{(C_1 \vee C_2)\sigma}$$

Let us revisit the inference that Socrates is an animal because all men are animals as a resolution step:

$$\frac{Man(\text{socrates}) \quad \neg Man(x) \vee Animal(x)}{Animal(\text{socrates})} \text{ [socrates}/x]$$

The clause $Man(\text{socrates})$ is called a *unit clause* because it has only one literal. A common strategy called *unit propagation* is to process all unit clauses first.

The *factoring rule* unifies literals within a clause. Let $\sigma = mgu(L_1, L_2)$:

$$\frac{C_1 \vee L_1 \vee L_2}{(C_1 \vee L_2)\sigma}$$

Superposition and paramodulation are inference rules that combine resolution with equality reasoning. Equality reasoning refers to performing term rewrites, e.g., if “masked_man = socrates” and “ $Animal(\text{masked_man})$ ”, then one can apply the equality to rewrite the literal into “ $Animal(\text{socrates})$ ”. Following the E manual, let $s[u \leftarrow t]$ denote the term s in which every occurrence of the subterm u has been replaced by t . In general, for terms s , t , and u , a rewrite with $s = t$ into a subterm p of u can take place when there is a (most general) unifier σ of p and s , that is, $p\sigma = s\sigma$:

$$\frac{s = t \quad u}{(u[p \leftarrow t])\sigma}$$

In the case of non-unit clauses, such as $s = t \vee S$, one standard interpretation is that $s = t$ is a conditional equality, that is, $\neg S \rightarrow s = t$. The *paramodulation* rule rewrites p as t , where p is a subterm of u and $\sigma = mgu(p, s)$:

$$\frac{s = t \vee C_1 \quad u \vee C_2}{(u[p \leftarrow t] \vee C_1 \vee C_2)\sigma}$$

The application of the paramodulation rule must be restricted to guarantee the completeness of the proof search. The most important component is a *term ordering* to ensure that rewriting is done in one direction, usually toward smaller terms. Term orderings are also called *reduction orderings*. The selection of literals also needs to be done in an ordered manner. *Superposition* is a term for paramodulation when its use is restricted to adhere to a complete reduction ordering.

The goal of the search is either to saturate the search space or to derive the empty clause, a refutation. A clause set is *saturated* when all inferences among the clauses from the set have been performed up to redundancy, and the resulting clauses have been included in the set. To effectively achieve this, we need *reduction rules* that reduce or simplify the number of clauses. One reduction rule is *tautology deletion*, which deletes tautologies. The rule of *subsumption deletion* removes clauses that are subsumed by others in the clause set. Recall that if C_1 subsumes C_2 , then C_1 logically implies C_2 ; thus, keeping C_2 in the clause set is redundant. If a clause set saturates without the empty clause, then the clause set is satisfiable.

The E theorem prover used in this thesis uses the superposition calculus with some additional rules to enhance performance. E uses a purely equational paradigm, which means that all literals are equations. Non-equational literals are encoded as equations or disequations, so $P(t_1, \dots, t_n)$ will be represented as $P(t_1, \dots, t_n) = \text{True}$ and $\neg P(t_1, \dots, t_n)$ will be represented as $P(t_1, \dots, t_n) \neq \text{True}$. The rules for resolution, factoring, and superposition are appropriately adapted.

3.3 Term Ordering and Literal Selection

The superposition calculus is one way to implement term rewriting based on equalities, which, if unrestricted, could loop indefinitely. The Knuth-Bendix Ordering (KBO) [148] is one of the most popular ways to restrict term rewriting and literal selection. The default in E is the KBO ordering. The KBO ordering is defined in terms of a weight function over the symbols in terms. E also supports lexicographic path ordering (LPO) [143]. *Literal selection* for resolution in the right order is also needed for completeness [17, 18] (see [121, Section 3] for an example).

Some work has been done on applying machine learning to improve the term ordering. Jakubův and Kaliszyk [134] implemented the relaxed weighted path ordering (WPO) [256] in E, which can be parametrized to behave similarly to KBO or LPO. Bártek and Suda [25] tried to use machine learning to predict symbol precedences, which did not perform better than “invfreq”, which orders symbols based on how frequently they are seen in the input problem. Reger et al. [121] achieve good results with an incomplete lookahead literal selection procedure in Vampire [153].

In the “Making E Smart Again” paper [96], I use a structural identity relation as the ordering, which limits E’s ability to do rewriting and makes E much less efficient. I show that machine learning can still learn to guide E effectively: I

attained a 260% performance increase (in terms of theorems proved on Mizar) over a simple strategy with the limited E.

3.4 Saturation-Based ATPs: The Given Clause Loop

State-of-the-art automated theorem provers (ATPs) for first-order logic (FOL), such as E [221] and Vampire [153], are based on *saturation* and employ the *given clause algorithm* with the superposition calculus. The description in this section primarily pertains to E, and while Vampire’s given clause loop is similar, there may be some differences. The input FOL problem consists of the theory and the conjecture. The first step is to negate the conjecture to search for a proof by contradiction. The theory and the negated conjecture are then transformed into a refutationally equivalent set of clauses in CNF (clausal normal form), which forms the initial clause set, $T \cup \{\neg C\}$. The goal of the proof search is to take this initial clause set and either to infer the empty clause, which represents the contradiction, or to produce a clause set that is saturated with respect to the inference rules.

The search for a contradiction is performed by maintaining sets of *processed* (P) and *unprocessed* (U) clauses (the *proof state* $\Pi = (P, U)$). The unprocessed clause set is initialized with the initial clause set, while $P = \emptyset$. The algorithm repeats the following process until a contradiction is found, U becomes empty, or a resource limit is reached:

1. **Select** a *given clause* g from U to process and add to P .
2. **Generate** clauses by performing all inferences between g and clauses in P .
3. **Simplify** the clauses, remove redundancies, and check for the empty clause.
4. **Evaluate** the simplified generated clauses and add them to U .

The given-clause algorithm is refutation complete for the superposition calculus, which means that if the initial clause set is unsatisfiable, a contradiction is derivable from the axioms and the negated conjecture, which constitutes a proof by contradiction. The processed clause set P is saturated if the unprocessed clause set U empties, which means that the initial clause set ($T \cup \{\neg C\}$) is satisfiable (and the conjecture is not a theorem). The search process may never terminate if the initial clause set is satisfiable with an infinite model.

3.5 ATP Search Strategies in E

The search space of this loop can grow exponentially, and the term ordering only restricts the number of eligible clauses for selection. It is well-known that selecting the right given clause is crucial for success. This observation is one of the results of Stephan Schulz’s Ph.D. dissertation [219]; he is the primary developer of E. Machine learning from a large number of proofs and proof searches can help guide the selection of the given clauses, and this is the approach taken in this thesis.

Given clause selection in E is managed via *clause evaluation functions* (CEFs). Clause evaluation functions consist of *priority* functions and parametrized *weight* functions that assign numeric evaluations to the clauses: (priority, weight), where the priority is an integer and the weight is a real number. The clauses are inserted into an ascending priority queue based on these (priority, weight) tuples. The conceptual idea is that the clauses are primarily selected based on weight functions, while priority functions partition the clauses into distinct subsets based on special properties.

For example, a CEF with the priority function **PreferGoals** will choose all goals (negative clauses) before selecting positive clauses. The parametrized weight function **Clauseweight** counts the symbols in a clause, and the weight function **FIFOWeight** assigns monotonically increasing weights to realize a first-in, first-out strategy.

A strategy in E is a combination of CEFs that follow a weighted round-robin scheme. For example, in the following classic strategy, five clauses from the **Clauseweight** priority queue will be selected for every one clause from the **FIFOWeight** priority queue:

```
(5*Clauseweight(PreferGoals,1,1,1),
 1*FIFOWeight(ConstPrio))
```

The classic strategy combining a FIFO weight function with a symbol counting weight function is so essential that one often discusses the age-weight ratio to refer to the fine-tuning of their balance. The clause weight function guides the ATP to process small clauses first, approaching the empty clause, and the FIFO weight function helps ensure the completeness of the proof search. There are many additional parameters to tune, such as whether to process all initial clauses before processing new ones. Moreover, strategy scheduling is an important optimization target because, for example, an ensemble of five strategies (of CEFs) run for one second each can outperform one strategy run for five seconds.

Most ATP guidance methods, including ProofWatch and ENIGMAWatch, target the given clause selection. The ATP guidance is done via specialized priority functions (e.g., **PreferWatchlist**) or weight functions (e.g., **Enigma**) that allow the machine learning methods to cooperate with other strategies in selecting clauses natively.

The generation of clauses in E uses term indexes and is very fast once compatible clauses are determined, so there is little room to insert machine-learned models into other parts of the given clause loop. Nonetheless, the Parental Guidance feature successfully filters generated clauses before the simplification step.

3.6 Strategy Invention and Selection

Much human thought can go into fine-tuning *strategy schedules*, which are sequences of strategies to be executed sequentially, each one for a selected time. A *strategy* is a fixed setting of the various options that affect the proof search, which, in the case of E, includes the clause evaluation functions. Automated

strategy invention can improve upon manually designed ATP strategies. Jakubův and Urban [129,239] used iterated local search to hierarchically invent ensembles of strategies that cover the most problems. These strategies are heavily used in our research. Schulz and Schäfer [217] used genetic algorithms to evolve effective strategies, one of which is used in ProofWatch experiments. Holden and Korovin [123], working with iProver [151], interleave Bayesian hyper-parameter optimization to discover heuristics with clustering of problems according to which heuristics work on them (in addition to other features) and then use machine learning to map problems into schedules of heuristics. Mangla et al. [165] use Bayesian statistics to propose permutations of strategies and to optimize time-allocations for iLeanCoP [188]. Rawson and Reger [206] experiment with varying the age-weight ratio throughout a proof in Vampire. Rawson and Reger [205] also implement a run-time strategy scheduler for Vampire to improve proof search time by using a neural network to predict whether a strategy is likely to succeed.

3.7 Watchlists

The hint list method developed by Veroff [247] directs the proof search by prioritizing clauses that *match* clauses (hints) on a hint list. These hints are usually clauses that were useful in related proofs. In E’s implementation, the hint list is called the *watchlist*. A clause is said to match a hint if the clause subsumes the hint. Logical subsumption implies logical entailment (but not vice versa); thus, the watchlist provides logically suggestive guidance.

Hint lists have proved essential in the AIM project [145] to prove an open conjecture in loop theory¹⁰ with the help of Prover9 [170]. The AIM project involved many proofs with over one hundred thousand steps and hint lists created from these long proofs. Some of the automated results allowed Michael Kinyon to prove the weak AIM conjecture.¹¹

ProofWatch is the first successful application of the watchlist technique to a large ITP library (Mizar). ENIGMAWatch combines the watchlist with machine learning guidance of clause selection and presents a novel multi-subsumption index to speed up checking the watchlist. Ruhdorfer and Schulz [216] implemented a special index for unit clause subsumption to help handle large watchlists. When dealing with a large library of diverse problems, automatic curation or creation of watchlists is important. I believe that there is room for future research in this area.

3.8 Datasets

The research in this thesis is conducted with problems from two ITP systems: Mizar [105] and Isabelle [185]. The languages of both systems are translated into the TPTP (Thousands of Problems for Theorem Provers) language [232],

¹⁰An algebraic loop is a quasigroup with an identity element.

¹¹See the talk, “Prover9 Assisted Proof of the Weak AIM Conjecture” from AITP 2021 at <http://aitp-conference.org/2021/> or Veroff’s page: https://www.cs.unm.edu/~veroff/AIM_REDONE/

which is the de-facto standard for automated theorem provers. Urban developed the MPTP (Mizar Problems for Theorem Proving) system [237] to translate from Mizar to TPTP. Mirabelle, a tool in Isabelle, can export to many TPTP formats.

The *Mizar Mathematical Library* (MML) [139] is one of the largest repositories of formal mathematics. The Mizar language uses classical first-order logic and aims to resemble normal human mathematical texts. The MML is built on the foundations of Tarski-Grothendieck set theory [45, 235] and allows weak types. Some famous theorems in the MML are: the Bolzano-Weierstrass theorem [152], Brouwer fixed point theorem [197], Gödel’s completeness theorem [42], and the Jordan curve theorem [150].

Isabelle is a generic proof assistant that uses a weak type theory-based formal proof language in which first-order logic (FOL), higher-order logic (HOL), and Zermelo-Fraenkel set theory (ZFC) have been encoded. The most widely used logic in Isabelle is *Isabelle/HOL*. Isabelle has been used to formalize mathematical results such as Gödel’s incompleteness theorem [191], Gödel’s ontological argument [28], and the prime number theorem [71, 236]. Isabelle is also used to verify software and hardware systems [190, 246], the correctness of security protocols [147, 186], and programming language semantics [126].

The ProofWatch, ENIGMAWatch, Make E Smart Again, and Parental Guidance papers are conducted on (subsets of) a large benchmark of 57 880 problems¹² from the MML exported to first-order logic by the MPTP system.¹³ The MML contains over 1000 articles on diverse mathematical topics. One important subset is the 33 articles leading up to the proof of the Bolzano-Weierstrass theorem, which formed the MPTP Challenge dataset [237, 242] in 2007.¹⁴ There are two common ways of creating problems for the MML datasets:

1. *Bushy*: the theory included for a conjecture are the lemmas used to prove the conjecture in the MML and the necessary background theory.
2. *Chainy*: the theory included for a conjecture consist of all lemmas available in the MML at the time of its proving, which resembles a chronological order and can include many unnecessary lemmas (that need to be filtered through).

The *chainy* problems are intended to be a more realistic setting that requires premise selection. This thesis’s research topic is proof search guidance, so we generally use the *bushy* problems in the FOF (First-Order Formula) TPTP format.

The Isabelle ENIGMA uses a large export of Sledgehammer problems from Isabelle libraries. The Sledgehammer is an interface for proof automation in Isabelle that exports goals to a format for ATPs and reconstructs proofs in

¹²http://grid01.ciirc.cvut.cz/~mtp/1147/MPTP2/problems_small_consist.tar.gz

¹³We are maintaining a compilation of interesting proofs at https://github.com/ai4reas/on/ATP_Proofs.

¹⁴<http://tptp.cs.miami.edu/~tptp/MPTPChallenge/>

Isabelle’s language. The dataset consists of 276 363 problems in the TFF (Many-sorted First-Order Form) TPTP format. These problems are exported from 1902 Isabelle theory files and 179 sessions using the Isabelle/Mirabelle tool. Out of the 179 sessions, 75 were distributed with the Isabelle 2021-1 release, 80 were selected from the Archive of Formal Proofs [36], and 24 sessions were distributed as part of the IsaFoR (Isabelle Formalization of Rewriting) library [234]. Unlike for the MML, for each problem, 512 premises are selected using the MePo filter, a heuristic premise selector developed by Meng and Paulson [175]. These problems include proof-intermediate goals that appear to users rather than just top-level lemmas.

There are many more formal math libraries to which our systems can, in principle, be applied and to which they will probably generalize with some work. The adaptation of our work to additional domains is left as future research for the author, colleagues, or anyone interested and capable.

3.9 ENIGMA: Machine Learning for Given Clause Selection

This section summarizes the development of the ENIGMA family of systems to guide the given clause selection inside E [56, 91, 97, 128, 130, 131, 133]. ENIGMA stands for Efficient learning-based Inference Guiding Machine. The ENIGMA systems have been developed in parallel with this thesis’s research and as a part of my work.

Schulz et al. [220, 224] demonstrated that the choice of the given clause is crucial for the success of the E prover and found that combining strategies in the proof search led to a synergistic gain. Schulz reported that there is huge potential for improvement, and in the following decades, machine learning has delivered. Schulz has developed many heuristics to automatically determine which of E’s parameters and option values to apply based on problem features, including how to schedule strategies. These heuristics are run by calling the “auto schedule” parameter. The latest ENIGMA results with machine learning guidance in Section 5.5 improve upon E’s auto-schedule mode by 60%.

The training of ENIGMA models is usually done in training/evaluation loops:

1. The training data \mathcal{T} are curated from (previous) successful proof searches.
2. A model \mathcal{M} is trained on \mathcal{T} to distinguish positive from negative clauses.
3. \mathcal{M} is run with the ATP (E), usually in *cooperation* with another strategy.
4. Go to step (1) with the new data obtained in step (3).

There are many choices to be made. For example: how do we choose which successful proof searches to mine for training data? Eventually, one can gather hundreds of gigabytes of data, including dozens of proofs for some conjectures. What machine learning models do we use? How does one featurize logical clauses into vectors? Are the same strategies effective independently and in cooperation

with an ENIGMA model? We consider proof clauses as “positive”, but how do we know that the other clauses are truly “negative” and could not contribute to a different proof?

To start with the easy questions, for each proof, we consider all clauses in the proof to be positive and all the selected clauses not in the proof to be negative. We have no guarantee that a “*negative*” clause not used in a proof is a “true negative”. We hope the learning algorithms will manage the ambiguity when the same clause is positive in one proof of a conjecture and negative in another proof of the same conjecture.

In the initial ENIGMA experiments, we collected all the training examples from all successful proof searches. Once the data grew too big, multiple approaches were employed. One simple approach is to start over using only the best model’s data. Another approach is to use a greedy cover over all runs to ensure that no solved problems’ proofs are lost and to use all the data from the runs in the greedy cover. The most sophisticated approach is to fix some k , such as $k := 3$, and only keep k proofs per problem in the training data. Jakubův found that the combination of a “shortest proof”, a “middle-length proof”, and a “longest proof”, was more effective than just taking one proof or three random proofs. I call this “infinite-order looping” as the data scales linearly with the number of proofs and can be continued indefinitely. In all of the methods, sometimes it has been helpful to prune negative samples (in addition to the data balancing provided by the ML toolkits). I recommend infinite-order looping for future research.

Strong strategies discovered with BliStrTune [129] have also performed well in cooperation with ENIGMA models. The strategies also perform well with watchlist methods (replacing priority functions with a **PreferWatchlist** priority function). E incorporates the ENIGMA models’ predictions into a weight function. One way to do this is to assign a weight of 10 to *negative* clauses and a weight of 1 to *positive* clauses. Usually, we combine the ENIGMA clause evaluation with the strong E strategies in a balanced manner where ENIGMA chooses 50% of the clauses.

Machine learning methods and data featurization techniques have undergone many developments over the generations. The models based on recursive neural networks [56] and graph neural networks [128, 187] aim to preserve the syntactic structure of the mathematical formulae, bypassing the need for hand-crafted featurization.

The *ENIGMA features* are based on term walks over the syntax trees of the literals in clauses. Term walks of length 3 are usually used and different values have not provided significantly different results. Variable and Skolem symbols are abstracted into the symbols \oplus and \ominus , respectively. Each unique combination of three symbols is assigned an index in the *feature vector*. For a literal (a formula), all (top-down) directed paths of length three are counted, and the counts are stored in the assigned indices of the feature vector. To process a clause, one sums the feature vectors of each literal. These are called *vertical features* and were the sole features in the first ENIGMA version [130]. In later versions, additional features have been added. I list some of the features below:

1. **Vertical features:** top-down term walks of length 3 of clause’s literals syntax trees. The literal features are summed up to produce the clause features. For example, some vertical features of $f(x, y) = g(\text{sko}_1, \text{sko}_2(x))$ would be $\{ (=, f, \oplus), (g, \odot, \oplus), (=, g, \odot), \dots \}$.
2. **Horizontal features:** include, for every term, the term’s head symbol and the top-level symbols of its arguments. For example, the horizontal features of the unit clause, $P(f(g(a), g(a)))$, are $\{P(f) : 1, f(g, g) : 1, g(a) : 2\}$
3. **Length features:** clause length, the counts of positive and negative literals, and similar statistics.
4. **Symbol features:** for each symbol in a clause, its number of occurrences and maximum depth in positive and negative literals.
5. **Variable statistics:** various statistics on the variable frequencies in the clause.
6. **Conjecture features:** merge the features of all the conjecture clauses in the target conjecture and append this to the clause features.
7. **Theory features:** merge the features of all axiom clauses one aims to use to prove the target conjecture and append this to the clause features.
8. **Problem features:** add the problem features that E already internally computes, such as the number of goals, axioms, and unit goals.
9. **Proof vector features:** use the watchlist feature to load multiple proofs and keep track of how many clauses in each proof have been matched. Add this watchlist *progress* vector to the feature vectors.
10. **Parent features:** append the parent clause feature vectors.
11. **Feature hashing:** use the *sdbm* hash function on *feature strings* to map features into a manageable number of buckets (e.g., from 2^{10} to 2^{15}).
12. **Anonymous features:** abstract function and symbol names into *fn* and *pn* based on the arity *n*.
13. **Type features:** when using a typed format (such as THF), the type (sort) information becomes part of the symbol names. For example, “mult” is of type “nat \rightarrow nat \rightarrow nat”, so the typed symbol name will be “mult:nat>nat>nat” and with anonymization the symbol name is “f2: _> _> _”.

The subset of features to be used is an additional parameter that can be tuned. One can imagine different features, such as using the whole derivation history of a clause instead of parent features, which Suda does in Deepire [230, 231]. The feature hashing was seminal to achieving performance on large libraries such as Mizar and Isabelle, as performance deteriorated when vectors exceeded

10^6 features. Curiously, in some experiments, hashing features into only 32 buckets led to some new proofs. ENIGMA models trained with anonymous features perform on par with those with access to symbol names. This feature is especially important on datasets where symbol names are not used consistently (such as Isabelle Sledgehammer datasets). The most up-to-date information on the features can be found in [56, Section 3] and [128, Section 3]. The selection of features and their parameters can be fine-tuned on a domain-specific basis.

The first ENIGMA version [130] used Support Vector Machine Classification [41] from LIBLINEAR [73] as the underlying machine learning for clause classification and selection. The second version [131] uses a fast logistic regression algorithm from LIBLINEAR. The version used in ENIGMAWatch [91, 93, 133] introduced feature hashing and gradient-boosted decision trees (GBDTs) with XGBoost [54] instead of LIBLINEAR. ENIGMA-NG [56] uses XGBoost and recursive neural networks, aiming for a syntactically faithful way to process formulae. ENIGMA Anonymous [128] introduced anonymous features and switched to LightGBM [144] because it handles large datasets better and has more parameters to toggle. The most distinct difference is that LightGBM grows decision trees leaf-first and allows the user to choose the number of leaves in the trees, whereas XGBoost grows trees breadth-first. ENIGMA Anonymous also successfully uses a symbol-independent graph neural network (GNN) [187] for given clause selection guidance, which should capture additional structural information about the clauses. The GNN allows for the contextual evaluation of clauses by adding processed clauses to the graph in addition to the clauses to be evaluated. The latest version is the “fast and slow” 2-phase ENIGMA [97], combining the GNN and GBDT guidance, using the GBDT as a fast rejection filter before clauses are sent to a GNN server for evaluation.

3.10 ENIGMA: Parental Guidance

The latest version of ENIGMA [97] also includes *Parental Guidance*, which filters generated clauses prior to clause selection based on the features of their parents alone. Combined with the 2-phase ENIGMA, this results in the 3-phase ENIGMA, a system that combines 3 ML models to guide E and attained our strongest single strategy on Mizar.

4 Related Research

4.1 ENIGMA: Leapfrogging and Reasoning Components

Chvalovský et al. [57] introduce two ATP-external learning techniques that rely on the GNN’s contextual evaluation of clauses. The *leapfrogging* approach is to run the theorem prover on a problem for a fixed time limit. If the problem is not solved, a graph-based predictor selects a subset of the processed clauses with which to run E again. Two rounds of leapfrogging led to new, complementary problems being proven in the Mizar dataset. The idea is that in addition to focusing the proof search, the GNN can also give different evaluations in different contexts. The next technique is to augment leapfrogging by using graph clustering algorithms to split the set of processed clauses into components that can be given to the theorem prover separately. Learning reasoning components is a bottom-up approach to identifying subgoals: mathematical problems often have well-separated reasoning components, for example, when there is a need to compute a derivative to use in another equation.

4.2 Premise Selection

In an ideal world, a conjecture may be given to a general-purpose automated theorem prover in an expressive, QED-like logical framework that allows theorems to be translated between formal mathematical libraries such as Dedukti¹⁵ or MMT¹⁶ [149, 202]. The theorem prover will then choose appropriate theorems and lemmata from the available formal mathematics libraries in order to prove the conjecture. If an additional lemma appears to be needed during the proof search, the general ATP will see if any usable theorems already exist. If not, the ATP will begin conjecturing and trying to discharge the sub-goals.

At present, ATPs can be confused and bogged down if given a large number of theory formulas. Thus there is the need to select, for each conjecture, a subset of axioms from which the conjecture is provable (with minimal false positives). The premises used by human formalizers can be helpful as learning data; however, sometimes, ATPs find alternative proofs using different premises. A process of *pseudo-minimization* [137] is often done where only the premises used in a proof are kept, and the ATP is re-run until a fixed point is reached.

Non-learning methods for premise selection usually iteratively add premises based on some heuristic. The SInE [122] method for axiom selection is a lightweight state-of-the-art default used in Vampire and E that iteratively selects axioms based on the overlapping symbols with the conjecture and already selected axioms. SInE stands for “SUMO Inference Engine”, which is a premise selector for large theories such as SUMO [184, 192]. The MePo relevance filter [175] by Meng and Paulson keeps track of relevant symbols and features, iteratively selecting premises with perfect or high ratios of relevant to irrelevant symbols. SRASS [233], the Semantic Relevance Axiom Selection System, employs finite

¹⁵<https://deducteam.github.io/>

¹⁶<https://uniformal.github.io/doc/>

model finders to find countermodels to the conjecture and gradually adds premises to exclude them.

Urban [238, 243] developed MaLAREa, a Machine Learner for Automated Reasoning, a metasystem that iterates between proving theorems and learning how to select premises for the problems, using the premises of the proven theorems as training data. MaLAREa SG-1 (with Semantic Guidance) incorporates information about which premises break which countermodels as additional training features. MaSh [35], the Machine learner for Sledgehammer, implements naive Bayes and k-nearest neighbors algorithms for premise selection for the Isabelle/HOL Sledgehammer. Färber et al. [74] performed preliminary experiments with random forests for premise selection. Piotrowski et al. [194] developed ATPBoost, which uses XGBoost [54] for iterative premise selection learning and theorem proving loops. Many alternative proofs with different sets of premises can be found. One challenge with learning premise selection is that “the absence of evidence is only weak evidence of absence”; the lack of a proof from a given set of axioms does not mean that one does not exist. However, proving counter-satisfiability does mean that there is no proof of contradiction.

A number of learning-based premise selection methods have been developed for large ITP corpora and hammers in the last two decades. See [6, 37, 156, 240] for their overviews.

Szegedy et al. [7] applied neural sequence models and convolutional neural networks (CNNs) to premise selection over Mizar and found that the CNNs worked better than the combination. Piotrowski et al. [195] apply recurrent neural machine translation (NMT) models to select sequences of premises so that premises are statefully chosen based on the previous premises. The recurrent NMT models perform well, proving theorems orthogonally to XGBoost’s premise selection, and are not dependent on the featurization of formulas. The property-invariant graph neural network developed by Olšák et al. [187], which is used to internally guide the proof search, is also applied to premise selection. The conjecture and theory formulas can be added to the graph neural network that outputs ranks for the premises. This network performs well on the Mizar/MPTP and Isabelle/Sledgehammer datasets.

4.3 Hammers

If the only tool you have is a hammer, it is tempting to treat everything as if it were a nail. (Abraham Maslow)

Hammers are automated tools for discharging (sub)goals in interactive theorem proving (ITP) environments. A hammer usually relies on an automated theorem prover or satisfiability modulo theories (SMT) solver. Because ATPs and SMTs often work with different logics than the ITP systems, a full-fledged hammer must translate a goal and premises into the appropriate logic and then reconstruct the proof in the ITP system’s logic. The TPTP language is the de-facto standard for the ATP and SMT solvers used by hammers. Moreover, a hammer must select premises from the ITP library to give to the solvers.

Hammering Towards QED [37] provides a good overview of hammers. Urban’s MPTP system [237] and the Sledgehammer system [39] were among the first to demonstrate that ATPs can be effectively applied as hammers. The MPTP system translates between the Mizar Mathematical Library and the TPTP format. In 2006, the MPTP translation allowed ATPs to prove 39% of the non-arithmetical problems in the MML [237]. The 2010 paper Judgment Day [40] demonstrated that hammers could prove 34% of the non-trivial goals and 45% of the goals in their dataset that reflects typical Isabelle/HOL developments.

Two recent benchmarks, GRUNGE: A Grand Unified ATP Challenge [44] and Seventeen Provers under the Hammer [66], achieve higher performance and cover many more theorem provers.

GRUNGE evaluates 19 state-of-the-art ATPs and SMTs on a HOL4 [228] dataset that contains higher-order and first-order TPTP formats with syntactic and semantic (set theoretic) translations. The 60s portfolio of the provers solves 61% of the problems. Except for Leo-III, the strongest prover on the GRUNGE benchmark, the provers performed better on first-order formats. Most provers perform better on the syntactic translations; however, Vampire and SPASS performed better on the semantic translations.

Seventeen Provers under the Hammer evaluates 17 ATPs and SMTs on an Isabelle/HOL dataset to determine which TPTP formats and encodings work best with each prover. The 30s portfolio of the provers solves 70% of the problems. One notable result is that the new higher-order E [248] performed best on the THF (many-sorted higher-order logic) format. Furthermore, the TFF (many-sorted first-order logic) format almost always outperformed the FOF encodings, suggesting the direction of future developments. Since human mathematics is often more easily expressed in higher-order logics,

There are also some hammers for Coq, for example: CoQHammer [61, 62] and SMTCoq [14].

4.4 Tactical Theorem Provers

Interactive theorem proving is usually done via high-level parametrizable tactics that perform sound proof transformations of the proof state. The tactic language for the HOL4 system [228] is Standard SML, which means that tactics can be arbitrarily complex programs. Thus AI for ITP systems can enjoy a larger search space over high-level tactics instead of just lower-level logical inferences on the formulas in the proof state.

The first such work is TacticToe [82, 83] by Gauthier for HOL4. Loos et al. aim to do similar work in HOL Light [115] with DeepHOL [22] and provide the environment HOList for prototyping AI for the task. Huang and Dhariwal introduce GamePad [125], a Python API for interacting with Coq to develop AI for position evaluation and tactic prediction. Blaauwbroek et al. developed the Tactician¹⁷ [33, 34, 257], which has been integrated into Coq’s package manager. Tactician learns from previously written tactic scripts to give suggestions or

¹⁷<https://coq-tactician.github.io/>

attempt to complete the proof automatically.

One limitation of these systems is that they rely on pre-defined tactics, learning from human proofs. The task of synthesizing tactics is one of program synthesis. To this end, Gauthier [80,81] worked on tree neural networks to synthesize term functions in HOL4, such as combinators and diophantine equations. More recently, Gauthier et al. [84] developed tree neural networks that synthesize programs to fit integer sequences in the OEIS (On-Line Encyclopedia of Integer Sequences) [229].

4.5 Machine Learning for the leanCoP Family

There is an older, parallel line of research to the ML guidance of saturation-based ATPs: machine learning for guiding variants of leanCoP¹⁸ [189], a theorem prover famously based on the connection (tableau) calculus [30–32]. The leanCoP Prolog code infamously fits in the paper’s abstract. A tableau is a tree with nodes labeled by formulas, and a connection calculus-based prover, such as leanCoP, starts with a goal clause and seeks to *close* the tableau, meaning that every branch has a *connection*. A connection is a pair of contradictory literals (e.g., $\{A(t_1, \dots, t_n), \neg A(t_1, \dots, t_n)\}$). The proof search consists of extension and reduction steps. Extension steps add new goals along a unifying connection, and reduction steps close off a branch by identifying a unifying connection along the active path. A closed tableau is a proof.

A partial tableau provides a compact notion of the proof search state, which in addition to the simplicity, renders the leanCoP family of provers a fertile ground for prototyping ML guidance. The ML models can be used to offer guidance for both extension and reduction steps. Sometimes multiple clauses unify with the current goal, and ML advice can prove helpful. Allowing a guidance system to learn when to apply reduction steps can also help.

The below list covers the systems incorporating ML guidance into versions of leanCoP in chronological order.

- MaLeCoP: the Machine Learning Connection Prover uses SNoW’s Naive Bayes classifier to guide leanCoP [244].
- FEMaLeCoP: the Fairly Efficient Machine Learning Connection Prover also uses a Naive Bayes Classifier, now implemented in OCaml, and is probably *the first efficient ATP-internal guidance system* [138].
- monteCoP: implements a Monte Carlo Tree Search (MCTS) to guide leanCoP, testing various evaluation heuristics [75].
- rlCoP: uses reinforcement learning with a GBDT framework to learn policy and value functions to guide leanCoP with MCTS [140].
- plCoP: is an open-source toolkit extending rlCoP in a Prolog implementation with a Python interface [259].

¹⁸The leanCoP prover’s name is always written in lowercase.

- graphCoP: is similar to rCoP and uses Olšák et al.’s Graph Neural Network instead of the GBDT framework XGBoost [187].
- pl-graphCoP: incorporates the GNN into plCoP and finds that adding entropy to the police during training greatly improves performance (following probability matching theory to reduce the certainty of predictions) [260].
- lazyCoP: introduces the *lazy paramodulation* proof calculus to allow equality-rewriting in the connection tableau setting. lazyCoP also introduces asynchronous policy evaluation of nodes by a directed graph neural network on a GPU, allowing the proof search speed to be less dependent on the evaluation speed [207].

The pl-graphCoP system proves 66.8% more ATP-minimized problems than leanCoP on the Mizar 40 dataset [139], which is a 17.4% improvement over rCoP. However, in part due to leanCoP’s handling of equality reasoning, the leanCoP family is generally weaker than the superposition-calculus-based saturation provers.

4.6 Automated Formalization and Neural Theorem Provers

Automated formalization (*autoformalization*) [141, 142, 250] aims to translate between informal mathematics, written by human mathematicians, and formal mathematics in specific formal languages. Writing formal mathematics is known to be difficult and significantly slower, even for trained mathematicians. Therefore the corpora of formal mathematics will be limited in size until autoformalization improves. There is the chicken-and-egg problem of acquiring quality data for training autoformalization models that are good enough to assist in generating more data to train better models. Kevin Buzzard is spearheading an effort to formalize all undergraduate mathematics in Lean [181]. The reverse, ‘informalization’, can also be valuable for people not well-versed in the formal library; however, this task is probably easier and more similar to “pretty printing”.

Building on work by Bancerek et al. [19–21], Wang et al. [250] developed three datasets on which to test neural machine translation: a synthetic dataset of LaTeX generated from Mizar, a partially aligned dataset of ProofWiki’s LaTeX and Mizar, and a dataset of aligned Mizar and TPTP formulas (FOF and THF, typed higher-order form). The perplexity and BLEU scores on the synthetic (formal-to-formal) tasks are encouraging. The authors concluded that more data is needed and next worked on joint embeddings of formal proof libraries to help match concepts across libraries [251].

There are newer datasets of problems drawn from competition problems, such as the MATH dataset [117], which aligns LaTeX and English problem statements and solutions, and the MiniF2F dataset [258], which aligns multiple formal systems, including Lean, Metamath, Isabelle, and some HOL Light.

There has been some preliminary research on using large language models directly to generate mathematical formulas and proof steps. Urban and

Jakubův [241] used GPT-2 [203] to generate Mizar texts, conjectures, and intermediate lemmas in several settings. Many of them are syntactically correct, and some of the conjectures are provable by the MizAR ATP hammer. This was followed by IsarStep [164], a dataset of Isabelle/HOL problems designed for the testing and development of AI for the task of filling in intermediate lemmas. Instead of GPT, the authors trained a hierarchical transformer. LeanStep [113] is a similar dataset for *next proof term* and *next lemma* prediction developed for Lean, using GPT again. The following pre-prints seem interesting: Wu et al. [254] apply the large language models PaLM [55] and Codex [53] to the intersection of MATH problems and MiniF2F’s Isabelle problems. Wu et al. then use the autoformalization of MATH problems as additional training data for a language model to be used as a neural theorem prover. Gur-Ari et al. [163] find that the additional training of PaLM on arXiv and mathematical web pages seems to help. Jiang et al. [135] have developed Thor, a system that integrates language models with ATP hammers by allowing the language model to call hammers with a keyword, “<hammer>”, which enables Thor to find proofs for some problems that neither Sledgehammer nor the language model could alone. This is an interesting area of ongoing research. It seems likely that neural networks that preserve the mathematical structure, such as Olšák et al.’s graph neural networks [187], will be a part of the development and publications.

The current neurally guided theorem proving systems, such as the GNN in ENIGMA and neural syntheses via large language models, are steps on the path to fully neural theorem provers. Gur-Ari et al. [163] find that using ensemble methods helps PaLM, and they run into the limitation that in LaTeX, one cannot automatically verify the correctness of solutions as in ITP system languages. Once the language model is integrated with automatic verification and additional algorithmic heuristics, the distinction becomes less clear. The graph neural network in ENIGMA Anonymous [128] only scores the generated mathematical clauses; however, the generation procedure with E’s superposition logic is computationally cheap. Neural generation of inference steps and the verification of their correctness remains an open topic.

There is work by Hahn et al. [108] using a transformer to predict satisfying assignments to propositional LTL (Linear Temporal Logic) formulas. In “LIME: Learning Inductive Bias for Primitives of Mathematical Reasoning” [255], Wu et al. create syntactic datasets illustrating *deduction*, *abduction*, and *induction* on which to pre-train a vanilla transformer, improving performance on IsarStep, LeanStep, and Metamath. Krishna et al. [154] develop ProofFVer, a system for fact-checking natural language claims with respect to a set of evidence. ProofFVer uses a pretrained BART [161] model with lexical constraints on the decoder to ensure that the proofs are well-formed. Riedel and Rocktäschel [213] developed a neural theorem prover (NTP) inspired by Prolog’s backward chaining to do gradient descent over all possible proof paths via *unification*, *or*, and *and* modules. Minervini et al. scale up the NTP with the Greedy [176] and Conditional NTPs [177], which limit the search space. The Conditional NTPs include a *select* module that, for a given goal, produces the rules needed to prove it. Piotrowski et al. [196] train neural machine translation (NMT) models that

can learn to do rewriting steps from a dataset of Prover9 instances, and that can do polynomial normalization. Piepenbrock et al. [193] develop a GNN2RNN architecture that produces instances of first-order clauses to give to a SAT-based solver.

5 Contributed Research Papers

This section contains the six research publications of the thesis, which are included without modification.

Overview

1. ProofWatch (published at ITP 2018) [89] is based on the *watchlist* (also *hint list*) technique. The watchlist technique focuses the proof search toward lemmas (*hints*) that were useful in related proofs. We test E’s watchlist feature on a dataset from the *Mizar Mathematical Library* [139], one of the largest libraries of formalized mathematics, written in the set theory-based proof assistant Mizar [105]. We add a *dynamic watchlist* feature that loads in multiple watchlists and guides the proof search based on the proof state’s similarity to the related proofs on the watchlists. We use the k-nearest neighbors algorithm to select related proofs (or clauses) to put on the watchlists. The best strategy, which uses 16 relevant proofs as watchlists, improves upon the E baseline strategy that it is based on by 26.5%. The best ensemble of five strategies proves 7% more on Mizar than the best five non-watchlist strategies.
2. ENIGMAWatch (published at IWIL 2018 and TABLEAUX 2019) [90, 91] combines ProofWatch with the ENIGMA system developed by Jakubuv and Urban [130, 131, 133]. ENIGMA stands for Efficient learNing-based Inference Guiding MAchine and is a system that uses fast statistical machine learning to train models from related proof searches to identify *positive* and *negative* (good and bad) clauses for the current conjecture. ENIGMA chooses the given clauses for E based on clause features extended with the problem’s conjecture features, which are static throughout the whole proof search.

ProofWatch provides semantic embeddings to provide a dynamic proof search state vector to ENIGMA’s model. The ENIGMAWatch system trains faster than ENIGMA and, in the first training loop, proved 8.8% more problems. The asymptotic performance beyond the fifth loop is the same with high complementarity, which means that the union of ENIGMA and ENIGMAWatch’s solved problems is larger than either method alone.

3. Make E Smart Again (published at IJCAR 2020) [96] tests the capacity of ENIGMA to learn to guide E without the help of well-crafted term orderings, strong literal selection functions, and strong strategies [129, 239]. Effective term orderings and literal selection can guarantee the completeness of the proof search [121, Section 3], and they greatly reduce the number of redundancies in generated clauses. For this project, I developed E0, which uses a structural identity relation as the minimal ordering, resulting in a simplified version of E without some of E’s features and the theoretical guarantees ENIGMA must filter out far more unnecessary clauses in this

setting than in the standard scenario. ENIGMA with E0 can prove 121-256% more problems than E0 alone and surpasses the performance of E with two strong strategies by 10%.

4. Parental Guidance (published at FroCoS 2021) [97] filters the generated clauses based on features of their parent clauses, thus easing the clause selection task of ENIGMA models and E strategies. This work also includes a graph neural network (GNN) server for given clause selection and the combination of this *slow* GNN clause selection with a *fast* gradient-boosted decision tree (GBDT) pre-filter, which is called a 2-phase ENIGMA. The combination of all three methods: Parental Guidance, the GNN, and the GBDT, make up the 3-phase ENIGMA, which improves upon E's auto-schedule by 60% and upon the best previous result by 17.4%.
5. The Isabelle ENIGMA (published at ITP 2022) [98] applies ENIGMA, Parental Guidance, and GNN-based premise selection to one of the largest corpora of Isabelle Sledgehammer problems, that is, problems translated into a format intended for ATPs. On the holdout set, the premise selection improves the performance of E's auto-schedule by 19.7%. The 2-phase ENIGMA, consisting of Parental Guidance and clause selection models, outperforms E's auto-schedule by 25.3% and outperforms the other provers, albeit only beating the CVC5 prover [23] by 0.3%.



ProofWatch: Watchlist Guidance for Large Theories in E

Zarathustra Goertzel¹() , Jan Jakubův¹, Stephan Schulz², and Josef Urban¹

¹ Czech Technical University in Prague, Prague, Czech Republic
`goertzar@fel.cvut.cz`

² DHBW Stuttgart, Stuttgart, Germany

Abstract. Watchlist (also hint list) is a mechanism that allows related proofs to guide a proof search for a new conjecture. This mechanism has been used with the Otter and Prover9 theorem provers, both for interactive formalizations and for human-assisted proving of open conjectures in small theories. In this work we explore the use of watchlists in large theories coming from first-order translations of large ITP libraries, aiming at improving hammer-style automation by smarter internal guidance of the ATP systems. In particular, we (i) design watchlist-based clause evaluation heuristics inside the E ATP system, and (ii) develop new proof guiding algorithms that load many previous proofs inside the ATP and focus the proof search using a dynamically updated notion of proof matching. The methods are evaluated on a large set of problems coming from the Mizar library, showing significant improvement of E's standard portfolio of strategies, and also of the previous best set of strategies invented for Mizar by evolutionary methods.

1 Introduction: Hammers, Learning and Watchlists

Hammer-style automation tools connecting interactive theorem provers (ITPs) with automated theorem provers (ATPs) have recently led to a significant speedup for formalization tasks [5]. An important component of such tools is *premise selection* [1]: choosing a small number of the most relevant facts that are given to the ATPs. Premise selection methods based on machine learning from many proofs available in the ITP libraries typically outperform manually specified heuristics [1, 2, 4, 7, 17, 19]. Given the performance of such *ATP-external guidance* methods, learning-based *internal proof search guidance* methods have started to be explored, both for ATPs [8, 15, 18, 23, 36] and also in the context of tactical ITPs [10, 12].

In this work we develop learning-based internal proof guidance methods for the E [30] ATP system and evaluate them on the large Mizar Mathematical Library [11]. The methods are based on the *watchlist* (also *hint list*) technique

Z. Goertzel, J. Jakubův and J. Urban—Supported by the *AI4REASON* ERC Consolidator grant number 649043, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15.003/0000466 and the European Regional Development Fund.

developed by Veroff [37], focusing proof search towards lemmas (*hints*) that were useful in related proofs. Watchlists have proved essential in the AIM project [21] done with Prover9 [25] for obtaining very long and advanced proofs of open conjectures. Problems in large ITP libraries however differ from one another much more than the AIM problems, making it more likely for unrelated watchlist lemmas to mislead the proof search. Also, Prover9 lacks a number of large-theory mechanisms and strategies developed recently for E [13, 15, 16].

Therefore, we first design watchlist-based clause evaluation heuristics for E that can be combined with other E strategies. Second, we complement the internal watchlist guidance by using external statistical machine learning to pre-select smaller numbers of watchlist clauses relevant for the current problem. Finally, we use the watchlist mechanism to develop new proof guiding algorithms that load many previous proofs inside the ATP and focus the search using a *dynamically* updated heuristic representation of *proof search state* based on matching the previous proofs.

The rest of the paper is structured as follows. Section 2 briefly summarizes the work of saturation-style ATPs such as E. Section 3 discusses heuristic representation of search state and its importance for learning-based proof guidance. We propose an abstract vectorial representation expressing similarity to other proofs as a suitable evolving characterization of saturation proof searches. We also propose a concrete implementation based on *proof completion ratios* tracked by the watchlist mechanism. Section 4 describes the standard (*static*) watchlist mechanism implemented in E and Sect. 5 introduces the new *dynamic* watchlist mechanisms and its use for guiding the proof search. Section 6 evaluates the static and dynamic watchlist guidance combined with learning-based pre-selection on the Mizar library. Section 7 shows several examples of nontrivial proofs obtained by the new methods, and Sect. 8 discusses related work and possible extensions.

2 Proof Search in Saturating First-Order Provers

The state of the art in first-order theorem proving is a saturating prover based on a combination of resolution/paramodulation and rewriting, usually implementing a variant of the superposition calculus [3]. In this model, the *proof state* is represented as a set of first-order clauses (created from the axioms and the negated conjecture), and the system systematically adds logical consequences to the state, trying to derive the empty clause and hence an explicit contradiction.

All current saturating first-order provers are based on variants of the *given-clause algorithm*. In this algorithm, the proof state is split into two subsets of clauses, the processed clauses P (initially empty) and the unprocessed clauses U . On each iteration of the algorithm, the prover picks one unprocessed clause g (the so-called *given clause*), performs all inferences which are possible with g and all clauses in P as premises, and then moves g into P . The newly generated consequences are added to U . This maintains the core invariant that all inferences between clauses in P have been performed. Provers differ in how they integrate simplification and redundancy into the system, but all enforce the variant that

P is maximally simplified (by first simplifying g with clauses in P , then back-simplifying P with g) and that P contains neither tautologies nor subsumed clauses.

The core choice point of the given-clause algorithm is the selection of the next clause to process. If theoretical completeness is desired, this has to be *fair*, in the sense that no clause is delayed forever. In practice, clauses are ranked using one or more heuristic evaluation functions, and are picked in order of increasing evaluation (i.e. small values are good). The most frequent heuristics are based on symbol counting, i.e., the evaluation is the number of symbol occurrences in the clause, possibly weighted for different symbols or symbols types. Most provers also support interleaving a symbol-counting heuristic with a first-in-first-out (FIFO) heuristic. E supports the dynamic specification of an arbitrary number of differently parameterized priority queues that are processed in weighted round-robin fashion via a small *domain-specific language* for heuristics.

Previous work [28,31] has both shown that the choice of given clauses is critical for the success rate of a prover, but also that existing heuristics are still quite bad - i.e. they select a large majority of clauses not useful for a given proof. Positively formulated, there still is a huge potential for improvement.

3 Proof Search State in Learning Based Guidance

A good representation of the current *state* is crucial for learning-based guidance. This is quite clear in theorem proving and famously so in Go and Chess [32,33]. For example, in the TacticToe system [10] proofs are composed from pre-programmed HOL4 [34] tactics that are chosen by statistical learning based on similarity of the evolving *goal state* to the goal states from related proofs. Similarly, in the learning versions of leanCoP [26] – (FE)MaLeCoP [18,36] – the tableau extension steps are guided by a trained learner using similarity of the evolving tableau (the ATP *proof search state*) to many other tableaux from related proofs.

Such intuitive and compact notion of proof search state is however hard to get when working with today’s high-performance saturation-style ATPs such as E [30] and Vampire [22]. The above definition of saturation-style proof state (Sect. 2) as either one or two (processed/unprocessed) large sets of clauses is very unfocused. Existing learning-based guiding methods for E [15,23] practically ignore this. Instead, they use only the original conjecture and its features for selecting the relevant given clauses throughout the whole proof search.

This is obviously unsatisfactory, both when compared to the evolving search state in the case of tableau and tactical proving, and also when compared to the way humans select the next steps when they search for proofs. The proof search state in our mind is certainly an evolving concept based on the search done so far, not a fixed set of features extracted just from the conjecture.

3.1 Proof Search State Representation for Guiding Saturation

One of the motivations for the work presented here is to produce an intuitive, compact and evolving heuristic representation of proof search state in the context of learning-guided saturation proving. As usual, it should be a vector of (real-valued) features that are either manually designed or learned. In a high-level way, our proposed representation is a *vector expressing an abstract similarity of the search state to (possibly many) previous related proofs*. This can be implemented in different ways, using both statistical and symbolic methods and their combinations. An example and motivation comes again from the work of Veroff, where a search is considered promising when the given clauses frequently match hints. The gaps between the hint matchings may correspond to the more brute-force bridges between the different proof ideas expressed by the hints.

Our first practical implementation introduced in Sect. 5 is to load upon the search initialization N related proofs P_i , and for each P_i keep track of the ratio of the clauses from P_i that have already been subsumed during the search. The subsumption checking is using E’s watchlist mechanism (Sect. 4). The N -long vector \mathbf{p} of such *proof completion ratios* is our heuristic representation of the proof search state, which is both compact and typically evolving, making it suitable for both hard-coded and learned clause selection heuristics.

In this work we start with fast hard-coded watchlist-style heuristics for focusing inferences on clauses that progress the more finished proofs (Sect. 5). However training e.g. a statistical ENIGMA-style [15] clause evaluation model by adding \mathbf{p} to the currently used ENIGMA features is a straightforward extension.

4 Static Watchlist Guidance and Its Implementation in E

E originally implemented a watchlist mechanism as a means to force direct, constructive proofs in first order logic. For this application, the watchlist contains a number of goal clauses (corresponding to the hypotheses to be proven), and all newly generated and processed clauses are checked against the watchlist. If one of the watchlist clauses is subsumed by a new clause, the former is removed from the watchlist. The proof search is complete, once all clauses from the watchlist have been removed. In contrast to the normal proof by contradiction, this mechanism is not complete. However, it is surprisingly effective in practice, and it produces a proof by forward reasoning.

It was quickly noted that the basic mechanism of the watchlist can also be used to implement a mechanism similar to the *hints* successfully used to guide Otter [24] (and its successor Prover9 [25]) in a semi-interactive manner [37]. Hints in this sense are intermediate results or lemmas expected to be useful in a proof. However, they are not provided as part of the logical premises, but have to be derived during the proof search. While the hints are specified when the prover is started, they are only used to guide the proof search - if a clause matches a hint, it is prioritized for processing. If all clauses needed for a proof are provided as hints, in theory the prover can be guided to prove a theorem without any

search, i.e. it can *replay* a previous proof. A more general idea, explored in this paper, is to fill the watchlist with a large number of clauses useful in proofs of similar problems.

In E, the watchlist is loaded on start-up, and is stored in a feature vector index [29] that allows for efficient retrieval of subsumed (and subsuming) clauses. By default, watchlist clauses are simplified in the same way as processed clauses, i.e. they are kept in normal form with respect to clauses in P . This increases the chance that a new clause (which is always simplified) can match a similar watchlist clause. If used to control the proof search, subsumed clauses can optionally remain on the watchlist.

We have extended E’s domain-specific language for search heuristics with two priority functions to access information about the relationship of clauses to the watchlist - the function `PreferWatchlist` gives higher rank to clauses that subsume at least one watchlist clause, and the dual function `DeferWatchlist` ranks them lower. Using the first, we have also defined four built-in heuristics that preferably process watchlist clauses. These include a pure watchlist heuristic, a simple interleaved watch list function (picking 10 out of every eleven clauses from the watchlist, the last using FIFO), and a modification of a strong heuristic obtained from a genetic algorithm [27] that interleaves several different evaluation schemes and was modified to prefer watchlist clauses in two of its four sub-evaluation functions.

5 Dynamic Watchlist Guidance

In addition to the above mentioned *static watchlist guidance*, we propose and experiment with an alternative: *dynamic watchlist guidance*. With dynamic watchlist guidance, several watchlists, as opposed to a single watchlist, are loaded on start-up. Separate watchlists are supposed to group clauses which are more likely to appear together in a single proof. The easiest way to produce watchlists with this property is to collect previously proved problems and use their proofs as watchlists. This is our current implementation, i.e., each watchlist corresponds to a previous proof. During a proof search, we maintain for each watchlist its *completion status*, i.e. the number of clauses that were already encountered. The main idea behind our dynamic watchlist guidance is to prefer clauses which appear on watchlists that are closer to completion. Since watchlists now exactly correspond to previous refutational proofs, completion of any watchlist implies that the current proof search is finished.

5.1 Watchlist Proof Progress

Let watchlists W_1, \dots, W_n be given for a proof search. For each watchlist W_i we keep a *watchlist progress counter*, denoted $progress(W_i)$, which is initially set to 0. Whenever a clause C is generated during the proof search, we have to check whether C subsumes some clause from some watchlist W_i . When C subsumes a clause from W_i we increase $progress(W_i)$ by 1. The subsumed clause from

W_i is then marked as encountered, and it is not considered in future watchlist subsumption checks.¹ Note that a single generated clause C can subsume several clauses from one or more watchlists, hence several progress counters might be increased multiple times as a result of generating C .

5.2 Standard Dynamic Watchlist Relevance

The easiest way to use progress counters to guide given clause selection is to assign the (*standard*) *dynamic watchlist relevance* to each generated clause C , denoted $relevance_0(C)$, as follows. Whenever C is generated, we check it against all the watchlists for subsumption and we update watchlist progress counters. Any clause C which does not subsume any watchlist clause is given $relevance_0(C) = 0$. When C subsumes some watchlist clause, its relevance is the maximum watchlist completion ratio over all the matched watchlists. Formally, let us write $C \sqsubseteq W_i$ when clause C subsumes some clause from watchlist W_i . For a clause C matching at least one watchlist, its relevance is computed as follows.

$$relevance_0(C) = \max_{W \in \{W_i : C \sqsubseteq W_i\}} \left(\frac{progress(W)}{|W|} \right)$$

The assumption is that a watchlist W that is matched more is more relevant to the current proof search. In our current implementation, the relevance is computed at the time of generation of C and it is not updated afterwards. As future work, we propose to also update the relevance of all generated but not yet processed clauses from time to time in order to reflect updates of the watchlist progress counters. Note that this is expensive, as the number of generated clauses is typically high. Suitable indexing could be used to lower this cost or even to do the update immediately just for the affected clauses.

To use the watchlist relevance in E, we extend E's domain-specific language for search heuristics with two priority functions `PreferWatchlistRelevant` and `DeferWatchlistRelevant`. The first priority function ranks higher the clauses with higher watchlist relevance², and the other function does the opposite. These priority functions can be used to build E's heuristics just like in the case of the static watchlist guidance. As a results, we can instruct E to process watchlist-relevant clauses in advance.

5.3 Inherited Dynamic Watchlist Relevance

The previous standard watchlist relevance prioritizes only clauses subsuming watchlist clauses but it behaves indifferently with respect to other clauses. In

¹ Alternatively, the subsumed watchlist clause $D \in W_i$ can be considered for future subsumption checks but the watchlist progress counter $progress(W_i)$ should not be increased when D is subsumed again. This is because we want the progress counter to represent the number of *different* clauses from W_i encountered so far.

² Technically, E's priority function returns an integer priority, and clauses with smaller values are preferred. Hence we compute the priority as $1000 * (1 - relevance_0(C))$.

order to provide some guidance even for clauses which do not subsume any watchlist clause, we can examine the watchlist relevance of the parents of each generated clause, and prioritize clauses with watchlist-relevant parents. Let $parents(C)$ denote the set of previously processed clauses from which C have been derived. *Inherited dynamic watchlist relevance*, denoted $relevance_1$, is a combination of the standard dynamic relevance with the average of parents relevances multiplied by a *decay* factor $\delta < 1$.

$$relevance_1(C) = relevance_0(C) + \delta * \operatorname{avg}_{D \in parents(C)} (relevance_1(D))$$

Clearly, the inherited relevance equals to the standard relevance for the initial clauses with no parents. The decay factor (δ) determines the importance of parents watchlist relevances.³ Note that the inherited relevances of $parents(C)$ are already precomputed at the time of generating C , hence no recursive computation is necessary.

With the above $relevance_1$ we compute the average of parents *inherited* relevances, hence the inherited watchlist relevance accumulates relevance of all the ancestors. As a result, $relevance_1(C)$ is greater than 0 if and only if C has some ancestor which subsumed a watchlist clause at some point. This might have an undesirable effect that clauses unrelated to the watchlist are completely ignored during the proof search. In practice, however, it seems important to consider also watchlist-unrelated clauses with some degree in order to prove new conjectures which do not appear on the input watchlist. Hence we introduce two *threshold* parameters α and β which resets the relevance to 0 as follows. Let $length(C)$ denote the length of clause C , counting occurrences of symbols in C .

$$relevance_2(C) = \begin{cases} 0 & \text{iff } relevance_1(C) < \alpha \text{ and } \frac{relevance_1(C)}{length(C)} < \beta \\ relevance_1(C) & \text{otherwise} \end{cases}$$

Parameter α is a threshold on the watchlist inherited relevance while β combines the relevance with the clause length.⁴ As a result, shorter watchlist-unrelated clauses are preferred to longer (distantly) watchlist-related clauses.

6 Experiments with Watchlist Guidance

For our experiments we construct watchlists from the proofs found by E on a benchmark of 57897 Mizar40 [19] problems in the MPTP dataset [35]^{5,6}. These

³ In our experiments, we use $\delta = 0.1$.

⁴ In our experiments, we use $\alpha = 0.03$ and $\beta = 0.009$. These values have been found useful by a small grid search over a random sample of 500 problems.

⁵ Precisely, we have used the small (*bushy*, re-proving) versions, but without ATP minimization. They can be found at http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/MPTP2/problems_small_consist.tar.gz.

⁶ Experimental results and code can be found at <https://github.com/ai4reason/prover-data/tree/master/ITP-18>.

initial proofs were found by an evolutionarily optimized [14] ensemble of 32 E strategies each run for 5s. These are our *baseline* strategies. Due to limited computational resources, we do most of the experiments with the top 5 strategies that (greedily) cover most solutions (*top 5 greedy cover*). These are strategies number 2, 8, 9, 26 and 28, henceforth called *A, B, C, D, E*. In 5s (in parallel) they together solve 21122 problems. We also evaluate these five strategies in 10s, jointly solving 21670 problems. The 21122 proofs yield over 100000 unique proof clauses that can be used for watchlist-based guidance in our experiments. We also use smaller datasets randomly sampled from the full set of 57897 problems to be able to explore more methods. All problems are run on the same hardware⁷ and with the same memory limits.

Each E strategy is specified as a frequency-weighted combination of parameterized *clause evaluation functions* (CEF) combined with a selection of inference rules. Below we show a simplified example strategy specifying the term ordering *KBO*, and combining (with weights 2 and 4) two CEFs made up of weight functions *Clauseweight* and *FIFOWeight* and priority functions *DeferSOS* and *PreferWatchlist*.

```
-tKBO -H(2*Clauseweight(DeferSoS,20,9999,4),4*FIFOWeight(PreferWatchlist))
```

6.1 Watchlist Selection Methods

We have experimented with several methods for creation of static and dynamic watchlists. Typically we use only the proofs found by a particular baseline strategy to construct the watchlists used for testing the guided version of that strategy. Using all 100000+ proof clauses as a watchlist slows E down to 6 given clauses per second. This is comparable to the speed of Prover9 with similarly large watchlists, but there are indexing methods that could speed this up. We have run several smaller tests, but do not include this method in the evaluation due to limited computational resources. Instead, we select a smaller set of clauses. The methods are as follows:

- (**art**) Use all proof clauses from theorems in the problem’s Mizar article⁸. Such watchlist sizes range from 0 to 4000, which does not cause any significant slowdown of E.
- (**freq**) Use high-frequency proof clauses for static watchlists, i.e., clauses that appear in many proofs.
- (**kNN-st**) Use *k*-nearest neighbor (*k*-NN) learning to suggest useful static watchlists for each problem, based on symbol and term-based features [20] of the conjecture. This is very similar to the standard use of *k*-NN and other learners for premise selection. In more detail, we use symbols, walks of length 2 on formula trees and common subterms (with variables and skolem symbols unified). Each proof is turned into a multi-label training example, where the labels are

⁷ Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30 GHz with 256G RAM.

⁸ Excluding the current theorem.

the (serially numbered) clauses used in the proof, and the features are extracted from the conjecture.

(kNN-dyn) Use k -NN in a similar way to suggest the most related proofs for dynamic watchlists. This is done in two iterations.

- (i) In the first iteration, only the conjecture-based similarity is used to select related problems and their proofs.
- (ii) The second iteration then uses data mined from the proofs obtained with dynamic guidance in the first iteration. From each such proof P we create a training example associating P 's conjecture features with the names of the proofs that matched (i.e., guided the inference of) the clauses needed in P . On this dataset we again train a k -NN learner, which recommends the most useful related proofs for guiding a particular conjecture.

6.2 Using Watchlists in E Strategies

As described in Sect. 4, watchlist subsumption defines the `PreferWatchlist` priority function that prioritizes clauses that subsume at least one watchlist clause. Below we describe several ways to use this priority function and the newly defined dynamic `PreferWatchlistRelevant` priority function and its relevance-inheriting modifications. Each of them can additionally take the “no-remove” option, to keep subsumed watchlist clauses in the watchlist, allowing repeated matching by different clauses. Preliminary testing has shown that just adding a single watchlist-based clause evaluation function (*CEF*) to the baseline CEFs⁹ is not as good as the methods defined below. In the rest of the paper we provide short names for the methods, such as *prefA* (baseline strategy A modified by the *pref* method described below).

1. *evo*: the default heuristic strategy (Sect. 4) evolved (genetically [27]) for static watchlist use.
2. *pref*: replace all priority functions in a baseline strategy with the `PreferWatchlist` priority function. The resulting strategies look as follows:
`-H(2*Clauseweight(PreferWatchlist,20,9999,4),
4*FIFOweight(PreferWatchlist))`
3. *const*: replace all priority functions in a baseline strategy with `ConstPrio`, which assigns the same priority to all clauses, so all ranking is done by weight functions alone.
4. *uwl*: always prefer clauses that match the watchlist, but use the baseline strategy's priority function otherwise¹⁰.
5. *ska*: modify watchlist subsumption in E to treat all skolem symbols of the same arity as equal, thus widening the watchlist guidance. This can be used with any strategy. In this paper it is used with *pref*.

⁹ Specifically we tried adding `Defaultweight(PreferWatchlist)` and `ConjectureRelativeSymbolWeight(PreferWatchlist)` with frequencies 1, 2, 5, 10, 20 times that of the rest of the CEFs in the strategy.

¹⁰ *uwl* is implemented in E's source code as an option.

6. *dyn*: replace all priority functions in a baseline strategy with `PreferWatchlistRelevant`, which dynamically weights watchlist clauses (Sect. 5.2).
7. *dyndec*: add the relevance inheritance mechanisms to *dyn* (Sect. 5.3).

6.3 Evaluation

First we measure the slowdown caused by larger static watchlists on the best baseline strategy and a random sample of 10000 problems. The results are shown in Table 1. We see that the speed significantly degrades with watchlists of size 10000, while 500-big watchlists incur only a small performance penalty.

Table 1. Tests of the watchlist size influence (ordered by frequency) on a random sample of 10000 problems using the “no-remove” option and one static watchlist with strategy *prefA*. PPS is average processed clauses per second, a measure of E’s speed.

Size	10	100	256	512	1000	10000
Proved	3275	3275	3287	3283	3248	2912
PPS	8935	9528	8661	7288	4807	575

Table 2 shows the 10s evaluation of several static and dynamic methods on a random sample of 5000 problems using article-based watchlists (method **art** in Sect. 6.1). For comparison, E’s *auto* strategy proves 1350 of the problems in 10s and its *auto-schedule* proves 1629. Given 50s the *auto-schedule* proves 1744 problems compared to our top 5 cover’s 1964.

The first surprising result is that *const* significantly outperforms the *baseline*. This indicates that the old-style simple E priority functions may do more harm than good if they are allowed to override the more recent and sophisticated weight functions. The *ska* strategy performs best here and a variety of strategies provide better coverage. It’s interesting to note that *ska* and *pref* overlap only on 1893 problems. The original *evo* strategy performs well, but lacks diversity.

Table 2. Article-based watchlist benchmark. A top 5 greedy cover proves 1964 problems (in bold).

Strategy	baseline	const	pref	ska	dyn	evo	uwl
A	1238	1493	1503	1510	1500	1303	1247
B	1255	1296	1315	1330	1316	1300	1277
C	1075	1166	1205	1183	1201	1068	1097
D	1102	1133	1176	1190	1175	1330	1132
E	1138	1141	1141	1153	1139	1070	1139
Total	1853	1910	1931	1933	1922	1659	1868

Table 3 briefly evaluates k -NN selection of watchlist clauses (method **kNN-st** in Sect. 6.1) on a single strategy *prefA*. Next we use k -NN to suggest watchlist proofs¹¹ (method **kNN-dyn.i**) for *pref* and *dyn*. Table 4 evaluates the influence of the number of related proofs loaded for the dynamic strategies. Interestingly, *pref* outperforms *dyn* almost everywhere but *dyn*'s ensemble of strategies A-E generally performs best and the top 5 cover is better. We conclude that *dyn*'s dynamic relevance weighting allows the strategies to diversify more.

Table 3. Evaluation of **kNN-st** on *prefA*

Watchlist size	16	64	256	1024	2048
Proved	1518	1531	1528	1532	1520

Table 5 evaluates the top 5 greedy cover from Table 4 on the full Mizar dataset, already showing significant improvement over the 21670 proofs produced by the 5 baseline strategies. Based on proof data from a full-run of the top-5 greedy cover in Table 5, new k -NN proof suggestions were made (method **kNN-dyn.ii**) and *dyn*'s grid search re-run, see Table 6 and Table 7 for k -NN round 2 results.

We also test the relevance inheriting dynamic watchlist feature (*dyndec*), primarily to determine if different proofs can be found. The results are shown in Table 8. This version adds 8 problems to the top 5 greedy cover of all the strategies run on the 5000 problem dataset, making it useful in a schedule despite lower performance alone. Table 9 shows this greedy cover, and then its evaluation on the full dataset. The 23192 problems proved by our new greedy cover is a 7% improvement over the top 5 baseline strategies.

7 Examples

The Mizar theorem **YELLOW_5:36**¹² states De Morgan's laws for Boolean lattices:

```
theorem Th36: :: YELLOW_5:36
for L being non empty Boolean RelStr for a, b being Element of L
holds ( 'not' ( a "∨" b ) = ( 'not' a ) "∧" ( 'not' b )
      & 'not' ( a "∧" b ) = ( 'not' a ) "∨" ( 'not' b ) )
```

Using 32 related proofs results in 2220 clauses placed on the watchlists. The dynamically guided proof search takes 5218 (nontrivial) given clause loops done in 2s and the resulting ATP proof is 436 inferences long. There are 194 given clauses that match the watchlist during the proof search and 120 (61.8%) of them end up being part of the proof. I.e., 27.5% of the proof consists of steps guided by the watchlist mechanism. The proof search using the same settings,

¹¹ All clauses in suggested proofs are used.

¹² http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/html/yellow_5#T36.

Table 4. k-NN proof recommendation watchlists (**kNN-dyn.i**) for *dyn pref*. Size is number of proofs, averaging 40 clauses per proof. A top 5 greedy cover of *dyn* proves 1972 and *pref* proves 1959 (in bold).

Size	dynA	dynB	dynC	dynD	dynE	Total
4	1531	1352	1235	1194	1165	1957
8	1543	1366	1253	1188	1170	1956
16	1529	1357	1224	1218	1185	1951
32	1546	1373	1240	1218	1188	1962
64	1535	1376	1216	1215	1166	1935
128	1506	1351	1195	1214	1147	1907
1024	1108	963	710	943	765	1404
Size	prefA	prefB	prefC	prefD	prefE	Total
4	1539	1369	1210	1220	1159	1944
8	1554	1385	1219	1240	1168	1941
16	1572	1405	1225	1254	1180	1952
32	1568	1412	1231	1271	1190	1958
64	1567	1402	1228	1262	1172	1952
128	1552	1388	1210	1248	1160	1934
1024	1195	1061	791	991	806	1501

Table 5. K-NN round 1 greedy cover on full dataset and proofs added by each successive strategy for a total of 22579. *dynA_32* means strategy *dynA* using 32 proof watchlists.

	dynA_32	dynC_8	dynD_16	dynE_4	dynB_64
Added	17964	2531	1024	760	282
Total	17964	14014	14294	13449	16175

Table 6. Problems proved by round 2 k-NN proof suggestions (**kNN-dyn.ii**). The top 5 greedy cover proves 1981 problems (in bold). *dyn2A* means *dynA* run on the 2nd iteration of k-NN suggestions.

Size	dyn2A	dyn2B	dyn2C	dyn2D	dyn2E	Total	Round 1 total
4	1539	1368	1235	1209	1179	1961	1957
8	1554	1376	1253	1217	1183	1971	1956
16	1565	1382	1256	1221	1181	1972	1951
32	1557	1383	1252	1227	1182	1968	1962
64	1545	1385	1244	1222	1171	1963	1935
128	1531	1374	1221	1227	1171	1941	1907

Table 7. K-NN round 2 greedy cover on full dataset and proofs added by each successive strategy for a total of 22996

	dyn2A_16	dyn2C_16	dyn2D_32	dyn2E_4	dyn2B_4
Total	18583	14486	14720	13532	16244
Added	18583	2553	1007	599	254

Table 8. Problems proved by round 2 k-NN proof suggestions with *dyndec*. The top 5 greedy cover proves 1898 problems (in bold).

Size	dyndec2A	dyndec2B	dyndec2C	dyndec2D	dyndec2E	Total
4	1432	1354	1184	1203	1152	1885
16	1384	1316	1176	1221	1140	1846
32	1381	1309	1157	1209	1133	1820
128	1326	1295	1127	1172	1082	1769

Table 9. Top: Cumulative sum of the 5000 test set greedy cover. The k-NN based dynamic watchlist methods dominate, improving by 2.1% over the baseline and article-based watchlist strategy greedy cover of 1964 (Table 2). Bottom: Greedy cover run on the full dataset, cumulative and total proved.

Total	dyn2A_16	dyn2C_16	dyndec2D_16	dyn2E_4	dyndec2A_128
2007	1565	230	97	68	47
23192	18583	2553	1050	584	422
23192	18583	14486	14514	13532	15916

but without the watchlist takes 6550 nontrivial given clause loops (25.5% more). The proof of the theorem `WAYBEL_1:85`¹³ is considerably used for this guidance:

```
theorem :: WAYBEL_1:85
for H being non empty lower-bounded RelStr st H is Heyting holds
for a, b being Element of H holds 'not' (a "∧" b) >= ('not' a) "∨" ('not' b)
```

Note that this proof is done under the weaker assumptions of H being lower bounded and Heyting, rather than being Boolean. Yet, 62 (80.5%) of the 77 clauses from the proof of `WAYBEL_1:85` are eventually matched during the proof search. 38 (49.4%) of these 77 clauses are used in the proof of `YELLOW_5:36`. In Table 10 we show the final state of proof progress for the 32 loaded proofs after the last non empty clause matched the watchlist. For each we show both the computed ratio and the number of matched and all clauses.

An example of a theorem that can be proved in 1.2s with guidance but cannot be proved in 10s with any unguided method is the following theorem `BOOLEALG:62`¹⁴ about the symmetric difference in Boolean lattices:

¹³ http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/html/waybel_1#T85.

¹⁴ <http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/html/boolealg#T62>.

Table 10. Final state of the proof progress for the (serially numbered) 32 proofs loaded to guide the proof of YELLOW_5:36. We show the computed ratio and the number of matched and all clauses.

0	0.438	42/96	1	0.727	56/77	2	0.865	45/52	3	0.360	9/25
4	0.750	51/68	5	0.259	7/27	6	0.805	62/77	7	0.302	73/242
8	0.652	15/23	9	0.286	8/28	10	0.259	7/27	11	0.338	24/71
12	0.680	17/25	13	0.509	27/53	14	0.357	10/28	15	0.568	25/44
16	0.703	52/74	17	0.029	8/272	18	0.379	33/87	19	0.424	14/33
20	0.471	16/34	21	0.323	20/62	22	0.333	7/21	23	0.520	26/50
24	0.524	22/42	25	0.523	45/86	26	0.462	6/13	27	0.370	20/54
28	0.411	30/73	29	0.364	20/55	30	0.571	16/28	31	0.357	10/28

```
for L being B_Lattice
for X, Y being Element of L holds (X \+ Y) \+ (X "^" Y) = X "\v" Y
```

Using 32 related proofs results in 2768 clauses placed on the watchlists. The proof search then takes 4748 (nontrivial) given clause loops and the watchlist-guided ATP proof is 633 inferences long. There are 613 given clauses that match the watchlist during the proof search and 266 (43.4%) of them end up being part of the proof. I.e., 42% of the proof consists of steps guided by the watchlist mechanism. Among the theorems whose proofs are most useful for the guidance are the following theorems LATTICES:23¹⁵, BOOLEALG:33¹⁶ and BOOLEALG:54¹⁷ on Boolean lattices:

```
theorem Th23: :: LATTICES:23
for L being B_Lattice
for a, b being Element of L holds (a "^" b)' = a' "\v" b'

theorem Th33: :: BOOLEALG:33
for L being B_Lattice for X, Y being Element of L holds X \ (X "^" Y) = X \ Y

theorem :: BOOLEALG:54
for L being B_Lattice for X, Y being Element of L
st X' "\v" Y' = X "\v" Y & X misses X' & Y misses Y'
holds X = Y' & Y = X'
```

Finally, we show several theorems^{18,19,20,21} with nontrivial Mizar proofs and relatively long ATP proofs obtained with significant guidance. These theorems cannot be proved by any other method used in this work.

¹⁵ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/lattices#T23.

¹⁶ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/boolealg#T33.

¹⁷ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/boolealg#T54.

¹⁸ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/boolealg#T68.

¹⁹ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/closure1#T21.

²⁰ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/bcialg_4#T44.

²¹ http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/xxreal_3#T67.

```

theorem :: BOOLEALG:68
for L being B_Lattice for X, Y being Element of L
holds (X \+\ Y)' = (X "\^" Y) "\v" ((X') "\^" (Y'))

theorem :: CLOSURE1:21
for I being set for M being ManySortedSet of I
for P, R being MSSetOp of M st P is monotonic & R is monotonic
holds P ** R is monotonic

theorem :: BCIALG_4:44
for X being commutative BCK-Algebra_with_Condition(S)
for a, b, c being Element of X st Condition_S (a,b) c= Initial_section c holds
for x being Element of Condition_S (a,b) holds x <= c \ ((c \ a) \ b)

theorem :: XXREAL_3:67
for f, g being ext-real number holds (f * g)"=(f") * (g")

```

8 Related Work and Possible Extensions

The closest related work is the hintguidance in Otter and Prover9. Our focus is however on large ITP-style theories with large signatures and heterogeneous facts and proofs spanning various areas of mathematics. This motivates using machine learning for reducing the size of the static watchlists and the implementation of the dynamic watchlist mechanisms. Several implementations of internal proof search guidance using statistical learning have been mentioned in Sects. 1 and 3. In both the tableau-based systems and the tactical ITP systems the statistical learning guidance benefits from a compact and directly usable notion of proof state, which is not immediately available in saturation-style ATP.

By delegating the notion of similarity to subsumption we are relying on fast, crisp and well-known symbolic ATP mechanisms. This has advantages as well as disadvantages. Compared to the ENIGMA [15] and neural [23] statistical guiding methods, the subsumption-based notion of clause similarity is not feature-based or learned. This similarity relation is crisp and sparser compared to the similarity relations induced by the statistical methods. The proof guidance is limited when no derived clauses subsume any of the loaded proof clauses. This can be countered by loading a high number of proofs and widening (or softening) the similarity relation in various approximate ways. On the other hand, subsumption is fast compared to the deep neural methods (see [23]) and enjoys clear guarantees of the underlying symbolic calculus. For example, when all the (non empty) clauses from a loaded related proof have been subsumed in the current proof search, it is clear that the current proof search is successfully finished.

A clear novelty is the focusing of the proof search towards the (possibly implausible) inferences needed for completing the loaded proofs. Existing statistical guiding methods will fail to notice such opportunities, and the static watchlist guidance has no way of distinguishing the watchlist matchers that lead faster to proof completion. In a way this mechanism resembles the feedback obtained by Monte Carlo exploration, where a seemingly statistically unlikely decision can be made, based on many rollouts and averaging of their results. Instead, we rely here on a database of previous proofs, similar to previously

played and finished games. The newly introduced heuristic proof search (proof progress) representation may however enable further experiments with Monte Carlo guidance.

8.1 Possible Extensions

Several extensions have been already discussed above. We list the most obvious.

More Sophisticated Progress Metrics: The current proof-progress criterion may be too crude. Subsuming all the *initial* clauses of a related proof is unlikely until the empty clause is derived. In general, a large part of a related proof may not be needed once the right clauses in the “middle of the proof” are subsumed by the current proof search. A better proof-progress metric would compute the smallest number of proof clauses that are still needed to entail the contradiction. This is achievable, however more technically involved, also due to issues such as rewriting of the watchlist clauses during the current proof search.

Clause Re-evaluation Based on the Evolving Proof Relevance: As more and more watchlist clauses are matched, the proof relevance of the clauses generated earlier should be updated to mirror the current state. This is in general expensive, so it could be done after each N given clause loops or after a significant number of watchlist matchings. An alternative is to add corresponding indexing mechanisms to the set of generated clauses, which will immediately reorder them in the evaluation queues based on the proof relevance updates.

More Abstract/Approximate Matching: Instead of the strict notion of subsumption, more abstract or heuristic matching methods could be used. An interesting symbolic method to consider is matching modulo symbol alignments [9]. A number of approximate methods are already used by the above mentioned statistical guiding methods.

Adding Statistical Methods for Clause Guidance: Instead of using only hard-coded watchlist-style heuristics for focusing inferences, a statistical (e.g. ENIGMA-style) clause evaluation model could be trained by adding the vector of proof completion ratios to the currently used ENIGMA features.

9 Conclusion

The portfolio of new proof guiding methods developed here significantly improves E’s standard portfolio of strategies, and also the previous best set of strategies invented for Mizar by evolutionary methods. The best combination of five new strategies run in parallel for 10 s (a reasonable hammering time) will prove over 7% more Mizar problems than the previous best combination of five non-watchlist strategies. Improvement over E’s standard portfolio is much higher. Even though we focus on developing the strongest portfolio rather than a single best method, it is clear that the best guided versions also significantly improve over their non-guided counterparts. This improvement for the best new strategy

(`dyn2A` used with 16 most relevant proofs) is 26.5% (=18583/14693). These are relatively high improvements in automated theorem proving.

We have shown that the new dynamic methods based on the idea of proof completion ratios improve over the static watchlist guidance. We have also shown that as usual with learning-based guidance, iterating the methods to produce more proofs leads to stronger methods in the next iteration. The first experiments with widening the watchlist-based guidance by relatively simple inheritance mechanisms seem quite promising, contributing many new proofs. A number of extensions and experiments with guiding saturation-style proving have been opened for future research. We believe that various extensions of the compact and evolving heuristic representation of saturation-style proof search as introduced here will turn out to be of great importance for further development of learning-based saturation provers.

Acknowledgments. We thank Bob Veroff for many enlightening explanations and discussions of the watchlist mechanisms in `Otter` and `Prover9`. His “industry-grade” projects that prove open and interesting mathematical conjectures with hints and proof sketches have been a great sort of inspiration for this work.

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsvitsivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning* **52**(2), 191–213 (2014)
2. Alemi, A.A., Chollet, F., Eén, N., Irving, G., Szegedy, C., Urban, J.: DeepMath - deep sequence models for premise selection. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, 5–10 December 2016, Barcelona, Spain, pp. 2235–2243 (2016)
3. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *J. Logic Comput.* **3**(4), 217–247 (1994)
4. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning* **57**(3), 219–244 (2016)
5. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. *J. Formalized Reasoning* **9**(1), 101–148 (2016)
6. Eiter, T., Sands, D. (eds.): *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, Maun, Botswana, 7–12 May 2017, *EPiC Series in Computing*, vol. 46. EasyChair (2017)
7. Färber, M., Kaliszyk, C.: Random forests for premise selection. In: Lutz, C., Ranise, S. (eds.) *FroCoS 2015. LNCS (LNAI)*, vol. 9322, pp. 325–340. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24246-0_20
8. Färber, M., Kaliszyk, C., Urban, J.: Monte carlo tableau proof search. In: de Moura, L. (ed.) *CADE 2017. LNCS (LNAI)*, vol. 10395, pp. 563–579. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-63046-5_34
9. Gauthier, T., Kaliszyk, C.: Matching concepts across HOL libraries. In: Watt, S.M., Davenport, J.H., Sexton, A.P., Sojka, P., Urban, J. (eds.) *CICM 2014. LNCS (LNAI)*, vol. 8543, pp. 267–281. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-08434-3_20

10. Gauthier, T., Kaliszyk, C., Urban, J.: TacticToe: learning to reason with HOL4 tactics. In: Eiter and Sands [6], pp. 125–143
11. Grabowski, A., Kornilowicz, A., Naumowicz, A.: Mizar in a nutshell. *J. Formalized Reasoning* **3**(2), 153–245 (2010)
12. Gransden, T., Walkinshaw, N., Raman, R.: SEPIA: search for proofs using inferred automata. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 246–255. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_16
13. Jakubův, J., Urban, J.: Extending E prover with similarity based clause selection strategies. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) CICM 2016. LNCS (LNAI), vol. 9791, pp. 151–156. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_11
14. Jakubův, J., Urban, J.: BliStrTune: hierarchical invention of theorem proving strategies. In: Bertot, Y., Vafeiadis, V. (eds.) Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, 16–17 January 2017, pp. 43–52. ACM (2017)
15. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS (LNAI), vol. 10383, pp. 292–302. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_20
16. Kaliszyk, C., Schulz, S., Urban, J., Vyskočil, J.: System description: E.T. 0.1. In: Felty, A.P., Middeldorp, A. (eds.) CADE 2015. LNCS (LNAI), vol. 9195, pp. 389–398. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-21401-6_27
17. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning* **53**(2), 173–213 (2014)
18. Kaliszyk, C., Urban, J.: FEMaLeCoP: fairly efficient machine learning connection prover. In: Davis, M., Fehnker, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 88–96. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_7
19. Kaliszyk, C., Urban, J.: MizAR 40 for Mizar 40. *J. Autom. Reasoning* **55**(3), 245–256 (2015)
20. Kaliszyk, C., Urban, J., Vyskočil, J.: Efficient semantic features for automated reasoning over large theories. In: Yang, Q., Wooldridge, M. (eds.) IJCAI 2015, pp. 3084–3090. AAAI Press (2015)
21. Kinyon, M., Veroff, R., Vojtěchovský, P.: Loops with abelian inner mapping groups: an application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS (LNAI), vol. 7788, pp. 151–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_8
22. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
23. Loos, S.M., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. In: Eiter and Sands [6], pp. 85–105
24. McCune, W., Wos, L.: Otter: the CADE-13 competition incarnations. *J. Autom. Reasoning* **18**(2), 211–220 (1997). Special Issue on the CADE 13 ATP System Competition
25. McCune, W.W.: Prover9 and Mace4 (2005–2010). <http://www.cs.unm.edu/~mccune/prover9/>. Accessed 29 Mar 2016
26. Otten, J., Bibel, W.: leanCoP: lean connection-based theorem proving. *J. Symb. Comput.* **36**(1–2), 139–161 (2003)

27. Schäfer, S., Schulz, S.: Breeding theorem proving heuristics with genetic algorithms. In: Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.) *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, 16–19 October 2015, EPiC Series in Computing*, vol. 36, pp. 263–274. EasyChair (2015)
28. Schulz, S.: Learning search control knowledge for equational theorem proving. In: Baader, F., Brewka, G., Eiter, T. (eds.) *KI 2001. LNCS (LNAI)*, vol. 2174, pp. 320–334. Springer, Heidelberg (2001). https://doi.org/10.1007/3-540-45422-5_23
29. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: Bonacina, M.P., Stickel, M.E. (eds.) *Automated Reasoning and Mathematics. LNCS (LNAI)*, vol. 7788, pp. 45–67. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_3
30. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) *LPAR 2013. LNCS*, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_49
31. Schulz, S., Möhrmann, M.: Performance of clause selection heuristics for saturation-based theorem proving. In: Olivetti, N., Tiwari, A. (eds.) *IJCAR 2016. LNCS (LNAI)*, vol. 9706, pp. 330–345. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_23
32. Silver, D., Huang, A., Maddison, C.J., Guez, A., Sifre, L., van den Driessche, G., Schrittwieser, J., Antonoglou, I., Panneershelvam, V., Lanctot, M., Dieleman, S., Grewe, D., Nham, J., Kalchbrenner, N., Sutskever, I., Lillicrap, T.P., Leach, M., Kavukcuoglu, K., Graepel, T., Hassabis, D.: Mastering the game of go with deep neural networks and tree search. *Nature* **529**(7587), 484–489 (2016)
33. Silver, D., Hubert, T., Schrittwieser, J., Antonoglou, I., Lai, M., Guez, A., Lanctot, M., Sifre, L., Kumaran, D., Graepel, T., Lillicrap, T.P., Simonyan, K., Hassabis, D.: Mastering Chess and Shogi by self-play with a general reinforcement learning algorithm. *CoRR*, abs/1712.01815 (2017)
34. Slind, K., Norrish, M.: A brief overview of HOL4. In: Mohamed, O.A., Muñoz, C., Tahar, S. (eds.) *TPHOLs 2008. LNCS*, vol. 5170, pp. 28–32. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71067-7_6
35. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. *J. Autom. Reasoning* **37**(1–2), 21–43 (2006)
36. Urban, J., Vyskočil, J., Štěpánek, P.: MaLeCoP machine learning connection prover. In: Brünner, K., Metcalfe, G. (eds.) *TABLEAUX 2011. LNCS (LNAI)*, vol. 6793, pp. 263–277. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22119-4_21
37. Veroff, R.: Using hints to increase the effectiveness of an automated reasoning program: case studies. *J. Autom. Reasoning* **16**(3), 223–239 (1996)



ProofWatch Meets ENIGMA: First Experiments

Zarathustra Goertzel, Jan Jakubův, and Josef Urban*

Czech Technical University in Prague

Abstract

Watchlist (also hint list) is a technique that allows lemmas from related proofs to guide a saturation-style proof search for a new conjecture. ProofWatch is a recent watchlist-style method that loads many previous proofs inside the ATP, maintains their completion ratios during the proof search and focuses the search by following the most completed proofs. In this work, we start to use the dynamically changing vector of proof completion ratios as additional information about the saturation-style proof state for statistical machine learning methods that evaluate the generated clauses. In particular, we add the proof completion vectors to ENIGMA (efficient learning-based inference guiding machine) and evaluate the new method on the MPTP Challenge benchmark, showing moderate improvement in E’s performance over ProofWatch and ENIGMA.

1 Introduction

This work proposes and develops a new learning-based proof guidance – *ENIGMAWatch* – for saturation-style first-order theorem provers. It is based on two previous guiding methods implemented for the E [13] ATP system: ProofWatch [4] and ENIGMA [7, 8]. Both ProofWatch and ENIGMA enable E to use related proofs for guiding the proof search for a new conjecture. ProofWatch is based on the hints (watchlist) mechanism. It uses standard symbolic subsumption to compute the completion ratios of related proofs, and focuses the current proof search towards the most completed ones. ENIGMA uses statistical machine learning from many related proofs to estimate the relevance of the generated clauses for the current conjecture. ENIGMAWatch combines the two approaches by using the completion ratios of the related proofs as an additional characterization of the current proof state, which is used together with the conjecture for ENIGMA-style machine learning of clause relevance.

ENIGMAWatch is implemented for E and evaluated here on the MPTP Challenge¹ [14, 15] benchmark. This is a set of 252 first-order problems extracted from the Mizar Mathematical Library (MML) [5]. This set of lemmas is used in Mizar to prove the Bolzano-Weierstrass theorem.²

*Supported by the *AI4REASON* ERC Consolidator grant number 649043, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15.003/0000466 and the European Regional Development Fund.

¹<http://tptp.cs.miami.edu/~tptp/MPTPChallenge/>

²http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/yellow19.html#T36

The rest of the paper is organized as follows. Sections 2 and 3 provide the background on ENIGMA and ProofWatch. Section 4 explains how ENIGMA and ProofWatch are combined, and Section 5 evaluates ENIGMAWatch on the MPTP Challenge benchmark.

2 Saturation-based ATP and ENIGMA

Saturation-style first-order theorem provers are based on the *given-clause algorithm*. This algorithm splits the proof state into two subsets of clauses, the initially empty *processed clauses* P and the *unprocessed clauses* U . In each step the algorithm picks one unprocessed clause g (the *given clause*), puts g into P , and performs all possible inferences between g and the clauses in P . The newly generated clauses are, if not trivially discarded, put into U . This process is repeated until U is empty or a proof (contradiction) has been found. The core choice point is the selection of the next given clause.

ENIGMA [7, 8] stands for *Efficient learning-based Inference Guiding MACHine*. It steers the selection of the given clauses in saturation-based ATPs like E. ENIGMA is based on the simple but fast *logistic regression* algorithm [2] effectively implemented by the LIBLINEAR open source library [3]. In order to employ logistic regression, first-order clauses need to be translated to fixed-length numeric *feature vectors*. The first version of ENIGMA [7] uses (top-down-)oriented term-tree walks of length 3 as *features*. For example, a unit clause “ $P(f(a, b))$ ” contains only features “ (P, f, a) ” and “ (P, f, b) ” (see [7, Sec. 3.2] for details). Features are enumerated and a clause C is translated to the feature vector φ_C whose i -th member counts the number of occurrences of the i -th feature in clause C .

In order to train an ENIGMA *predictor* \mathcal{E} , all the given clauses \mathcal{C} from a set of previous successful proof searches are collected. The given clauses used in the proofs are classified as positive ($\mathcal{C}^+ \subseteq \mathcal{C}$) and the remaining given clauses as negative ($\mathcal{C}^- \subseteq \mathcal{C}$). The clause sets ($\mathcal{C}^+, \mathcal{C}^-$) are turned into feature vector sets (Φ^+, Φ^-) using a fixed *feature enumeration* π . Then a LIBLINEAR *classifier* w (a *weight vector*) is trained on the classification (Φ^+, Φ^-), classifying each clause as *useful* or *un-useful*. The classifier w and enumeration π produce a predictor $\mathcal{E} = (w, \pi)$ which is used to guide next proof searches in combination with other E heuristics.

The above ENIGMA predictors recommend clauses independently of the conjecture being currently proved. The second version of ENIGMA [8] overcomes this weakness by adding the *conjecture context*. Instead of representing just the clause C using the vector φ_C of length n (where n is the number of different features appearing in the training data), we use a vector (φ_C, φ_G) of length $2n$ where φ_G contains the features of the conjecture G . For a training clause C , G corresponds to the conjecture of the proof search where C was selected as a given clause. When classifying a clause C during a proof search, G corresponds to the conjecture currently being proved. In this way, ENIGMA provides conjecture-specific predictions. The enhanced ENIGMA additionally supports more features, like *horizontal features* and *static features* (see [8, Sec. 2] for more details).

Even with the above conjecture context, ENIGMA predictors still recommend clauses independently on the current state of the proof search. In this work we add the *proof state context*, that is, instead of representing the choice of a clause C by the vector (φ_C, φ_G) we represent the choice of the clause *in the current proof state* by the vector $(\varphi_C, \varphi_G, \varphi_\Pi)$ where φ_Π is a *proof-state vector* which describes the specific proof search state where C was selected. The next sections describe how we construct the proof vectors.

3 ProofWatch

3.1 Standard Watchlist Guidance

The watchlist (hint list) mechanism steers given clause selection via symbolic matching between generated clauses and clauses on a *watchlist* W . This technique has been developed and used extensively by Veroff [16] for the AIM project [9] with Prover9 [12] for obtaining long and advanced proofs of open conjectures. The standard watchlist mechanism as originally implemented in E, Otter [11], and Prover9 [12] uses only one watchlist W . In E, the watchlist mechanism uses a priority function `PreferWatchlist` that gives higher priority to clauses that match the watchlist W .³ Clauses with higher priority are selected as given before clauses with lower priority⁴. When clauses from previous proofs are put on W , E thus prefers to follow steps from the previous proofs whenever it can.

3.2 ProofWatch

ProofWatch [4, Sec. 5] extends standard watchlist guidance by allowing for multiple watchlists W_1, \dots, W_n , e.g., one corresponding to each related proof used. We say that a generated clause C *matches* a watchlist W if C subsumes a clause $C_W \in W$ (this implies that C logically entails C_W). During a proof search, clauses from some watchlist might get matched more often than clauses from others. The more clauses are matched from a watchlist W_i , the more the current proof search resembles W_i , and hence W_i might be more relevant for this proof search. The idea of ProofWatch is to prioritize clauses that match more relevant watchlists (proofs).

Watchlist relevance is dynamically computed. We define $progress(W)$ to be the count of clauses $C_W \in W$ that have been matched in the proof search thus far. The completion ratio, $\frac{progress(W)}{|W|}$, measures how much of the watchlist W has been matched. The *dynamic relevance* of each generated clause C is defined as the maximum completion ratio over all the watchlists W_i that C matches:⁵

$$relevance(C) = \max_{W \in \{W_i : C \sqsubseteq W_i\}} \left(\frac{progress(W)}{|W|} \right)$$

The higher the dynamic relevance, the higher priority a clause matching that watchlist is given.

4 ENIGMAWatch: ProofWatch meets ENIGMA

The watchlist completion ratios at each step in E’s proof search can be taken as a vectorial representation of the proof state, and used as input to ENIGMA. This is how the *proof-state vector* φ_Π is constructed. The general motivation for this approach is to come up with an *evolving* characterization of the saturation-style proof state, preferably in a vectorial form suitable for machine learning tools. In general, this could be, e.g., a vector of more abstract similarities of the current proof state to other proofs measured in various (possibly approximate) ways. The ProofWatch based proof-state vector is thus the first reasonable implementation of this general idea.

³See [4, Sec. 4 and 6] for details.

⁴Numerically the lower the priority, the better. 0 is the best priority.

⁵ $C \sqsubseteq W_i$ stands for C subsuming a clause from W_i

In particular, the positive \mathcal{C}^+ and negative \mathcal{C}^- given clauses are output along with φ_Π , the proof-state vector at the time of their selection, and used in ENIGMA training.

Table 1 shows a sample proof-state vector based on 32 related proofs⁶ for the Mizar theorem **YELLOW 5:36**⁷ at the end of the proof search. Note that some related proofs, such as #2, were almost fully matched, while others, such as #7 were mostly not matched in the proof search.

0	0.438	42/96	1	0.727	56/77	2	0.865	45/52	3	0.360	9/25
4	0.750	51/68	5	0.259	7/27	6	0.805	62/77	7	0.302	73/242
8	0.652	15/23	9	0.286	8/28	10	0.259	7/27	11	0.338	24/71
12	0.680	17/25	13	0.509	27/53	14	0.357	10/28	15	0.568	25/44
16	0.703	52/74	17	0.029	8/272	18	0.379	33/87	19	0.424	14/33
20	0.471	16/34	21	0.323	20/62	22	0.333	7/21	23	0.520	26/50
24	0.524	22/42	25	0.523	45/86	26	0.462	6/13	27	0.370	20/54
28	0.411	30/73	29	0.364	20/55	30	0.571	16/28	31	0.357	10/28

Table 1: Example of the proof-state vector for the (serially numbered) 32 proofs loaded to guide the proof of **YELLOW_5:36**. The three columns are the watchlist i , the completion ratio of i , and $progress(W_i)/|W_i|$.

5 Evaluation on the MPTP Challenge Benchmark

5.1 MPTP Challenge

The Mizar Mathematical Library (MML) contains over 1000 articles on diverse topics. The MPTP Challenge was chosen as an initial benchmark because it covers 33 articles, is of manageable size, and focuses on a single problem. Thus the proof-state vector is hypothesized to be meaningful without need for curation. The problems range from easy to hard, and the challenge is still unsolved.⁸

In the *bushy* division used here, each problem’s axioms are precisely the ones needed in the human proofs in the MML, thus the premise selection [1] task does not need to be done by the ATP. In 2007, 82% of the 252 bushy problems were solved by 14 ATP systems. Presently we ran Vampire [10] version 4.0, the state of the art theorem prover that performed best on the challenge, for 300 seconds per problem and solved 87% (220/252).

5.2 Results

The experiments are conducted using as the baseline 10 strategies that were previously evolved to perform well as an ensemble on the Mizar problems [6].⁹ These strategies outperform E’s auto-schedule strategy [6]. All benchmarks are run on the same hardware¹⁰, with the same memory limits, and using E prover version 2.1¹¹.

⁶The proofs were chosen via k-NN. See [4, Sec. 6.1] for details.

⁷http://grid01.ciirc.cvut.cz/~mptp/7.13.01_4.181.1147/html/yellow_5#T36

⁸The TPTP version of the Bolzano-Weierstrass theorem and its MML proof was however cross-verified [15].

⁹We care about the problems proven by the union of 10 strategies than the performance of any individual strategy.

¹⁰Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz with 256G RAM.

¹¹Our version of E can be found at <https://github.com/ai4reason/eprover/tree/devel>.

We conduct three benchmarks to see how many more MPTP Challenge problems each method enables E to solve. We first run the baseline strategies, then ProofWatch¹² and ENIGMA using those results. For ENIGMAWatch, a second run of the baseline strategies while recording the proof-state vector φ_{Π} is needed before training the ENIGMA models.

The first two benchmarks run E for 1s and 30s per problem and strategy. As ProofWatch and ENIGMA can slow E down, we run one benchmark using abstract time instead of CPU time. Abstract time is measured by given-clause loops. $T15 + C40000$ means that each problem is run until 40000 given clauses are processed or 15s passes.¹³

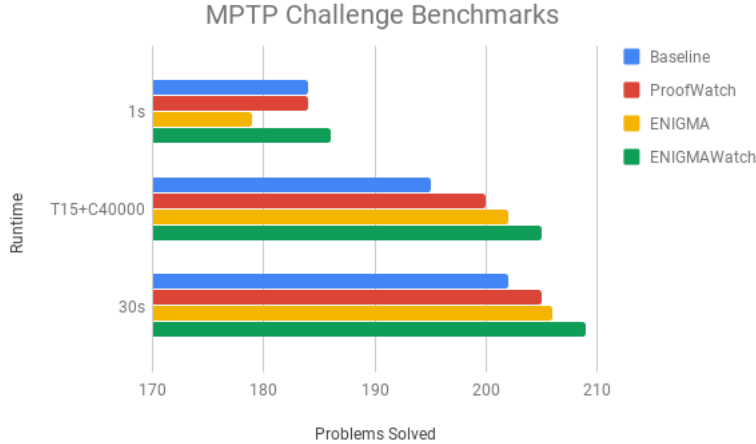


Figure 1: The absolute number of problems solved by each method. In 1s the baseline strategies prove 184 and ENIGMAWatch proves 186. With $T15 + C40000$ the baselines prove 195 and ENIGMAWatch proves 205. In 30s the baselines prove 202 and ENIGMAWatch proves 209.

ENIGMAWatch performs best in all of the benchmarks in Figure 1; however the difference in problems proved is small. As anticipated, the difference is most distinct when using abstract time rather than CPU time.

A performance metric in addition to the number of problems proven is how many given-clause loops E takes to find the proof.¹⁴ This allows given-clause selection strategies to be compared on problems solved.

Figure 2 shows the average number of processed clauses used to find the proof on the $T15 + C40000$ benchmark. ProofWatch cuts the abstract proof-search time by about 75%, while ENIGMA and ENIGMAWatch do significantly better. The ENIGMA-based methods likely have superior efficiency because they provide guidance for each generated clause, whereas ProofWatch only provides guidance when a clause subsumes a watchlist clause. This shows that with proper guidance, E can find proofs much faster. In scenarios where many similar proofs have to be done, this seems useful. The MPTP Challenge is too small to use a standard train/test split¹⁵, so we so far only measure the number of problems additionally proved and

¹²ProofWatch is run with a static watchlist.

¹³The baseline strategies often process this many clauses in 5 – 10s.

¹⁴Which is best measured by looking at non-trivial processed clauses, as E has heuristics for labeling clauses trivial, and checks to see if they are subsumed by another processed clause.

¹⁵One could do leave-one-out testing to test this on this 252 problem dataset.

the proof shortening in terms of the abstract time.

Figure 3 examines how much more efficient ENIGMAWatch is on each problem by taking the average of the ratios:

$$\frac{\text{clauses by ENIGMAWatch on problem } p}{\text{clauses by Baseline on problem } p}$$

The same trend is present as with average clauses, but the outliers seem to stand out less. It’s interesting that with one strategy, *mzr06*, ENIGMA uses more processed clauses per problem than the baseline. However *mzr06* only proved 17 problems, so ENIGMA did not have much training data at its disposal.

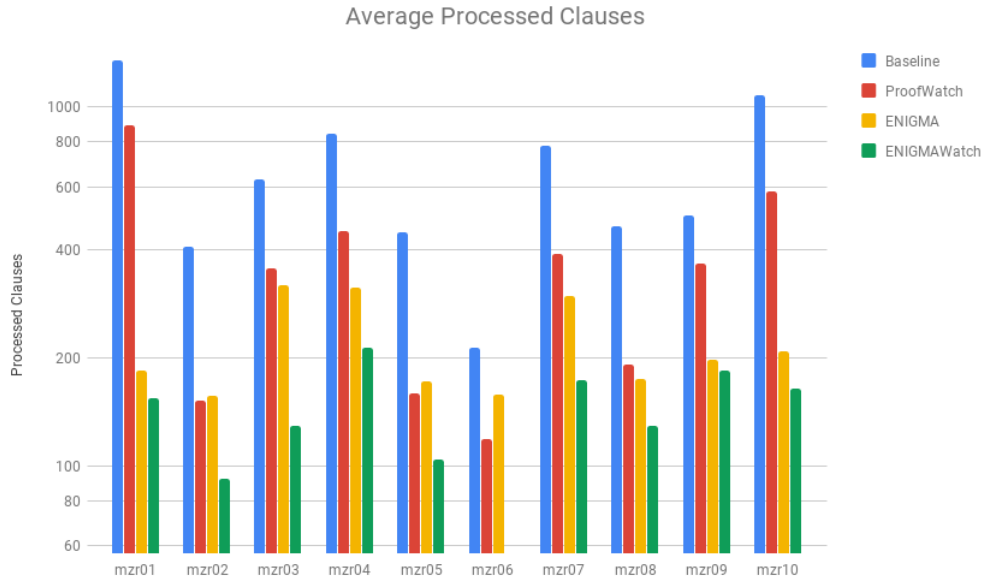


Figure 2: The average processed clauses used to find proofs on the $T15 + C40000$ benchmark.

6 MPTP2078: The Next Frontier

The MPTP2078 benchmark is similar to the MPTP Challenge however all theorems from the 33 Mizar articles are included, growing the number of problems to 2078. In previous work [4], we discovered that ProofWatch becomes slow when there are 10,000 clauses on the watchlist. The inference speed is good enough with up to 128 proofs on the watchlist.¹⁶ The baseline ensemble proves 1461 problems. Thus to use ENIGMAWatch, a small subset of the proofs available must be chosen as the proof-state vector.

We have tried to use k-medoids based on ENIGMA-style features of the problems and their initial clause sets for $k \in \{4, 8, 16, 32, 64\}$ to select the proof-state vectors; however this is not yet effective. The most effective watchlist curation method in ProofWatch is to use k-NN based on

¹⁶Thus the small MPTP Challenge dataset already uses watchlists near the limits of ProofWatch’s capabilities.

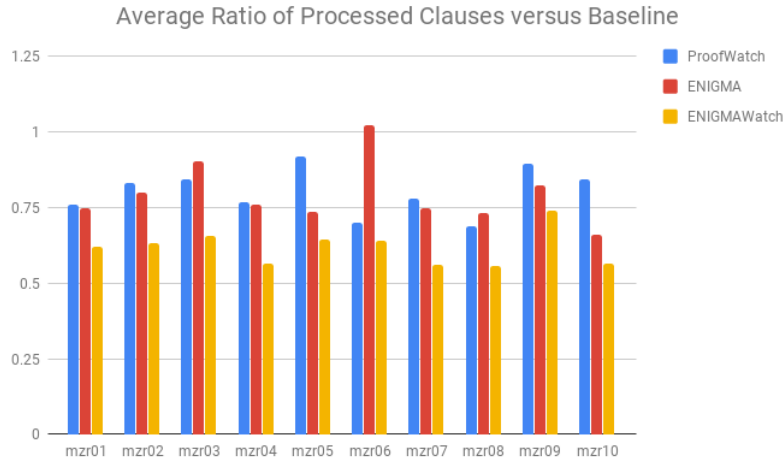


Figure 3: The average ratio of each method over the Baseline, which is a constant bar at 1.

ENIGMA-style features to suggest proofs for each problem [4]. ENIGMAWatch needs to have a consistent proof-state vector, so one idea is to take the union of k-NN suggested proofs and set the unused proofs’ watchlists to zero (the empty clause) on a problem-specific basis. Another option is to have faster algorithms for matching (based on better indexing), for approximate matching or in general for estimating how much a clause belongs to a related proof. The latter ones could again be based on learning such approximate concepts from a large body of proofs.

7 Conclusion

The first experiment with ENIGMAWatch on the MPTP Challenge is encouraging. The performance is better than both ProofWatch and ENIGMA, especially with regard to the number of processed clauses needed to find a proof. This indicates that combining symbolic and statistical machine learning in this way can be fruitful.

However additional work is needed to find out how to best leverage the potential of the ENIGMAWatch method. ProofWatch works best when the watchlists are targeted on the specific problem. ENIGMA, using logistic regression, works best when given lots of data to learn from. Reconciling the two and applying ENIGMAWatch to larger datasets presents interesting research challenges.

References

- [1] J. Alama, T. Heskes, D. Kühlwein, E. Tsvitvadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.
- [2] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT ’92*, pages 144–152, New York, NY, USA, 1992. ACM.
- [3] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.

- [4] Z. Goertzel, J. Jakubův, S. Schulz, and J. Urban. ProofWatch: Watchlist guidance for large theories in E. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018.
- [5] A. Grabowski, A. Kornilowicz, and A. Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.
- [6] J. Jakubův and J. Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Y. Bertot and V. Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017.
- [7] J. Jakubův and J. Urban. ENIGMA: efficient learning-based inference guiding machine. In H. Geuvers, M. England, O. Hasan, F. Rabe, and O. 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.
- [8] J. Jakubův and J. Urban. Enhancing ENIGMA given clause guidance. In F. Rabe, W. M. Farmer, G. O. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.
- [9] M. K. Kinyon, R. Veroff, and P. Vojtechovský. Loops with abelian inner mapping groups: An application of automated deduction. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 151–164. Springer, 2013.
- [10] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
- [11] W. McCune and L. Wos. Otter: The CADE-13 Competition Incarnations. *Journal of Automated Reasoning*, 18(2):211–220, 1997. Special Issue on the CADE 13 ATP System Competition.
- [12] W. W. McCune. Prover9 and Mace4. <http://www.cs.unm.edu/~mccune/prover9/>, 2005–2010. (accessed 2016-03-29).
- [13] S. Schulz. System description: E 1.8. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.
- [14] J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.
- [15] J. Urban and G. Sutcliffe. ATP cross-verification of the Mizar MPTP Challenge problems. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 546–560, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [16] R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *Journal of Automated Reasoning*, 16(3):223–239, 1996.



ENIGMAWatch: ProofWatch Meets ENIGMA

Zarathustra Goertzel, Jan Jakubův, and Josef Urban^{([✉](#))}

Czech Technical University in Prague, Prague, Czech Republic
josef.urban@gmail.com

Abstract. In this work we describe a new learning-based proof guidance – ENIGMAWatch – for saturation-style first-order theorem provers. ENIGMAWatch combines two guiding approaches for the given-clause selection implemented for the E ATP system: ProofWatch and ENIGMA. ProofWatch is motivated by the watchlist (hints) method and based on symbolic matching of multiple related proofs, while ENIGMA is based on statistical machine learning. The two methods are combined by using the evolving information about symbolic proof matching as additional characterization of the saturation-style proof search for the statistical learning methods. The new system is evaluated on a large set of problems from the Mizar library. We show that the added proof-matching information is considered important by the statistical machine learners, and that it leads to improved performance over ProofWatch and ENIGMA.

1 Introduction

This work describes a new learning-based proof guidance – *ENIGMAWatch* – for saturation-style first-order theorem provers. ENIGMAWatch¹ is the combination of two previous guidance methods implemented for the E theorem prover [35]: ProofWatch [11] and ENIGMA [16, 17]. Both ProofWatch and ENIGMA learn to guide E’s proof search for a new conjecture based on related proofs.

ProofWatch uses the hints (watchlist) mechanism, which is a form of precise symbolic memory that can allow inference chains done in a former proof to be replayed in the current proof search. It uses standard symbolic subsumption to check which clauses subsume clauses in related proofs. In addition to boosting the priority of these clauses, the completion ratios of the related proofs are computed, and the proof search is biased towards the most completed ones.

ENIGMA uses fast statistical machine learning to learn from related proof-searches to identify good and bad (positive and negative) clauses for the current

¹ The E version used in this paper can be found at <https://github.com/ai4reason/e-prover/tree/devel>, and the library for running ENIGMA with E can be found at <https://github.com/ai4reason/enigma>.

J. Urban—Supported by the *AI4REASON* ERC Consolidator grant number 649043, and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15-003/0000466 and the European Regional Development Fund.

conjecture. ENIGMA chooses the given clauses based only on features of the problem’s conjecture, which is static throughout the whole proof search. This seems suboptimal: as the proof search evolves, information about the work done so far should influence the selection of the next given clauses.

ENIGMAWatch combines the two approaches by giving the ENIGMA’s learner the ProofWatch completion ratios of the related proofs as an evolving vectorial characterization of the current proof search state. This allows E’s machine learning guidance to have more information about how the proof search is unfolding.

An early version of ENIGMAWatch was tested on the MPTP Challenge² [36, 39] benchmark. It contains 252 first-order problems extracted from the Mizar Mathematical Library (MML) [14], used in Mizar to prove the Bolzano-Weierstrass theorem. Initially, ENIGMAWatch could not be run on a larger dataset, such as the 57897 Mizar40 [21] benchmark, in a reasonable time. Since then, ENIGMA implemented dimensionality reduction using feature hashing [6], extending its applicability to large corpora. We have additionally improved watchlist mechanism in E through enhanced indexing, first time presented in this work in Sect. 4. This allows also ENIGMAWatch to be applied to larger corpora.

The rest of the paper is organized as follows. Section 2 provides an introduction to saturation-based theorem proving and briefly describes ENIGMA and ProofWatch. Section 3 explains how ENIGMA and ProofWatch are combined into ENIGMAWatch, and how watchlists can be selected. Section 4 describes our improved watchlist indexing in E. Both ENIGMAWatch and the improved watchlist indexing are evaluated in Sect. 5.

2 Guiding the Given Clause Selection in ATPs

2.1 Automated Theorem Proving and Machine Learning

State-of-the-art saturation-based automated theorem provers (ATPs) for first-order logic (FOL), such as E [33] and Vampire [25] employ the *given clause algorithm*, translating the input FOL problem $T \cup \{\neg C\}$ into a refutationally equivalent set of clauses. The search for a contradiction is performed maintaining sets of *processed* (P) and *unprocessed* (U) clauses (the *proof state II*). The algorithm repeatedly selects a *given clause* g from U , moves g to P , and extends U with all clauses inferred with g and P . This process continues until a contradiction is found, U becomes empty, or a resource limit is reached.

The search space of this loop grows quickly and it is a well-known fact that the selection of the right given clause is crucial for success. Machine learning from a large number of proofs and proof searches [1–4, 7–10, 15, 16, 19, 20, 22, 26, 29, 31, 32, 38, 40, 41] may help guide the selection of the given clauses.

² <http://tptp.cs.miami.edu/~tptp/MPTPChallenge/>.

2.2 ENIGMA: Learning from Successful Proof Searches

ENIGMA [6, 16–18] (*Efficient learNing-based Internal Guidance MACHine*) is our method for guiding given clause selection in saturation-based ATPs. The method needs to be efficient because it is internally applied to every generated clause. ENIGMA uses E’s capability to analyze successful proof searches, and to output lists of given clauses annotated as either *positive* or *negative* training examples. Each processed clause which is present in the final proof is classified as positive. On the other hand, processing of clauses not present in the final proof was redundant, hence they are classified as negative. ENIGMA’s goal is to learn such classification (possibly conditioned on the problem and its features) in a way that generalizes and allows solving new related problems.

ENIGMA Learning and Models. Given a set of problems \mathcal{P} , we can run E with a strategy \mathcal{S} and obtain positive and negative training data \mathcal{T} from each of the successful proof searches. Various machine learning methods can be used to learn the clause classification given by \mathcal{T} , each method yielding a *classifier* or a (classification) *model* \mathcal{M} . In order to use the model \mathcal{M} in E, \mathcal{M} is used as a function that computes clause weights. This weight function is then used to guide future E runs.

First-order clauses need to be represented in a format recognized by the selected learning method. While neural networks have been very recently practically used for internal guidance with ENIGMA [6], the strongest setting currently uses manually engineered *clause features* and fast non-neural state-of-the-art gradient boosted trees libraries such as XGBoost [5]. The model \mathcal{M} produced by XGBoost consists of a set (*ensemble* [30]) of decision trees. Given a clause C , the model \mathcal{M} yields the probability that C represents a positive clause. When using \mathcal{M} as a weight function in E, the probabilities are turned into binary classification, assigning weight 1.0 for probabilities ≥ 0.5 and weight 10.0 otherwise.

Clause Features. Clause features represent a finite set of various syntactic properties of clauses, and are used to encode clauses by a fixed-length numeric vector. Various machine learning methods can handle numeric vectors and their success heavily depends on the selection of correct clause features. Various possible choices of efficient clause features for theorem prover guidance have been experimented with [16, 17, 22, 23]. The original ENIGMA [16] uses term-tree walks of length 3 as features, while the second version [17] reaches better results by employing various additional features.

Since there are only finitely many features in any training data, the features can be serially numbered. This numbering is fixed for each experiment. Let n be the number of different features appearing in the training data. A clause C is translated to a feature vector φ_C whose i -th member counts the number of occurrences of the i -th feature in C . Hence every clause is represented by a sparse numeric vector of length n . Additionally, we embed information about the conjecture currently being proved in the feature vector, yielding vectors of length $2n$. See [6, 17] for more details.

Feature Hashing. Experiments revealed that XGBoost is capable of dealing with vectors up to the length of 10^5 with a reasonable performance. In experiments with the whole translated Mizar Mathematical Library, the feature vector length can easily grow over 10^6 . This significantly increases both the training and the clause evaluation times. To handle such larger data sets, a simple *hashing* method has previously been implemented to decrease the dimension of the vectors.

Instead of serially numbering all features, we represent each feature f by a unique string and apply a general-purpose string hashing function to obtain a number n_f within a required range (between 0 and an adjustable *hash base*). The value of f is then stored in the feature vector at the position n_f . If different features get mapped to the same vector index, the corresponding values are summed up. See [6] for more details.

2.3 ProofWatch: Proof Guidance by Clause Subsumption

In this section we explain the ProofWatch guiding mechanisms. Unlike the statistical approach in ENIGMA, ProofWatch implements a form of *symbolic* memory and guidance. It produces a notion of *proof-state vector* that is dynamically created and updated.

Standard Watchlist Guidance. The watchlist (hint list) mechanism itself does not perform any statistical machine learning. It steers given clause selection via symbolic matching between generated clauses and a set of clauses called a *watchlist*. This technique has been originally developed by Veroff [42] and implemented in Otter [27] and Prover9 [28]. Since then, it has been extensively used in the AIM project [24] for obtaining long and advanced proofs of open algebraic conjectures. The watchlist mechanism is nowadays implemented also in E. All the above implementations use only a single watchlist, as opposed to ProofWatch discussed below.

Recall that a clause C *subsumes* a clause D , written $C \sqsubseteq D$, when there exists a substitution σ such that $C\sigma \subseteq D$ (where clauses are considered to be sets of literals). The watchlist guidance then works as follows. Every generated clause C is checked for subsumption with every watchlist clause $D \in W$. When C subsumes at least one of the watchlist clauses, then C is considered important for the proof search and is processed with high priority. The idea behind this is that the watchlist W contains clauses which were processed during a previous successful proof search of a related conjecture. Hence processing of similar clauses may lead to success again.

In E, the watchlist mechanism is implemented using a priority function³ which takes precedence over the weight function used to select the next given clause. Priority functions assign the priority to each clause, and clauses with higher priority are selected as given before clauses with lower priority⁴.

³ See the priority function `PreferWatchlist` in the E manual.

⁴ Numerically the lower the priority, the better. Hence 0 is the best priority.

When clauses from previous proofs are put on a watchlist, E thus prefers to follow steps from the previous proofs whenever it can.

ProofWatch. Our approach [11, Sect. 5] extends standard watchlist guidance by allowing for multiple watchlists W_1, \dots, W_n , for example, one corresponding to each related proof found before. We say that a generated clause C *matches* the watchlist W_i , written $C \sqsubseteq W_i$, iff C subsumes some clause $D \in W_i$ ($C \sqsubseteq D$). Similarly, the above watchlist clause D is said to be *matched* by C .

The reason to include multiple watchlists is that during a proof search, clauses from some watchlists might get matched more often than clauses from others. The more clauses are matched from some watchlist W_i , the more the current proof search resembles W_i , and hence W_i might be more relevant for this proof search. Thus the idea of ProofWatch is to prioritize clauses that match more relevant watchlists (proofs).

Watchlist *relevance* is dynamically computed as follows. We define $progress(W_i)$ to be the count of clauses from W_i that have been matched in the proof search thus far. The *completion ratio*, $c_i = \frac{progress(W_i)}{|W_i|}$, measures how much of the watchlist W_i has been matched. The *dynamic relevance* of each generated clause C is defined as the maximum completion ratio over all the watchlists W_i that C matches:

$$relevance(C) = \max_{W \in \{W_i: C \sqsubseteq W_i\}} \left(\frac{progress(W)}{|W|} \right)$$

The higher the dynamic relevance $relevance(C)$, the higher the priority of C . The dynamic watchlist mechanism is implemented using the E priority function.⁵ The results of experiments in [11, Sect. 6.3] on the same dataset as this work (Mizar40 [21]) indicate that dynamic relevance improves performance over an ensemble of strategies, whereas the single watchlist approach is stronger on each individual strategy.

When using a large problem library such as Mizar40, it is practically useful to choose only some proofs for watchlists. First, E's speed decreases with each additional proof on the watchlist, so if working on a large dataset, loading all available proofs as watchlists will lead to a large slowdown (cf. Sect. 4). Second, it's not guaranteed that all proofs will help E with proving the problem at hand.

3 ENIGMAWatch: ProofWatch Meets ENIGMA

3.1 Completion Ratios as Semantic Embeddings of the Proof Search

The watchlist completion ratios (c_0, \dots, c_N) (N ranges over the watchlist proofs) at each step in E's proof search can be taken as a vectorial representation of the current proof state Π . The general motivation for this approach is to come up with an *evolving* characterization of the saturation-style proof state Π , preferably in a vectorial form φ_Π suitable for machine learning tools, such as ENIGMA.

⁵ See `PreferWatchlistRelevant` in [11].

Recall that the proof state Π is a set of processed clauses P and unprocessed clauses U . The vector of watchlist completion ratios thus maintains a running tally of where clauses in $P \cup U$ match the different related proofs. In general, this could be replaced, e.g., by a vector of more abstract similarities of the current proof state to other proofs measured in various (possibly approximate) ways. In ENIGMAWatch we use the ProofWatch based *proof-state vector* for a proof state Π defined by the completion ratios, i.e., $\varphi_{\Pi} = (c_0, \dots, c_N)$. This is the first practical implementation of the general idea: using *semantic embeddings* (i.e., representations in R^n) of the proof state Π for guiding statistical learning methods. ENIGMAWatch uses the proof-state vectors φ_{Π} as follows. The positive \mathcal{C}^+ and negative \mathcal{C}^- given clauses are output along with φ_{Π} , the proof-state vector at the time of their selection, and used as added features of the proof state when training ENIGMA-style classifiers.

Table 1. Example of the proof-state vector for 8 (of 32) (serially numbered) proofs loaded to guide the proof of **YELLOW_5:36**. The three columns are the watchlist i , the completion ratio of i , and $\text{progress}(W_i)/|W_i|$.

0	0.438	42/96	1	0.727	56/77	2	0.865	45/52	3	0.360	9/25
4	0.750	51/68	5	0.259	7/27	6	0.805	62/77	7	0.302	73/242

Table 1 shows a sample proof-state vector based on 32 related proofs⁶ for the Mizar theorem **YELLOW 5:36**⁷ (De Morgan’s law⁸) at the end of the proof search. Note that some related proofs, such as #2, were almost fully matched, while others, such as #7 were mostly not matched in the proof search.

3.2 Proof Vector Construction

Data Construction. In the ProofWatch [11] experiments, the best method for selecting related proofs (watchlists) was to use k-nearest neighbor (k-NN) to recommend 32 proofs per problem. The watchlists there are thus problem specific. In ENIGMAWatch, we want the watchlists to be globally fixed across the whole library, so that the proof completion ratios have the same meaning in all proofs. To construct the proof vectors, we first use a strong E strategy to produce a set of initial proofs (14882 over the 57897 Mizar40 problems). Then we run E with ProofWatch and the same strategy over the full 57897 problems with the 14882 proofs loaded into the watchlist. The time limit for both runs was *T60-G10000*, which means that E stops after 60s or 10000 generated clauses. This data provides information on how often each watchlist was encountered in each successful proof search. The training data then

⁶ The proofs were chosen via k-NN. See [11, Sec. 6.1] for details.

⁷ http://grid01.ciirc.cvut.cz/~mptp/7.13.01.4.181.1147/html/yellow_5#T36.

⁸ $\neg(P \vee Q) \iff (\neg P) \wedge (\neg Q)$.

consists of a proof vector for each given clause (for each conjecture/problem): (*conjecture, given-clause, proof-state vector*).

Dimensionality Reduction. Next, we experiment with various pre-processing methods to reduce the *proof-state vector* dimension and thus decrease the number of watchlists loaded in E. For each problem we compute the mean of proof-state vectors over all given clauses g : $\frac{1}{\#g} \sum_g \varphi_{\Pi_g}$. This vector consists of the averaged completion ratios for each watchlist, which will be higher if the watchlist was matched earlier in the proof. This results in the mean proof-state matrix M consisting of row vectors (*mean-proof-vector*) (one for each conjecture/problem).

The following are methods experimented with in this paper for constructing the globally fixed vector of 512 watchlists from matrix M :

- *Mean*: compute the mean of M across the rows to obtain a mean proof-state vector that contains for each watchlist its average use across all problems. Then we take the top 512 watchlists.
- *Corr*: compute the Pearson correlation matrix⁹ based on (the transpose of) M , and find a relatively uncorrelated set of 512 watchlists.
- *Var*: compute the variance (across the rows) of each column in M , and take the 512 watchlists with the highest variance. The intuition is that watchlists whose completion ratio vary more over the problem corpus may be more useful for learning.
- *Rand*: randomly select 512 watchlists.

4 Multi-indices Subsumption Indexing

In order to determine whether a generated clause matches a watchlist, the generated clause must be checked for subsumption with every watchlist clause. A major limitation of previous work [11, 12] was the slowdown of E as the watchlist size increased beyond 4000 clauses. Including more than 128 proofs was impractical. This section describes a method we have developed to speed up watchlist matching.

E already implements feature vector indexing [34] used also for the purpose of watchlist matching. The watchlist clauses are inserted into an indexing data structure and various properties of clauses are used to prune possible subsumption candidates. In this way, the number of possibly expensive subsumption calls is reduced. We build upon this, and further limit the number of required subsumption checks by using multiple indices instead of a single index.¹⁰

We take advantage of the fact that a clause C cannot subsume a clause D if the top-level predicate symbols do not match. In particular, $C \sqsubseteq D$ can only hold if all the predicate symbols from C also appear in D , because substitution can neither introduce nor remove predicate symbols from a clause.

⁹ <https://docs.scipy.org/doc/numpy/reference/generated/numpy.corrcoef.html>.

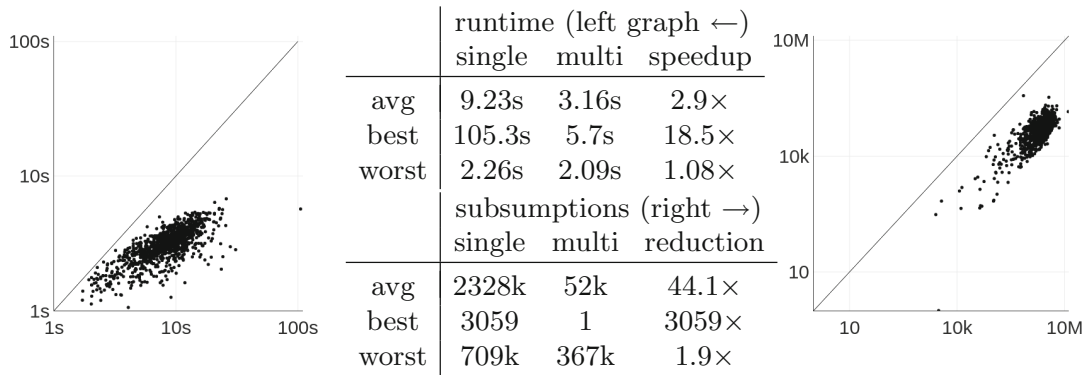
¹⁰ Even with multiple watchlists, all the watchlist clauses are inserted into a single index, and only the name of the original watchlist is additionally stored.

We define the *code* of a clause C , denoted $\text{code}(C)$, as the set of predicate symbols with their logical signs (either $+$ for positive predicates, or $-$ for negated ones). For example, the code of the clause “ $P(a) \vee \neg P(b) \vee P(f(x))$ ” is the set $\{+P, -P\}$. The following holds because codes are preserved under substitution.

Lemma 1. *Given clauses C and D , $C \sqsubseteq D$ implies $\text{code}(C) \subseteq \text{code}(D)$.*

We create a separate index for every different clause code. Each watchlist clause D is inserted only to the index corresponding to $\text{code}(D)$. In order to check whether some clause C matches a watchlist, we only need to search in the indices whose codes are supersets of (or equal to) $\text{code}(C)$. Each index is implemented using E’s native feature vector indexing structure. Evaluation of this simple indexing method is provided in Sect. 5.1.

Table 2. Evaluation of multi-indices subsumption indexing.



5 Experiments

This section describes the experimental evaluation¹¹ of

1. the improved watchlist mechanism from Sect. 4
2. the watchlist selection for ENIGMAWatch from Sect. 3

5.1 Multi-indices Subsumption Indexing Evaluation

We propose a simple experiment to evaluate our implementation of multi-indices subsumption indexing from Sect. 4. We take a random sample of 1000 problems from the Mizar40 [21] data set and create a watchlist with around 60 k clauses coming from proofs of problems similar to the sample problems. We then run E

¹¹ Experiments code and data are available at <https://github.com/ai4reason/eprover-data/tree/master/TABLEAUX-19>

All experiments are run on the same hardware: Intel(R) Xeon(R) Gold 6140 CPU @ 2.30 GHz with 188 GB RAM.

on the sample problems with a fixed limit of 1000 generated clauses. This gives us a measure of how fast the single-index and multi-indices versions are, that is, how fast they can generate the first 1000 clauses. As the watchlist indexing does not influence the proof search, both versions process the same clauses and output the same result. Each generated clause has to be checked for watchlist subsumption and hence the limit on generated clauses is also the limit on different watchlist checks. We expect the number of clause-to-clause subsumption checks to decrease with multi-indices, as the method prunes possible subsumption candidates.

The results of the experiments are presented in Table 2. For each problem, we measure the runtime (left graph) and the number of different clause subsumption calls (right graph). The suffix “s” stands for seconds, “k” stands for thousands, and “M” stands for millions. Although subsumption is also used for purposes other than watchlist matching, we should be able to observe a decrease in the number of calls. Each point in the graphs corresponds to one sample problem, and is drawn at the position (x, y) corresponding to the results of single-index (x) and multi-indices (y) versions. Hence points below the diagonal signify an improvement. Also note logarithmic axes. The table shows the average improvement, and also the best and the worst cases. From the results, we can see that an average speed-up is almost 3 times. Furthermore, the average reduction of subsumption calls is more than 44 times and the number is reduced even in the worst case.

Table 3. ProofWatch evaluation: Problems solved by different versions.

Baseline	Mean	Var	Corr	Rand	Baseline \cup Mean	Total
1140	1357	1345	1337	1352	1416	1483

Table 4. ENIGMAWatch evaluation: Problems solved and the effect of looping.

loop	ENIGMA	Mean	Var	Corr	Rand	ENIGMA \cup Mean	Total
0	1557	1694	1674	1665	1690	1830	1974
1	1776	1815	1812	1812	1847	1983	2131
2	1871	1902	1912	1882	1915	2058	2200
3	1931	1954	1946	1920	1926	2110	2227

The number of watchlist clauses in the experiments was 61501, and the multi-indices version used 11442 different indices. This means that there were less than 6 clauses per index in average, although the count of clauses in different indices varied from 1 to 3837. The most crowded index was for the code $\{+ =\}$, that is, for positive equality clauses. Finally, 6955 indices contained only a single clause.

5.2 Experimental Evaluation of ENIGMAWatch

The experiments are done on a random subset of 5000 Mizar40 [21] problems. The time limit of 60 s and 30000 generated clauses is used to allow a comparison to be done without regard for the differences in clause processing speed. The 30000 is approximately the average number of clauses that the baseline strategy generates in 10s. Table 3 provides the evaluation of different watchlist selection mechanisms using ProofWatch (without ENIGMA) and making use of the improved watchlist indexing. The last two columns show the number of problems solved by (1) the Baseline together with Mean, and by (2) all the five methods. This shows the relative complementarity of the methods. We can see that the Mean method yields the best results, reaching more than 15% improvement over the baseline strategy. The Rand method is however quite competitive.

Table 4 provides the evaluation of ENIGMAWatch and its comparison to ENIGMA. The experiments are done in multiple loops, where in each loop all the proof-runs in prior loops can be used as training data. This way ENIGMA can learn increasingly effective models.

We can see that ENIGMAWatch can attain superior performance to ENIGMA. The relation of looping and results is interesting. The largest absolute improvement over ENIGMA is in loop 0 – 8.8% by the Mean method. This however drops to 1.2% in loop 4. In loops 1 and 2, Rand is the strongest, but Mean ends up being the best in loop 3. In total, all the ENIGMA and ENIGMAWatch methods solve together nearly twice as many problems as the baseline strategy. Figure 1 shows the results of running ENIGMA and Mean for 13 loops. The rate of improvement slows down, both methods eventually converge to a similar level of performance, and the union of the two is ca. 150 problems better.

Table 5. ENIGMA and ENIGMAWatch: Model and training statistics.

Model	Pos. acc	Neg. acc	Features	Watchlist F	Train size	Train time
ENIGMA0	99.12%	92.16%	5061	0	0.4 GB	14 min
ENIGMA1	97.39%	86.82%	7071	0	0.8 GB	31 min
ENIGMA2	96.13%	83.92%	8089	0	1.4 GB	55 min
ENIGMA3	95.39%	82.5%	8662	0	2.0 GB	85 min
Mean0	99.05%	92.59%	5424	308	2.9 GB	19 min
Mean1	96.92%	88.16%	6950	316	6.2 GB	29 min
Mean2	95.75%	86.46%	7809	331	9.6 GB	38 min
Mean3	95.04%	85.24%	8313	330	13.0 GB	39 min

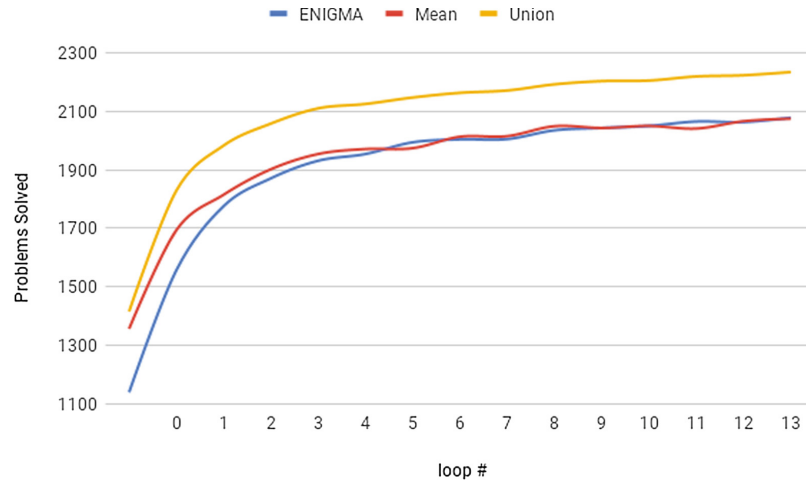


Fig. 1. Convergence: The improvement of ENIGMA and Mean decreases over 13 loops, and their performance converges. The Union is consistently ca. 150 problems better.

5.3 Training, Model Statistics and Analysis

The XGBoost models used in our experiments are trained with a maximum tree depth of 9 and 200 rounds (which means 200 trees are learned). There are 300000 features in the 5000 problem dataset hashed into 2^{15} buckets. Combining clause and conjecture features with the watchlist completion ratios, XGBoost makes its predictions based on 66048 features ($2 \cdot 2^{15}$ plus the count of completion ratios).

Table 5 provides various training and model statistics of the ENIGMA and ENIGMAWatch models and their loops. The columns “Pos. Acc.” and “Neg. Acc.” describe the training accuracy of the models on positive and negative training examples. The column “Features” presents the number of features referenced in the decision trees. We see that the models use a small fraction of all the 66048 available features. The column “Watchlist F.” provides the number of watchlist features out of all the used features. Finally, “Train Size” and “Train Time” specify the size of the input training file (in GB) and training times (in minutes). The XGBoost models after the training are smaller than 4 MB.

We can see that the accuracy decreases with the increase of the training data size, but the number of theorems proved increases. About 62% of the watchlists are judged as useful by XGBoost and used in the decision trees. Figure 2 shows the root of the first decision tree of the Mean model in loop 3. Green means “yes” (the condition holds), red means “no”, and blue means that the feature is not present. The multi line box is a (shortened) bucket of features, and single line boxes correspond to watchlists (#194, etc.). We can see that ENIGMAWatch uses a watchlist feature for the very first decision when judging newly generated clauses. This shows that the features that characterize the evolving proof state are indeed considered very significant by the methods that automatically learn given clause guidance.

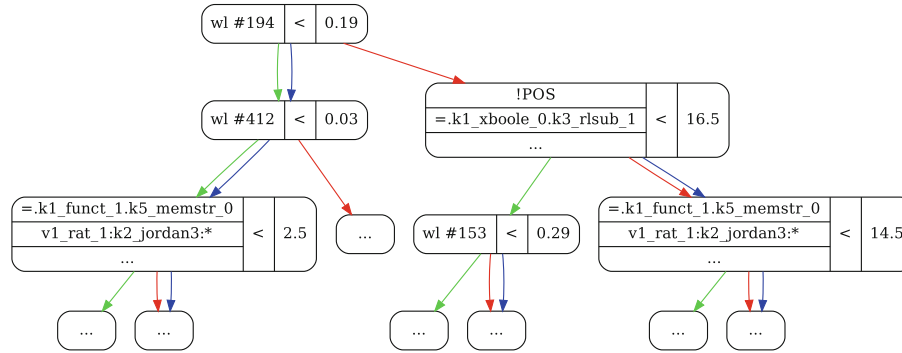


Fig. 2. Example of an XGBoost decision tree.

6 Conclusion and Future Work

We have produced and evaluated the first practically usable version of the ENIGMAWatch system which can now be efficiently used over large mathematical datasets. The previous experiments with the first prototype on the small MPTP Challenge [12] demonstrated that ENIGMAWatch can find proofs faster (in terms of how many processed clauses are needed). The work presented here shows that with improved subsumption indexing, feature hashing, and suitable global watchlist selection, ENIGMAWatch outperforms ENIGMA on the large Mizar40 dataset. In particular, ENIGMAWatch significantly outperforms both ProofWatch and ENIGMA when used without looping. With several MaLAREa-style [37, 40] iterations of proving and learning, the difference to ENIGMA gets smaller, however the two methods are still quite complementary, providing solutions to a large number of different problems. In total, all the ENIGMA and ENIGMAWatch methods (Table 4) together solve almost twice as many problems as the baseline strategy after four iterations of learning and proving.

The system is ready to be used on hard problems and to expand the set of Mizar problems for which an ATP proof has been found. Future work includes refining the watchlist selection, defining more sophisticated methods of computing the proof completion ratios, analyzing the learned decision tree models to see which watchlists are the most useful, and also defining further and more abstract meaningful representations and embeddings of saturation-style proof search.

References

1. Alama, J., Heskes, T., Kühlwein, D., Tsvitshivadze, E., Urban, J.: Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning* **52**(2), 191–213 (2014)
2. Alemi, A.A., Chollet, F., Eén, N., Irving, G., Szegedy, C., Urban, J.: DeepMath - deep sequence models for premise selection. In: Lee, D.D., Sugiyama, M., Luxburg, U.V., Guyon, I., Garnett, R. (eds.) *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016*, 5–10 December 2016, Barcelona, Spain, pp. 2235–2243 (2016)

3. Blanchette, J.C., Greenaway, D., Kaliszyk, C., Kühlwein, D., Urban, J.: A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning* **57**(3), 219–244 (2016)
4. Bridge, J.P., Holden, S.B., Paulson, L.C.: Machine learning for first-order theorem proving - learning to select a good heuristic. *J. Autom. Reasoning* **53**(2), 141–172 (2014)
5. Chen, T., Guestrin, C.: Xgboost: a scalable tree boosting system. In: *KDD*, pp. 785–794. ACM (2016)
6. Chvalovský, K., Jakubův, J., Suda, M., Urban, J.: ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. CoRR, abs/1903.03182 (2019)
7. Denzinger, J., Fuchs, M., Goller, C., Schulz, S.: Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München (1999)
8. Ertel, W., Schumann, J., Suttner, C.B.: Learning heuristics for a theorem prover using back propagation. In: Retti, J., Leidlmaier, K. (eds.) *Österreichische Artificial Intelligence-Tagung, Igls, Tirol*, vol. 208, pp. 87–95. Springer, Heidelberg (1989). https://doi.org/10.1007/978-3-642-74688-8_10
9. Färber, M., Brown, C.: Internal Guidance for Satallax. In: Olivetti, N., Tiwari, A. (eds.) *IJCAR 2016. LNCS (LNAI)*, vol. 9706, pp. 349–361. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_24
10. Gauthier, T., Kaliszyk, C.: Premise selection and external provers for HOL4. In: *Certified Programs and Proofs (CPP 2015). LNCS*. Springer, Berlin (2015). <https://doi.org/10.1145/2676724.2693173>
11. Goertzel, Z., Jakubův, J., Schulz, S., Urban, J.: ProofWatch: watchlist guidance for large theories in E. In: Avigad, J., Mahboubi, A. (eds.) *ITP 2018. LNCS*, vol. 10895, pp. 270–288. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94821-8_16
12. Goertzel, Z., Jakubův, J., Urban, J.: ProofWatch meets ENIGMA: first experiments. In: Barthe, G., Korovin, K., Schulz, S., Suda, M., Sutcliffe, G., Veanes, M. (eds.) *LPAR-22 Workshop and Short Paper Proceedings*, vol. 9, pp. 15–22, Kalpa Publications in Computing. EasyChair (2018)
13. Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.): *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16–19, 2015*, vol. 36, EPiC Series in Computing. EasyChair (2015)
14. Grabowski, A., Kornilowicz, A., Naumowicz, A.: Mizar in a nutshell. *J. Formalized Reasoning* **3**(2), 153–245 (2010)
15. Jakubův, J., Urban, J.: Hierarchical invention of theorem proving strategies. *AI Commun.* **31**(3), 237–250 (2018)
16. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) *CICM 2017. LNCS (LNAI)*, vol. 10383, pp. 292–302. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_20
17. Jakubův, J., Urban, J.: Enhancing ENIGMA given clause guidance. In: Rabe, F., Farmer, W.M., Passmore, G.O., Youssef, A. (eds.) *CICM 2018. LNCS (LNAI)*, vol. 11006, pp. 118–124. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96812-4_11
18. Jakubův, J., Urban, J.: Hammering Mizar by learning clause guidance. CoRR, abs/1904.01677 (2019)
19. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning* **53**(2), 173–213 (2014)

20. Kaliszyk, C., Urban, J.: FEMaLeCoP: fairly efficient machine learning connection prover. In: Davis, M., Fehner, A., McIver, A., Voronkov, A. (eds.) LPAR 2015. LNCS, vol. 9450, pp. 88–96. Springer, Heidelberg (2015). https://doi.org/10.1007/978-3-662-48899-7_7
21. Kaliszyk, C., Urban, J.: MizAR 40 for Mizar 40. *J. Autom. Reasoning* **55**(3), 245–256 (2015)
22. Kaliszyk, C., Urban, J., Michalewski, H., Olsák, M.: Reinforcement learning of theorem proving. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, Montréal, Canada, 3–8 December 2018, pp. 8836–8847 (2018)
23. Kaliszyk, C., Urban, J., Vyskocil, J.: Efficient semantic features for automated reasoning over large theories. In: IJCAI, pp. 3084–3090. AAAI Press (2015)
24. Kinyon, M., Veroff, R., Vojtěchovský, P.: Loops with abelian inner mapping groups: an application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS (LNAI), vol. 7788, pp. 151–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_8
25. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
26. Loos, S.M., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. In: Eiter, T., Sands, D. (eds.) LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, 7–12 May 2017. EPiC Series in Computing, vol. 46, pp. 85–105. EasyChair (2017)
27. McCune, W., Wos, L.: Otter: the CADE-13 competition incarnations. *J. Autom. Reasoning* **18**(2), 211–220 (1997). Special Issue on the CADE 13 ATP System Competition
28. McCune, W.W.: Prover9 and Mace4. <http://www.cs.unm.edu/~mccune/prover9/>, 2005–2010. Accessed 29 Mar 2016
29. Piotrowski, B., Urban, J.: ATPBOOST: learning premise selection in binary setting with ATP feedback. In: Galmiche, D., Schulz, S., Sebastiani, R. (eds.) IJCAR 2018. LNCS (LNAI), vol. 10900, pp. 566–574. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94205-6_37
30. Polikar, R.: Ensemble based systems in decision making. *IEEE Circuits Syst. Mag.* **6**(3), 21–45 (2006)
31. Schäfer, S., Schulz, S.: Breeding theorem proving heuristics with genetic algorithms. In: Gottlob et al. [13], pp. 263–274 (2015)
32. Schulz, S.: Learning search control knowledge for equational deduction. In: DISKI, vol. 230, Infix Akademische Verlagsgesellschaft (2000)
33. Schulz, S.: E - a brainiac theorem prover. *AI Commun.* **15**(2–3), 111–126 (2002)
34. Schulz, S.: Simple and efficient clause subsumption with feature vector indexing. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS (LNAI), vol. 7788, pp. 45–67. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_3
35. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) LPAR 2013. LNCS, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_49
36. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning* **37**(1–2), 21–43 (2006)
37. Urban, J.: MaLAREa: a metasystem for automated reasoning in large theories. In: Sutcliffe, G., Urban, J., Schulz, S. (eds.) ESARLT, vol. 257, CEUR Workshop Proceedings. CEUR-WS.org (2007)

38. Urban, J.: BliStr: the blind strategymaker. In: Gottlob et al. [13], pp. 312–319 (2013)
39. Urban, J., Sutcliffe, G.: ATP cross-verification of the mizar MPTP challenge problems. In: Dershowitz, N., Voronkov, A. (eds.) LPAR 2007. LNCS (LNAI), vol. 4790, pp. 546–560. Springer, Heidelberg (2007). https://doi.org/10.1007/978-3-540-75560-9_39
40. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLAREa SG1 - machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 441–456. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_37
41. Urban, J., Vyskočil, J., Štěpánek, P.: MaLeCoP machine learning connection prover. In: Brünnler, K., Metcalfe, G. (eds.) TABLEAUX 2011. LNCS (LNAI), vol. 6793, pp. 263–277. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22119-4_21
42. Veroff, R.: Using hints to increase the effectiveness of an automated reasoning program: case studies. *J. Autom. Reasoning* **16**(3), 223–239 (1996)



Make E Smart Again (Short Paper)

Zarathustra Amadeus Goertzel^{([1](#))}

Czech Technical University in Prague, Prague, Czechia
zariuq@gmail.com

Abstract. In this work in progress, we demonstrate a new use-case for the ENIGMA system. The ENIGMA system using the XGBoost implementation of gradient boosted decision trees has demonstrated high capability to learn to guide the E theorem prover’s inferences in real-time. Here, we strip E to the bare bones: we replace the KBO term ordering with an identity relation as the minimal possible ordering, disable literal selection, and replace evolved strategies with a simple combination of the clause weight and FIFO (first in first out) clause evaluation functions. We experimentally demonstrate that ENIGMA can learn to guide E as well as the smart, evolved strategies even without these standard automated theorem prover functionalities. To this end, we experiment with XGBoost’s meta-parameters over a dozen loops.

1 Introduction: Making E Stupid and Then Smart Again

State-of-the-art saturation-based automated theorem provers (ATPs) for first-order logic (FOL), such as E and Vampire [12], employ the *given clause algorithm* [13], translating the input FOL problem $T \cup \{\neg C\}$ (background theory and negated conjecture) into a refutationally equivalent set of clauses. The search for a contradiction is performed maintaining sets of *processed* (P) and *unprocessed* (U) clauses (the *proof state* Π). The algorithm repeatedly selects a *given clause* g from U , moves g to P , and extends U with all clauses inferred with g and P . This process continues until a contradiction is found, U becomes empty, or a resource limit is reached.

Historically, *term ordering*, together with *literal selection*, is used to guarantee the completeness of the proof search [1] and to “tame the growth of the search space and help steer proof search” [5]. Term ordering ensures that rewriting happens in only one direction, toward smaller terms. Literal selection limits the inferences done with each given clause g to the selected literals, which slows down the growth of the search space and reduces redundant inferences.

E includes a *strategy* language of *clause evaluation functions*, made up of weight and priority functions, to heuristically guide the proof search. In this

Supported by the ERC Consolidator grant no. 649043 AI4REASON and by the Czech project AI&Reasoning CZ.02.1.01/0.0/0.0/15_003/ 0000466 and the European Regional Development Fund.

work, I use two algorithmically invented [6,7] strategies, E1 and E2¹, that use many sophisticated clause evaluation functions, the Knuth-Bendix ordering (KBO6), literal selection, and other E heuristics.

The ENIGMA [4,8–10] system with the XGBoost [2] implementation of gradient boosted decision trees has recently demonstrated high capability to learn to guide the E [14] theorem prover’s inferences in real-time. ENIGMA uses the XGBoost model as a clause evaluation function to recommend clauses for selection based on clause and conjecture features. In particular, after several proving and learning iterations, its performance on the 57880 problems from the Mizar40 [11] benchmark improved by 70% (= 25397/14933) [10] over the strategy E1 used for the initial proving phase.

In this work, E is stripped to the bare bones by disabling term ordering and literal selection. KBO6 is replaced with an identity relation as the minimal possible ordering (called IDEN – an addition to E²). While this frees E to do inferences in any order, E can no longer perform rewriting inferences. The strategy E1 is replaced with the simple combination of the clause weight and FIFO (first in first out) evaluation functions. E is thus practically reduced to a basic superposition prover, without advanced heuristics, rewriting, or completeness guarantees. We call this strategy E0:

```
--definitional-cnf=24 --prefer-initial-clauses -tIDEN
--restrict-literal-comparisons -WNoSelection
-H'(5*Clauseweight(ConstPrio,1,1,1),1*FIFOweight(ConstPrio))'
```

E0 solves only 3872 of the Mizar40 problems in 5 s compared to 14526 for E1. The first research question is the extent to which ENIGMA with this basic prover can learn ATP guidance completely on its own. The second is to what extent ENIGMA’s learning can be boosted with data from strong strategies and models. That is, I explore how smart machine learning can become in this *zero-strategy* setting. The more general related question is to what extent can machine learning replace the sophisticated human-invented theorem-proving body of wisdom used in today’s ATPs for restricting advanced proof calculi.

2 Experiments

We evaluate ENIGMA with the basic strategy, E0, in several scenarios and over two datasets of different sizes. All experiments are run with 5 s per problem^{3 4}.

¹ Strategies E1 and E2 are displayed in the appendix.

² The E version used in this paper can be found at <https://github.com/zariuq/eprover/tree/identity-order>, and the library for running ENIGMA with E can be found at <https://github.com/zariuq/enigmatic>.

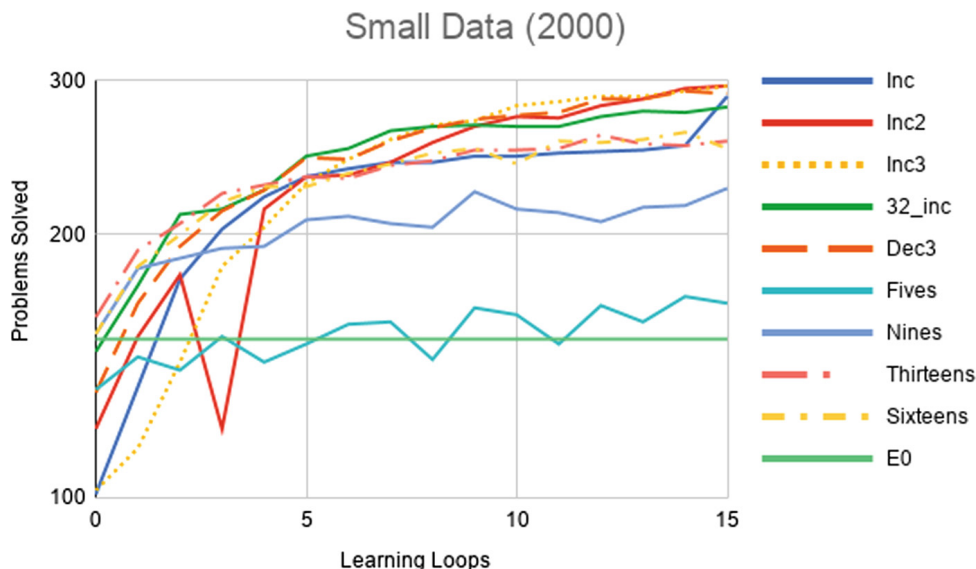
³ As a rule of thumb, E solves most problems within a few seconds or *not for a very long time*.

⁴ All the experiments are run on the same hardware unless otherwise specified: Intel(R) Xeon(R) Gold 6140 CPU @ 2.30GHz with 188GB RAM.

ENIGMA has so far been used in two ways: *coop* combines the learned model with some standard E strategy equally (50:50) while *solo* only uses the learned model for choosing the given clauses. The best results have been achieved by MaLAREa-style [16] looping: that is, an ENIGMA model is trained and run with E (loop 0), then the resulting data are added to the initial training data and a new ENIGMA model is trained (loop 1).

In this work, ENIGMA trains with both *solo* and *coop* data. I present results from *solo* runs because they represent the most minimal setting.

2.1 Small Data (2000 Problems)



The E evaluations and XGBoost training can take a long time on the full Mizar40 dataset, so 2000 randomly sampled problems are used to test meta-parameters on. Each XGBoost model consists of T decision trees of depth D , the most important training meta-parameters in addition to the learning rate ($\eta = 0.2$). In previous work with ENIGMA, T and D were fixed for all loops of learning. Here we try to vary the values of T and D during 16 loops. Let $S_{D,T}$ denote the experiment with specific T and D . Of the many protocols tested, the following are included in the plot of solved problems (above): *Fives* ($S_{5,100}$), *Nines* ($S_{9,100}$), *Thirteens* ($S_{13,200}$), *Sixteens* ($S_{16,100}$).

We also experiment with adaptively setting the meta-parameters as the number of training examples increases according to the following protocols:

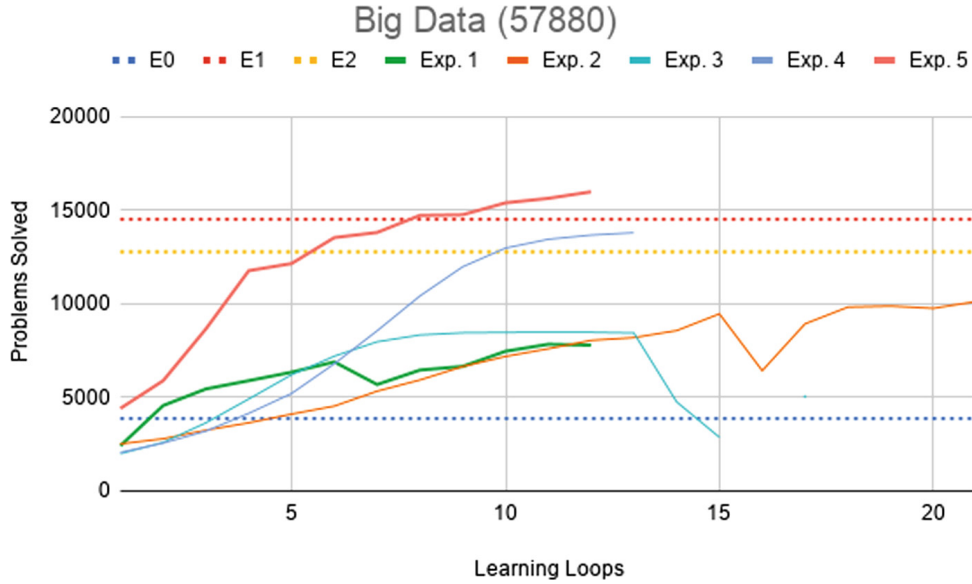
- *Inc* ($S_{[3,33],100}$) increases D by 2 from 3 to 33 and keeps $T = 100$ fixed.
- *32_inc* ($S_{32,[50,250]}$) fixes $D = 32$ and gradually increases T from 50 to 250.
- *Inc2* ($S_{[3,33],*}$) gradually decreases T from 150 to 50, varying the value intuitively⁵.

⁵ Precise details of intuitively set parameters can be seen in the appendix.

- $Inc3(S_{[3,33],[50,250]})$ aims to be more systematic and steps T from 50 to 250.
- $Dec3(S_{[3,33],[250,50]})$ decreases T from 250 to 50.

At the 16th loop Inc 's performance is best, solving 299 problems, doubling the performance of E0 (152). However $Inc2$ and $Inc3$ solve 298 problems and 32_inc solves 291 problems. The conclusion is that simple protocols work well so long as T or D is incremented adaptively rather than fixed.

2.2 Big Data (57880 Problems)



These experiments are done on the large benchmark of 57880 Mizar40 [11] problems from the MPTP dataset [15]. E1 and E2 are two strong E strategies that solve 14526 and 12788 problems.

- **Experiment 1** is done with $D = 9$ and $T = 200$ and uses our previously trained model that allowed us to solve 25562 problems when cooperating with E1 in our previous experiments [10]. This strong model, which hashes the features into 32768 (2^{15}) buckets [3, Sect. 3.4], is used with E0 now.
- **Experiment 2**'s parameters were intuitively toggled during the looping as in $Inc3$, and a feature size of 2^{16} is used. Exp. 2 uses training data from E1 and E2 for additional guidance up to the 4th loop (and then stops including them in the training data based on the assumption they may confuse learning).
- **Experiment 3** sets T and D according to protocol $Inc3$. Exp. 3 only learns from E run with E0 and trains on the GPU, which requires the feature size to be reduced to 256.
- **Experiment 4** mimics Exp. 3 but uses E1 and E2 data for training (up to the 4th loop).
- **Experiment 5** further tests boosting with data from an E0 ENIGMA model that proved 9759 problems and an E1 that proved 21542 problems.

Tree depth is intuitively varied among 32, 512, and 1000, the number of trees is varied among 2, 100, 200, and 32. The feature vector size starts at 2^{14} and is decreased to allow the data to fit on the RAM, down to 32 ($= 2^5$).

As seen in the figure, the strong model does not help much in guiding E without ordering or selection in Exp. 1. Exp. 2 learns gradually and catches up with Exp. 1, but seems to plateau around 10,000. Surprisingly the pure Exp. 3 learns fast with the small feature size, but plateaus and drops in performance (perhaps due to overfitting). Exp. 4 indicates that guidance is useful and surpasses E2 with 13805 in round 13. Exp. 5 solves 15990 problems, showing that ENIGMA can take E0 beyond the smart strategies with appropriate parameters and boosting. This is a great improvement over the 3872 problems solved by E0.

3 Conclusion

ENIGMA can learn to guide the E prover effectively even without smart strategies and term orderings. The models confer a 256% increase over the naive E0 after 13 rounds of the proving/learning loop, and even trained without guidance data, a 121% increase.

The experiments indicate that machine learning can be used to fully control an ATP's guidance, learning to replace orderings, heuristic strategies, and deal with the increase in generated clauses without literal selection. However the combination of ENIGMA and standard ATP heuristics still significantly outperforms ENIGMA alone.

Given the large symmetry-breaking impact of these methods in classical ATP, future work includes, e.g., training the guidance in such a way that redundant (symmetric) inferences are not done by the trained model once it has committed to a certain path. This probably means equipping the learning with more history and knowledge of the proof state in the saturation-style setting. ENIGMAWatch [4] may aid with symmetry breaking by focusing the proof search on particular proof paths. Additional work is needed to isolate the factors in Exp. 5's performance, and determine the most effective boosting methods in addition to increasing D and T with training loops. Ablation studies should be done to discover the impact of term ordering and literal selection individually on E and ENIGMA's performance. Perhaps term ordering alone is sufficient to train good ENIGMA models.

Running ENIGMA without term ordering and other restrictions is important because it may allow us to combine training data from different strategies, and it may allow ENIGMA to find novel proofs.

Acknowledgments. The research topic was proposed by Jan Jakubuv and Josef Urban, and further discussed with them, Martin Suda, and Thomas Tan. I also thank the AITP'20 anonymous referees for their comments on the first extended abstract of this work.

A Strategies

Strategy E1 is:

```
--definitional-cnf=24 --split-aggressive --simul-paramod
--forward-context-sr --destructive-er-aggressive --destructive-er
--prefer-initial-clauses -tKBO -winvfrequank -c1 -Ginvfreq -F1
--delete-bad-limit=150000000 -WSelectMaxLComplexAvoidPosPred
-H' (1*ConjectureTermPrefixWeight(DeferSOS,1,3,0.1,5,0,0.1,1,4),
1*ConjectureTermPrefixWeight(DeferSOS,1,3,0.5,100,0,0.2,0.2,4),
1*Refinedweight(PreferWatchlist,4,300,4,4,0.7),
1*RelevanceLevelWeight2(PreferProcessed,0,1,2,1,1,1,200,200,2.5,9999.9,9999.9),
1*StaggeredWeight(DeferSOS,1),
1*SymbolTypeweight(DeferSOS,18,7,-2,5,9999.9,2,1.5),
2*Clauseweight(PreferWatchlist,20,9999,4),
2*ConjectureSymbolWeight(DeferSOS,9999,20,50,-1,50,3,3,0.5),
2*StaggeredWeight(DeferSOS,2))'
```

Strategy E2 is:

```
--definitional-cnf=24 --split-aggressive --split-reuse-defs
--simul-paramod --forward-context-sr --destructive-er-aggressive
--destructive-er --prefer-initial-clauses -tKBO -winvfrequank
-c1 -Ginvfreq -F1 --delete-bad-limit=150000000
-WSelectMaxLComplexAvoidPosPred -H' (
3*ConjectureRelativeSymbolWeight(PreferUnitGroundGoals,0.1,100,100,50,100,0.3,1.5,1.5),
4*FIFOWeight(PreferNonGoals),
5*RelevanceLevelWeight2(ConstPrio,1,0,2,1,50,-2,-2,100,0.2,3,4))'
```

B Additional Protocol Details

In this section I include the details for *intuitively toggled* protocols.

Protocol *Inc2* is as follows:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Depth	3	5	7	9	11	13	15	17	19	21	23	25	27	29	31	33
Trees	150	150	150	100	100	100	75	50	75	100	150	75	100	150	75	100

The protocol for Exp. 2 is as follows:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
D	4	5	6	7	8	9	10	11	12	13	14	15	16	16	32	9	16	32	64	24	25	32
T	50	150	160	170	180	190	200	200	200	200	210	220	225	225	225	300	300	225	150	250	250	250

The protocol for Exp. 5 requires some explanation. The motivation is to see how far E0 can be taken, even if the methods are too CPU-intensive for a thorough grid search.

Exp. 2 and Exp. 4 demonstrate the utility of boosting. Thus to create better boosting data I trained ENIGMA for 10 loops with strategies E1 through E12 and used this as boosting data for the first 4 of 10 loops of training. In addition to training E0, and in the spirit of ablation studies, I also trained ENIGMA models for E0 with KBO ordering (and no literal selection) and for E0 with KBO ordering and restricted literal comparisons. The motivation is that these versions may serve as a bridge between standard E and the basic E0.

Then I used these results to boost an ENIGMA model in loop 0, and trained based on this for 10 loops, proving 9759 problems.

Finally this data and the data from a loop 3 ENIGMA model trained with E1 is used to boost E0 with the following meta-parameters:

	0	1	2	3	4	5	6	7	8	9	10	11
Depth	512	512	32	1000	32	1000	32	1000	32	1000	1000	100
Trees	2	2	100	100	200	100	200	32	300	32	32	32
Feature size	16384	8192	4096	28	4096	28	4096	32	2048	64	32	128


References

1. Bachmair, L., Ganzinger, H.: Rewrite-based equational theorem proving with selection and simplification. *J. Log. Comput.* **3**(4), 217–247 (1994)
2. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD 2016)*, pp. 785–794. ACM, New York (2016)
3. Chvalovský, K., Jakubův, J., Suda, M., Urban, J.: ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In: Fontaine, P. (ed.) *CADE 2019*. LNCS (LNAI), vol. 11716, pp. 197–215. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_12
4. Goertzel, Z., Jakubův, J., Urban, J.: ENIGMAWatch: proofWatch meets ENIGMA. In: Cerrito, S., Popescu, A. (eds.) *TABLEAUX 2019*. LNCS (LNAI), vol. 11714, pp. 374–388. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29026-9_21
5. Hoder, K., Reger, G., Suda, M., Voronkov, A.: Selecting the selection. In: Olivetti, N., Tiwari, A. (eds.) *IJCAR 2016*. LNCS (LNAI), vol. 9706, pp. 313–329. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-40229-1_22
6. Jakubův, J., Urban, J.: Hierarchical invention of theorem proving strategies. *AI Commun.* **31**(3), 237–250 (2018)
7. Jakubův, J., Urban, J.: Extending E prover with similarity based clause selection strategies. In: Kohlhase, M., Johansson, M., Miller, B., de Moura, L., Tompa, F. (eds.) *CICM 2016*. LNCS (LNAI), vol. 9791, pp. 151–156. Springer, Cham (2016). https://doi.org/10.1007/978-3-319-42547-4_11
8. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) *CICM 2017*. LNCS (LNAI), vol. 10383, pp. 292–302. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_20

9. Jakubův, J., Urban, J.: Enhancing ENIGMA given clause guidance. In: Rabe, F., Farmer, W.M., Passmore, G.O., Youssef, A. (eds.) CICM 2018. LNCS (LNAI), vol. 11006, pp. 118–124. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96812-4_11
10. Jakubův, J., Urban, J.: Hammering Mizar by learning clause guidance. In: Harrison, J., O’Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving, (ITP 2019) of LIPIcs, 9–12 September 2019, Portland, OR, USA, vol. 141, pp. 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
11. Kaliszzyk, C., Urban, J.: MizAR 40 for Mizar 40. *J. Autom. Reasoning* **55**(3), 245–256 (2015). <https://doi.org/10.1007/s10817-015-9330-8>
12. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
13. Overbeek, R.A.: A new class of automated theorem-proving algorithms. *J. ACM* **21**(2), 191–200 (1974)
14. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 495–507. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_29
15. Urban, J.: MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning* **37**(1–2), 21–43 (2006)
16. Urban, J., Sutcliffe, G., Pudlák, P., Vyskočil, J.: MaLAREa SG1 - machine learner for automated reasoning with semantic guidance. In: Armando, A., Baumgartner, P., Dowek, G. (eds.) IJCAR 2008. LNCS (LNAI), vol. 5195, pp. 441–456. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-71070-7_37



Fast and Slow Enigmas and Parental Guidance

Zarathustra A. Goertzel¹() , Karel Chvalovský¹, Jan Jakubův^{1,2},
Miroslav Olšák², and Josef Urban¹

¹ Czech Technical University in Prague, Prague, Czech Republic

² University of Innsbruck, Innsbruck, Austria

Abstract. We describe several additions to the ENIGMA system that guides clause selection in the E automated theorem prover. First, we significantly speed up its neural guidance by adding server-based GPU evaluation. The second addition is motivated by fast weight-based rejection filters that are currently used in systems like E and Prover9. Such systems can be made more intelligent by instead training fast versions of ENIGMA that implement more intelligent pre-filtering. This results in combinations of trainable fast and slow thinking that improves over both the fast-only and slow-only methods. The third addition is based on “judging the children by their parents”, i.e., possibly rejecting an inference before it produces a clause. This is motivated by standard evolutionary mechanisms, where there is always a cost to producing all possible offsprings in the current population. This saves time by not evaluating all clauses by more expensive methods and provides a complementary view of the generated clauses. The methods are evaluated on a large benchmark coming from the Mizar Mathematical Library, showing good improvements over the state of the art.

1 Introduction: The Fast and The Smart

Throughout the history of automated theorem proving, there have been two very different approaches to strengthening automated theorem provers (ATPs). The first one (*the fast*) relies on better engineering, such as improving the indexing for inference and reduction rules and on optimized low-level implementations. The gains achieved in this way can be quite high [9, 15, 22, 28, 31, 38].

The second approach (*the smart*) relies on advanced strategies and heuristics for guiding the proof search. This includes methods using extensive previous knowledge, e.g., various kinds of *symbolic* machine learning, such as the *hints* method in Otter [37] and Prover9 [19], and its *watchlist* [26] and *proofwatch* [6] variants implemented in E [29, 30]. With the recent advent of *statistical* machine learning (ML), a number of knowledge-based ATP-guiding methods have been created [3, 10, 11, 17]. This is done by compiling (extracting, compressing, generalizing) the previous knowledge into statistical ML *predictors* (models) that are then used to predict the usefulness of inference steps in the proof search.

The *smart* approaches, while potentially sophisticated and AI-motivated, may incur prohibitively high costs in their prediction modules, in particular when naively implemented [21, 36]. This can make them inferior in practice to faster alternative approaches, such as various kinds of randomization [25] and building of portfolios of complementary fast strategies [13, 27, 35]. This issue is getting increasingly important as deep learning (DL) is used for ATP guidance, sometimes with large cloud-based DL-predictors running on specialized hardware that hides the amount of resources used. It also complicates rigorous comparisons in established ATP competitions such as CASC/LTB [32, 33].

Another issue related to the use of expensive predictors can be summarized as the *explore-exploit tradeoff* introduced in reinforcement learning research [5]. In short, running an ATP guided by a 100-times slower predictor that is only slightly better (possibly due to insufficient previous data for learning) will not only typically solve fewer problems due to much more expensive backtracking but also generate much less data for training the predictor in the next iteration. Hence, given a global time limit allowing many proving/learning iterations over a large set of related problems in a realistic problem-solving setup such as CASC LTB, a faster predictor will in the same time generate much more data to learn from. This in turn often leads to better performance: a slightly weaker ML system trained on much more data will often ultimately outperform a slightly stronger ML system trained on much less data.

1.1 Contributions

In this work we develop combinations of the fast(er) and smart(er) approaches in the context of the learning-guided ENIGMA framework. After giving a summary of ENIGMA in Sect. 2, Sect. 3 introduces our new methods.¹

First, Sect. 3.1 describes a large increase in the speed of neural guidance in ENIGMA. We add an efficient server-based evaluation that uses dedicated GPUs instead of a CPU. When using four commodity GPU cards, this speeds up the neural evaluation of the clauses about four times in real time.

Section 3.2 describes the second addition, motivated by fast weight-based rejection filters used in systems such as E and Prover9. Such methods can be replaced by training fast predictors that implement more intelligent pre-filtering. In the context of ENIGMA, fast(er) is easy to implement by variously parameterized predictors based on gradient-boosted decision trees (GBDTs). Slow(er) models are in those based on graph neural networks (GNNs).

Section 3.3 describes the third addition based on “judging the children by their parents”, i.e., possibly rejecting an inference before it even produces a clause. This grants the machine learning methods greater control of the proof search and saves time by not evaluating all clauses by more expensive methods, also providing a complementary view of the generated clauses.

¹ The E and ENIGMA versions used in this paper can be found at <https://github.com/ai4reason/enigma-gpu-server>.

In Sect. 4 we describe the experimental setting and a large evaluation corpus based on the Mizar Mathematical Library and its MPTP translation. We also present our baseline methods there. Section 5 evaluates the new methods and shows that even in relatively low time limits the methods provide good performance improvements over the previous versions of ENIGMA.

2 Saturation Proving and Its Guidance by ENIGMA

State-of-the-art automated theorem provers (ATP), such as E, Prover9, and Vampire [20], are based on the saturation loop paradigm and the *given clause algorithm* [24]. The input problem, in first-order logic (FOF), is translated into a refutationally equivalent set of clauses, and a search for contradiction is initiated. The ATP maintains two sets of clauses: *processed* (initially empty) and *unprocessed* (initially the input clauses). At each iteration, one unprocessed clause is selected (*given*), and all of the possible inferences with all the processed clauses are generated (typically using resolution, paramodulation, etc.), extending the unprocessed clause set. The selected clause is then moved to the processed clause set. Hence the invariant holds that all the mutual inferences among the processed clauses have been computed.

The selection of the “right” given clause is known to be vital for the success of the proof search. The ENIGMA system [3, 7, 10–12, 14] applies various machine learning methods for given clause selection, learning from a large number of previous successful proof searches. The training data consists of clauses processed during a proof search, labeling the clauses that appear in the discovered proof as *positive*, and the other (thus unnecessary) processed clauses as *negative*.

The first ENIGMA [11] used fast linear classification [4] with hand-crafted clause *features* based on symbol names, representing clauses by fixed-length numeric vectors. Follow-up versions [3, 7, 12, 14] introduced context-based clause evaluation and fast dimensionality reduction by feature hashing, and employed Gradient Boosting Decision Trees (GBDTs), implemented by the XGBoost and LightGBM systems [2, 18]), and Recursive Neural Networks (implemented in PyTorch) as the underlying machine learning methods.

The latest version, ENIGMA Anonymous [10], abstracts from name-based clause representations and provides the best results so far both with GBDTs and Graph Neural Networks (GNNs) [1]. For GBDTs, clauses are again represented by fixed-length vectors based on syntax trees and anonymization is achieved by replacing symbol names by their arities. Our GNN [23] represents clauses by variable-length numeric tensors encapsulating syntax trees as graph structures with symbol names omitted. ENIGMA-GNN evaluates new clauses jointly in larger batches (*queries*) and with respect to a large number of already selected clauses (*context*). The GNN predicts the collectively most useful subset of the clauses in several rounds (*layers*) of message passing. This means that approximative inference rounds done by the GNN are efficiently interleaved with precise symbolic inference rounds done inside E. The GBDT and GNN versions have so far been used separately and only with CPU-based evaluation. In this

work, we add efficiently implemented GPU-based evaluation for the GNN and start to use the two methods cooperatively.

3 Cooperative Filtering: Faster and Smarter

The set of generated clauses in saturation-style ATPs typically grows quadratically with the number of processed clauses. Each new given clause is combined with all compatible previously processed clauses, followed by (possibly expensive) evaluation of all newly generated clauses. In particular, the GNN predictors typically incur a significant evaluation cost per clause. The quadratic growth means that longer ENIGMA-GNN runs may get very slow.

To avoid large memory consumption and similar expensive evaluations in long hint-based Prover9 runs (often taking several days) on the AIM problems [19], Veroff has used weight-based filtering, discarding immediately clauses that reach a certain weight limit. This often helps, but counterexamples are common, and in practice, such schemes often need to be made more complicated.² The three methods that we introduce below are instead targeting this issue by using faster learning-based filtering.

3.1 Fast GNN Evaluation Using a GPU Server

The main weakness of the GNN version of ENIGMA is its slow clause evaluation. In our previous ENIGMA Anonymous experiments [10], we used GPUs for model training, but during the proof search we evaluated the clauses on a single CPU (per each E prover’s instance). This was partly to provide a fair comparison with GBDTs which we also evaluate on a single CPU, but also to avoid large start-up overheads when loading the neural models to a GPU and running with low time limits. Here we instead develop a persistent multi-threaded GPU server that evaluates clauses from multiple E prover runs using multiple GPUs.

The modification is as follows. During the proof search, after computing the tensor representation of the newly generated clauses, an E Prover client sends the tensors (in a JSON text format) over a network socket to a remote server. The client then waits for the server response which provides the scores (GNN evaluations) of the new clauses. This means that the clients are inactive for some time and more of them are needed to saturate the CPUs on the machines (see the detailed experimental discussion in Sect. 5.1). This is typically not a problem due to many instances of E running with different premises and parameters in hammering and CASC LTB scenarios, as well as in many iterations of the learning/proving loop that attempt to solve harder and harder problems over a large problem set.

The remote server, written in Python, is launched before the E clients, loading the GNN model to the (multiple) GPUs in advance. Once the model is loaded

² We thank Bob Veroff for explaining that this is done by gradually lowering the weight limit inside a single longer Prover9 run, and by raising the initial weight limit and slowing down the weight reduction scheme across multiple Prover9 runs.

to the GPUs, the server accepts tensor queries on a designated port, evaluates them on the GPUs, and sends the clause evaluations back to the clients. In more detail, the server is parameterized by the number N (our default is 28) of independent worker threads, the batch size b (our default is 8) and the waiting time T (our default is 0.01 s). The client queries are accumulated in a shared queue that the N worker threads process. Each worker operates in two steps. First, it checks the queue, and if it contains less than b queries, it waits for T seconds. Then it evaluates the first b queries on the queue, or less if there are not enough of them available. Note that when the worker waits or evaluates queries, other workers can process the queue.

The advantage is that the single GNN server amortizes the startup costs and handles queries of many E prover clients and distributes them across multiple GPUs. This means that much larger batches (containing clauses coming from multiple clients) are typically loaded onto the GPUs, amortizing also the relatively high cost of communication with the GPUs. This results in large real time speed-ups over the CPU version, see Sect. 5.1. In our experiments, we run the GPU server and the E clients on the same machine. Hence the network overhead is low because the communication is done over a local loopback interface. In the case of a remote connection, the architecture would benefit from data compression and/or binary data formats to decrease the network overhead. See Sect. 5.1 for the current average sizes of the data exchanged.

3.2 Best of Both Worlds: GNN with GBDT Filtering

While the GPU server evaluation provides a considerable speed up, the evaluation of clauses on a GPU is still relatively costly compared to the GBDT clause evaluation. Hence we develop the following combination of the two methods, where the GBDT is used to pre-filter the clauses for the GNN.

In more detail, the set of clauses to be evaluated by the GNN is first evaluated by a fast GBDT model.³ The GBDT model assigns a score between 0 and 1 to each clause, and only the clauses with scores higher than a selected threshold are sent to the GPU server for evaluation by the GNN. The clauses which are filtered out by the GBDT model are assigned a very high weight inside E Prover, which makes them unlikely to ever be selected for processing. This way we prevent E from incorrectly reporting satisfiability when the good clauses run out.

Several requirements must be met for this filtering to be effective. First, the GBDT filtering model must be small enough so that the evaluation is fast, yet precise enough so that the more important clauses are not mistakenly filtered out too often. Second, the score threshold must be properly fine-tuned, which typically requires experimental grid search on smaller samples. Experiments with a GBDT pre-filtering for a GNN are presented in Sect. 5.2.

³ This feature is implemented for the LightGBM models, which seem more easily tunable for such tasks.

3.3 Parental Guidance: Pruning the Given Clause Loop

We define (*clausal*) *parental guidance* as clause evaluation based on the features of the parents of a clause rather than on the clause itself. Such fast rejection filters often help: in nature, mating is typically highly restricted by various features of parents (e.g., their age, appearance, finances, etc.). Similarly, it does not often happen that clauses from very different parts of mathematics (e.g., differential geometry and graph theory) need to be resolved.

Parental guidance can be seen as “just another filter” of the generated clauses, but its motivation is more radical: The “good old”⁴ given clause loop [24] insists, for completeness reasons, on performing all possible inferences between the processed clauses and the given clause, typically leading to a quadratic growth of the set of generated clauses. However, if we had perfect information about the proof, this would be wasteful and could be replaced by just performing the inferences needed for the proof in each given clause loop. With parental guidance, we instead propose to prune the given clause loop in a soft way: a trained predictor judges the likelihood of the particular inference being needed for the proof. When an inference is deemed useless, the clause is still generated but immediately *frozen* so that it does not have to be evaluated by additional heuristics.

The parental guidance is implemented using GBDTs (our *parental model*), and the filter is directly put inside E’s given clause loop as follows. When E selects a given clause g , E uses term indexes to efficiently determine which clauses can be combined with g to generate new clauses. After generating the clauses, E performs simplifications, removes trivial clauses, evaluates the remaining clauses with the clause evaluation functions, and inserts them into the unprocessed set. The call to the parental model is executed after the clause generation and prior to the simplifications. Clauses generated by paramodulation, which also implements resolution in E, have two parents, and these are judged by the parental model. Clauses whose parents are jointly scored below a chosen threshold are put into the *freezer* set to avoid impairing the completeness of the proof search. Clauses with good parents continue on to the unprocessed set. In case the unprocessed set becomes empty, the frozen clauses are revived and treated as usual.

Note that a naive alternative way to implement parental guidance would be to evaluate each given clause’s compatibility with all previously processed clauses. This would, however, result in many unnecessary GBDT queries and evaluations. Instead, our approach allows E’s indexing to find the typically much smaller set of potential inferences and to limit the parental evaluation to them.⁵

There are various ways to represent the pair of parent clauses for the learning of the parental model. In this work, we evaluate two methods:

⁴ The given clause loop is almost 50 years old as of 2021.

⁵ The efficiency boost obtained by using intelligent indexing is analogous to the boost obtained by using our structure-aware GNN for context-based neural clause selection (Sect. 2) rather than off-the-shelf Transformer models. The latter would quadratically consider interactions of all symbols in the context and query clauses, decreasing the evaluation speed by orders of magnitude, resulting in a very inefficient prover.

1. $\mathcal{P}_{\text{fuse}}$ merges the feature vectors of the parent clauses into one vector, typically by simply adding the feature counts⁶
2. \mathcal{P}_{cat} concatenates the feature vectors of the parent clauses to preserve their information in full.

An interesting future alternative is to include the difference of the parents’ feature vectors in addition to their union and concatenation, which allows the GBDT to choose the most informative features.

4 Experimental Setting and Baselines

4.1 Evaluation Problems and Training Data

All our experiments are performed⁷ on a large benchmark of 57 880 problems⁸ originating from the Mizar Mathematical Library (MML) [16] exported to first-order logic by MPTP [34]. We make use of our ongoing extensive evaluation of many AI/TP methods over this corpus⁹ that measures the overall improvement on this large dataset over the last similar evaluation done in [16]. In these experiments we have significantly extended our previously published results [10].¹⁰ Proofs of 73.5% (more than 40k) Mizar problems have been so far found by learning-guided ATPs, and numerous GBDT and GNN models for ATP guidance have been trained.

In that experiment, all Mizar problems¹¹ are split (in a 90-5-5% ratio) into 3 subsets: (1) 52k problems for *training*, (2) 2896 problems for *development*, and (3) 2896 problems for final evaluation (*holdout*). We use this split here, and additionally we use a random subset of 5792 of the training problems to speed up the training of various experimental methods.

4.2 Baseline ENIGMA Models

Out of the 52k training problems, we were previously able to prove more than 36k problems, obtaining varied numbers of proofs for each problem (ranging from 1 to hundreds). On these 36k problems we train our baseline GBDT and GNN predictors. To balance the contribution of different problems during the training of the predictors, we randomly choose at most 3 proofs for every proved training problem. This yields a set of about 100k proofs, denoted further as the *large* (training) set. When limited to the 5792 random subset of the training problems, this yields 11 748 proofs, denoted further as the *small* training set.

⁶ In some special cases of features, we instead take their maximum/minimum.

⁷ On a server with 36 hyperthreading Intel(R) Xeon(R) Gold 6140 CPU @ 2.30 GHz cores, 755 GB of memory, and 4 NVIDIA GeForce GTX 1080 Ti GPUs.

⁸ http://grid01.ciirc.cvut.cz/~mptp/1147/MPTP2/problems_small_consist.tar.gz.

⁹ https://github.com/ai4reason/ATP_Proofs.

¹⁰ The publication of this large evaluation is in preparation.

¹¹ http://grid01.ciirc.cvut.cz/~mptp/Mizar_eval_final_split.

On the *large* set we train the first baseline predictor denoted by $\mathcal{D}_{\text{large}}$. This is a GBDT model (implemented by the LightGBM framework) trained using the ENIGMA Anonymous clause representation (Sect. 2). The model consists of 150 decision trees of depth 40 with 2048 leaves. This model was selected as it performed best in our previous experiments with standard GBDTs, being able to prove 1377 of the *holdout* problems using a 5 s limit per problem. Additionally, we train another model $\mathcal{D}_{\text{small}}$ only on the *small* set of training problems. The model $\mathcal{D}_{\text{small}}$ is a LightGBM model with 150 trees of depth 30 and with 9728 leaves. The training of $\mathcal{D}_{\text{large}}$ took around 27 min and the training of $\mathcal{D}_{\text{small}}$ around 10 min, both on 30 CPUs. These are relatively low and practical times compared to the training of neural networks.

We also train baseline GNN models on the same data, denoted $\mathcal{G}_{\text{large}}$ and $\mathcal{G}_{\text{small}}$ respectively. The training of $\mathcal{G}_{\text{large}}$ for 45 epochs takes about 15 h on the full set of 100k proofs on a high-end NVIDIA V100 GPU card.¹² It would likely take days when training with CPUs only. We choose for the ATP evaluation the (39th) snapshot that achieves both the best loss (0.2063) and the best weighted accuracy (0.9147) on 5% of the data that we do not use for training. The training of $\mathcal{G}_{\text{small}}$ for 100 epochs takes about 4 h on the *small* set using the same GPU card. We choose for the ATP evaluation the (56th) snapshot that achieves the best loss (0.2988) on 5% of the data that we do not use for training. The weighted accuracy on this set is 0.8685, which is also among the highest values.

In the evaluation we run all our baseline ENIGMA predictors in an equal combination with a strong non-learning E strategy \mathcal{S} . This means that the processed clauses are selected in (equal) turns by ENIGMA and by \mathcal{S} . This *coop* mode has typically worked better than the *solo* mode, where only the ENIGMA predictor is doing the clause selection.

4.3 Training of the Parental GBDT Models

The training data for the parental guidance models are generated by running E using either $\mathcal{D}_{\text{large}}$ or $\mathcal{G}_{\text{large}}$ on the 52k *training* problems with a 30 s time limit and by printing the derivation of all clauses generated during the proof search.¹³ We considered the following two schemes to classify the good pairs of parents and to generate the training data:

1. $\mathcal{P}^{\text{proof}}$ classifies parents of only the proof clauses as *positive* and all other generated clauses as *negative*.
2. $\mathcal{P}^{\text{given}}$ classifies parents of all processed (selected) clauses as *positive* and the unprocessed generated clauses as *negative*.

The rationale behind $\mathcal{P}^{\text{proof}}$ is that every non-proof clause should be pruned if possible. The rationale behind $\mathcal{P}^{\text{given}}$ is that if an effective clause selection strategy, such as $\mathcal{D}_{\text{large}}$, predicted a clause to be useful, then it is probably worth

¹² We use the same GNN hyper-parameters as in [10, 23] with the exception of the number of *layers* that we increase here to 10.

¹³ Using E’s option “--full-deriv”.

generating. However, such data may be confusing as it includes clauses that did not contribute to the proof.

If a pair of parents produces both positive and negative clauses, we consider the pair positive in our implementation. However, this does not happen very often. Based on a survey on the *small* set labeled according to $\mathcal{P}_{\text{fuse}}^{\text{proof}}$, 73% of the problems have no conflict. There are 1519 parents of both positive and negative clauses, 53 359 are positive, and 6086 414 are negative. Under $\mathcal{P}_{\text{fuse}}^{\text{given}}$, 9798 of the parents are mixed, 854 778 are positive, and 5178 592 are negative. In either case, the primary learning task is to identify and prune as many negative clauses as possible without filtering a necessary proof clause by mistake.

One parameter to experimentally tune is the *pos-neg ratio* used in the GBDT training: the ratio of positive and negative examples. The pos-neg ratio is 1:192 over the *large* $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ data, which is more than ten times more than the ratio of the training data for $\mathcal{D}_{\text{large}}$ and $\mathcal{G}_{\text{large}}$. Hence, reducing the pos-neg ratio by randomly sampling negative examples could further boost the training performance.

The parental guidance models are trained using GBDTs. Trained models are evaluated in combination with the GBDT or GNN clause evaluation heuristic using either the $\mathcal{D}_{\text{large}}$ or $\mathcal{G}_{\text{large}}$ model, see Sect. 5.3.

5 Evaluation of the New Methods

5.1 Speedup by Using a GPU Server

First we measure the speedup obtained by evaluating the ENIGMA GNN calls on a separate GPU server. To avoid network latency and for a cleaner comparison, we run both the clients (E/ENIGMA) and the GPU server on the same machine equipped with four NVIDIA GeForce GTX 1080 GPU cards and 36 hyperthreading CPU cores. We configure the server to use all four GPU cards. Its other important parameters are the number of worker threads and the batch size. We experimentally set them to 28 and 8, and we use $\mathcal{G}_{\text{large}}$ for all proof runs.

Comparison of the CPU-only and GPU-server versions is complicated by the fact that the server-based GNN evaluations do not count towards the CPU time taken by E, as reported by the operating system. Still, a comparison using the CPU time is interesting and we include it, using 30 and 60 s CPU limits for the CPU-only version, and a 30 s CPU limit for the client-server version.

Another way to compare the two is by using parallelization, i.e., running many instances of E in parallel. In the client-server version the instances talk to the GPU server simultaneously. We saturate the machine’s CPUs fully for both versions, and run for approximately equal overall real time over the development and holdout sets. This is roughly achieved by using 60 s time limit with 70-fold parallelization for the CPU version, and 30 s time limit with 160-fold parallelization for the client/server version. The CPU version then takes about 27.5 min to finish on the 2896 problems, while the client-server takes about 34 min to finish. Table 1 compares the number of solved problems on the development and holdout sets. The GPU server improves the performance on the development resp. holdout sets by 9.5% resp. 11.5%.

We also compare the average number of generated clauses on the problems that timed out in both versions. In the 60s CPU version it is 16 835, while in the 30s client-server it is 63 305. This is a considerable speedup, achieved by employing the additional custom hardware—our four GPU cards. The average number of GNN queries in the 1358 problems that timed out in the 30s GPU server runs is 243.8, and on average the communication with the GPU server took 155 MB in a timed-out problem. A single GNN query took on average 637 kB.

Table 1. Comparison of the CPU-only GNN ENIGMA with the client-server version using GPUs. All runs are evaluating $\mathcal{G}_{\text{large}}$ on the whole development (D) and holdout (H) datasets. The percentage improvement is computed over the 60 s CPU version that corresponds more closely in real time to the client-server version. All runs use queries of size 256 and contexts of size 768.

Set	Model	Method	Time	Solved	Set	Model	Method	Time	Solved
D	$\mathcal{G}_{\text{large}}$	CPU	30	1311	H	$\mathcal{G}_{\text{large}}$	CPU	30	1301
D	$\mathcal{G}_{\text{large}}$	CPU	60	1380	H	$\mathcal{G}_{\text{large}}$	CPU	60	1371
D	$\mathcal{G}_{\text{large}}$	GPU	30	1511 (+9.5%)	H	$\mathcal{G}_{\text{large}}$	GPU	30	1529 (+11.5%)

5.2 Evaluation of 2-Phase ENIGMA

Small GBDT and Small GNN: In the first experiment we use the GBDT and GNN predictors $\mathcal{D}_{\text{small}}$ and $\mathcal{G}_{\text{small}}$ trained on the *small* subset of the training dataset. We first do a grid search over the parameters on a smaller dataset of 300 development problems. Then we evaluate the best parameters on the development and holdout sets and compare them with the standalone performance of $\mathcal{G}_{\text{small}}$, which is the stronger of the two baselines (Table 2). The best combined methods are then evaluated also in 60s. This gives a relatively fair real-time comparison to the standalone GNN, because the reported CPU times do not include the time taken by the GPU server.¹⁴

Our best combined method solves (in real time) 10.4%, resp. 9.0%, more problems on the development, resp. holdout, set than the standalone GNN. This is a significant improvement, which will likely get even more visible with higher time limits, because of the quadratic growth of the set of generated clauses. The performance improvement over the standalone GBDT model is even larger.

¹⁴ We have made this estimate based on a comparison of real and CPU times done on a set of problems that time out in both methods.

Table 2. Final evaluation of the best combination of $\mathcal{D}_{\text{small}}$ with $\mathcal{G}_{\text{small}}$ on the whole development (D) and holdout (H) datasets.

Set	Model	Thresh.	Time	Query	Context	Solved
D	$\mathcal{G}_{\text{small}}$	–	30	256	768	1251
D	$\mathcal{D}_{\text{small}}$	–	30	–	–	1011
D	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.01	60	512	1024	1381 (+10.4%)
D	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.03	60	512	1024	1371 (+9.6%)
D	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.03	30	512	1024	1341 (+7.2%)
D	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.01	30	512	1024	1339 (+7.0%)
H	$\mathcal{G}_{\text{small}}$	–	30	256	768	1277
H	$\mathcal{D}_{\text{small}}$	–	30	–	–	1002
H	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.01	60	512	1024	1392 (+9.0%)
H	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.03	60	512	1024	1387 (+8.6%)
H	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.01	30	512	1024	1361 (+6.6%)
H	$\mathcal{D}_{\text{small}}+\mathcal{G}_{\text{small}}$	0.03	30	512	1024	1353 (+6.0%)

Large GBDT and Small GNN: In the next experiment, we want to see how much the training of the less expensive model (GBDT) on more data helps. I.e., we replace $\mathcal{D}_{\text{small}}$ with $\mathcal{D}_{\text{large}}$ and keep $\mathcal{G}_{\text{small}}$. This has practical applications in real time, because cheaper ML predictors such as GBDTs are faster to train than more expensive ones such as the GNN. We again first do a grid search over the parameters on a small dataset of 300 development problems. Then we evaluate the best models on the development and holdout sets and compare them with the standalone performance of $\mathcal{D}_{\text{large}}$ and $\mathcal{G}_{\text{small}}$ (Table 3). The best combined methods are then again evaluated also in 60s, which makes it comparable in real time to the standalone GNN model.

Table 3. Final evaluation of the best combination of $\mathcal{D}_{\text{large}}$ and $\mathcal{G}_{\text{small}}$ on the whole development (D) and holdout (H) datasets.

Set	Model	Thresh.	Time	Query	Context	Solved
D	$\mathcal{G}_{\text{small}}$	–	30	256	768	1251
D	$\mathcal{D}_{\text{large}}$	–	30	–	–	1397
D	$\mathcal{D}_{\text{large}}+\mathcal{G}_{\text{small}}$	0.3	60	2048	768	1527 (+9.3%)
D	$\mathcal{D}_{\text{large}}+\mathcal{G}_{\text{small}}$	0.3	30	2048	768	1496 (+7.1%)
H	$\mathcal{G}_{\text{small}}$	–	30	256	768	1277
H	$\mathcal{D}_{\text{large}}$	–	30	–	–	1390
H	$\mathcal{D}_{\text{large}}+\mathcal{G}_{\text{small}}$	0.3	60	2048	768	1494 (+7.5%)
H	$\mathcal{D}_{\text{large}}+\mathcal{G}_{\text{small}}$	0.3	30	2048	768	1467 (+5.5%)

Our best combined method solves (in CPU time) 7.1%, resp. 5.5%, more problems on the development, resp. holdout, set than the standalone GBDT. For the GNN, this is (in real time) 9.3% resp. 7.5%. These are smaller gains than in the previous $\mathcal{D}_{\text{small}} + \mathcal{G}_{\text{small}}$ scenario, most likely because the stronger predictor dominates here. Also note that the large query (2048) used in our strongest model is typically diminished a lot by the GBDT pre-filter, resulting in average query sizes after the GBDT pre-filtering of 256–512.

Large GBDT and Large GNN: Finally, we evaluate the large setting, using the GBDT and GNN predictors $\mathcal{D}_{\text{large}}$ and $\mathcal{G}_{\text{large}}$ trained on the full training dataset. Again, we first do a grid search over the parameters on the small set of 300 development problems. Then we evaluate the best parameters on the development and holdout sets, and we compare them with the standalone performance of $\mathcal{D}_{\text{large}}$ and $\mathcal{G}_{\text{large}}$ (Table 4). The improvements on the development, resp. holdout, set is 9.1%, resp. 7.3%, in real time, and 6.9%, resp. 4.8%, when using CPU time. The E auto-schedule solves in 30s (CPU time) 1020 of the holdout problems. Our strongest 2-phase method solves 1602 of these problems in the same CPU time, i.e., 57.1% more problems.

Table 4. Final evaluation of the best combination of $\mathcal{D}_{\text{large}}$ and $\mathcal{G}_{\text{large}}$ on the whole development (D) and holdout (H) datasets.

Set	Model	Thresh.	Time	Query	Context	Solved
D	$\mathcal{G}_{\text{large}}$	–	30	256	768	1511
D	$\mathcal{D}_{\text{large}}$	–	30	–	–	1397
D	$\mathcal{D}_{\text{large}} + \mathcal{G}_{\text{large}}$	0.1	60	1024	768	1648 (+9.1%)
D	$\mathcal{D}_{\text{large}} + \mathcal{G}_{\text{large}}$	0.1	30	1024	768	1615 (+6.9%)
H	$\mathcal{G}_{\text{large}}$	–	30	256	768	1529
H	$\mathcal{D}_{\text{large}}$	–	30	–	–	1390
H	$\mathcal{D}_{\text{large}} + \mathcal{G}_{\text{large}}$	0.1	60	1024	768	1640 (+7.3%)
H	$\mathcal{D}_{\text{large}} + \mathcal{G}_{\text{large}}$	0.1	30	1024	768	1602 (+4.8%)

5.3 Evaluation of the Parental Guidance Combined with $\mathcal{D}_{\text{large}}$

The parameters for parental guidance models are explored via a series of grid searches to reduce the number of combinations. Initially, we only use $\mathcal{D}_{\text{large}}$ in conjunction with the parental models. First, the training data classification schemes, $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ and $\mathcal{P}_{\text{fuse}}^{\text{given}}$, are compared with a grid search over the pos-neg reduction ratio. The best combination of reduction ratio and classification scheme is used to perform a grid search over LightGBM parameters for $\mathcal{P}_{\text{fuse}}$. Next, reduction ratio and LightGBM parameter grid searches are done with the \mathcal{P}_{cat} featurization method data, starting with the best $\mathcal{P}_{\text{fuse}}$ parameters from the previous

experiments. Every model is evaluated with the same set of nine parental filtering thresholds $\{0.005, 0.01, 0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5\}$. The grid searches are done over the 300 problem development set and run for 30s. On this dataset, $\mathcal{D}_{\text{large}}$ solves 159 problems.

Pos-Neg Reduction Ratio Tuning (Merge): The first grid search examines the pos-neg reduction ratio denoted as ρ . Before the reduction, the average pos-neg ratio for $\mathcal{P}_{\text{fuse}}^{\text{given}}$ is 1:9.2 and the average for $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ is 1:191.8. We reduce the pos-neg ratio to a given ρ by randomly sampling the negative examples on a problem-specific basis. This means that the average pos-neg ratio over the whole dataset is typically a bit smaller than ρ . For example, using $\rho = 4$ on the $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ results in an average of 3.95 times more negative than positive examples. Both $\mathcal{P}_{\text{fuse}}^{\text{given}}$ and $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ are tested using $\rho \in \{-, 1, 2, 4, 8, 16\}$ where “-” denotes using the full training dataset. We use the best LightGBM model parameters discovered during prototyping of the parental guidance features: the parameters are 50 trees of depth 13 with 1024 leaves.

Table 5 shows that the reduction ratio makes significant difference for the $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ data and almost none for $\mathcal{P}_{\text{fuse}}^{\text{given}}$ data, which is probably because the $\mathcal{P}_{\text{fuse}}^{\text{given}}$ data are already reasonably balanced. Moreover, parental guidance seems to perform better with $\mathcal{P}_{\text{fuse}}^{\text{proof}}$ data than $\mathcal{P}_{\text{fuse}}^{\text{given}}$ data, probably because mistakes of $\mathcal{D}_{\text{large}}$ are included in the training data. In the following experiments, only the $\mathcal{P}^{\text{proof}}$ classification scheme is used (so the prefix is dropped).

Table 5. The best threshold for each tested reduction ratio. The threshold of 0.03 was identical to 0.05 for all tested ratios with $\mathcal{P}_{\text{fuse}}^{\text{given}}$, whereas there are no ties among thresholds for $\mathcal{P}_{\text{fuse}}^{\text{proof}}$.

$\rho_{\text{fuse}}^{\text{given}}$	-	1	2	4	8	16	$\rho_{\text{fuse}}^{\text{proof}}$	-	1	2	4	8	16
Threshold	0.05	0.05	0.05	0.05	0.05	0.05	Threshold	0.005	0.2	0.2	0.2	0.2	0.2
Solved	161	161	161	161	161	160	Solved	111	164	163	165	162	164

Table 6. The best threshold for each tested reduction ratio of \mathcal{P}_{cat} .

ρ_{cat}	-	1	2	4	8	16
Threshold	0.5	0.1	0.05	0.3	0.1	0.05
Solved	117	168	170	168	173	169

LightGBM Parameter Tuning (Merge): Next we perform the second grid search over the LightGBM training hyper-parameters for $\mathcal{P}_{\text{fuse}}$, fixing $\rho = 4$ as it performed best. We try the following values for the three main hyper-parameters, namely, for the number of trees in a model, the maximum number of tree leaves, and the maximum tree depth:

$$\begin{aligned} \text{trees} &\in \{50, 100, 150\} \\ \text{leaves} &\in \{1024, 2048, 4096, 8192, 16384\} \\ \text{depth} &\in \{13, 40, 60, 256\} \end{aligned}$$

The best model for $\mathcal{P}_{\text{fuse}}$ solves 171 problems and consists of 100 trees, with the depth 40, and 8192 leaves, and a threshold of 0.05. Another eight models solve 169 problems. We also tested these parameters to find a better model for $\mathcal{P}_{\text{fuse}}^{\text{given}}$, which solves 163 problems with $\rho = 8$ and a threshold of 0.1.

Pos-Neg Reduction Ratio Tuning (Concat): This grid search uses the best LightGBM hyper-parameters for $\mathcal{P}_{\text{fuse}}$ to test the same reduction ratios and thresholds for \mathcal{P}_{cat} . Table 6 shows that \mathcal{P}_{cat} outperforms $\mathcal{P}_{\text{fuse}}$ and $\rho = 8$ is the best. Reducing the negatives is even more important here.

LightGBM Parameter Tuning (Concat): The grid search for the \mathcal{P}_{cat} data is done over the following hyper-parameters:

$$\begin{aligned} \text{trees} &\in \{50, 100, 150, 200\} \\ \text{leaves} &\in \{1024, 2048, 4096, 8192, 16384, 32768\} \\ \text{depth} &\in \{13, 40, 60, 256, 512\} \end{aligned}$$

The upper limits have increased compared to the $\mathcal{P}_{\text{fuse}}$ grid-search because one of the best models had 150 trees of depth 256, placing it at the edge of the grid. The best models solve 174–175 problems. These are evaluated on the full development set (Table 7). The larger models seem to work best with a threshold of 0.05 and the smaller models with a threshold of 0.2, which is likely because they can be less precise. The full distribution of the results can be seen in Fig. 1. The number of parameter configurations that outperform the baseline suggests that parental guidance is an effective method.

Table 7. The best \mathcal{P}_{cat} models with $\rho = 8$.

Trees	Depth	Leaves	Threshold	Solved (300)	Solved (D)
200	60	4096	0.05	175	1557
200	512	4096	0.05	175	1561
200	256	4096	0.05	174	1558
150	512	1024	0.2	174	1568
150	256	1024	0.2	174	1556
100	60	8192	0.05	174	1571
100	40	2048	0.2	174	1544
100	40	2048	0.1	174	1544

Table 8. Final 30s evaluation on small trains (T), development (D), and holdout (H) compared with $\mathcal{D}_{\text{large}}$.

Model	Threshold	Solved (T)	Solved (D)	Solved (H)
$\mathcal{D}_{\text{large}}$	–	3269	1397	1390
$\mathcal{P}_{\text{fuse}}^{\text{given}} + \mathcal{D}_{\text{large}}$	0.05	3302 (+1.0%)	1411 (+1.0%)	1417 (+1.9%)
$\mathcal{P}_{\text{fuse}}^{\text{proof}} + \mathcal{D}_{\text{large}}$	0.1	3389 (+3.7%)	1489 (+6.6%)	1486 (+6.9%)
$\mathcal{P}_{\text{cat}} + \mathcal{D}_{\text{large}}$	0.05	3452 (+5.6%)	1571 (+12.4%)	1553 (+11.7%)

Finally we evaluate the best models on the small training, development, and holdout sets, and we compare them with the standalone performance of $\mathcal{D}_{\text{large}}$ (Table 8). Parental guidance achieves a significant improvement in performance on all datasets, solving 11.7% more on the holdout set. It is interesting to note that the improvement is greater on the development and holdout sets than on the training set. For parental guidance it seems superior to classify only *proof clauses* as positive examples. This is most likely due to LightGBM being confused by processed clauses that did not contribute to any proof. The method of concatenating the parent clause feature vectors (\mathcal{P}_{cat}) seems far superior to merging them ($\mathcal{P}_{\text{fuse}}$). This is likely because merging the features is lossy and the order of the parents matters when performing inferences.

The results indicate that pruning clauses prior to clause evaluation is helpful. ENIGMA models tend to run best in equal combination with a strong E strategy, but this means they have no control over 50% of the clauses selected for processing. The ability to filter which clauses the strong E strategy can evaluate and select may be part of the strength behind parental guidance.

5.4 Parental Guidance with $\mathcal{G}_{\text{large}}$ and 3-Phase ENIGMAS

We also explore a limited number of the most useful hyper-parameters from Sects. 5.3 and 5.2 to combine the parental filtering with ENIGMA-GNN using $\mathcal{G}_{\text{large}}$ and to create a 3-phase ENIGMA. We train a new LightGBM parental filtering model on the \mathcal{P}_{cat} data generated by running $\mathcal{G}_{\text{large}}$, using $\rho = 8$, trees = 100, leaves = 8192, and depth = 60. The grid search on the 300 development problems leads to the best threshold values of 0.005 and 0.01 when using context = 768 and query = 256 for ENIGMA-GNN with $\mathcal{G}_{\text{large}}$.

The version with the 0.01 threshold then reaches so far the highest value of 1621 development problems in 30s CPU time. This is 50 more than the best parental result using $\mathcal{D}_{\text{large}}$ and 6 more than the best 2-phase result. On the holdout set this setting yields 1623 problems, i.e., 70 more than the best $\mathcal{D}_{\text{large}}$ parental result and 21 more than the best 2-phase result.

Finally, we explore 3-phase ENIGMAS, i.e., combinations of all the methods developed in this work. This means that we first use the parental guidance filtering, followed by the 2-phase evaluation which in turn uses the GPU server. This implies a higher evaluation cost, since both the parental and the first-stage LightGBM models are loaded on startup and are used to filter the clauses.

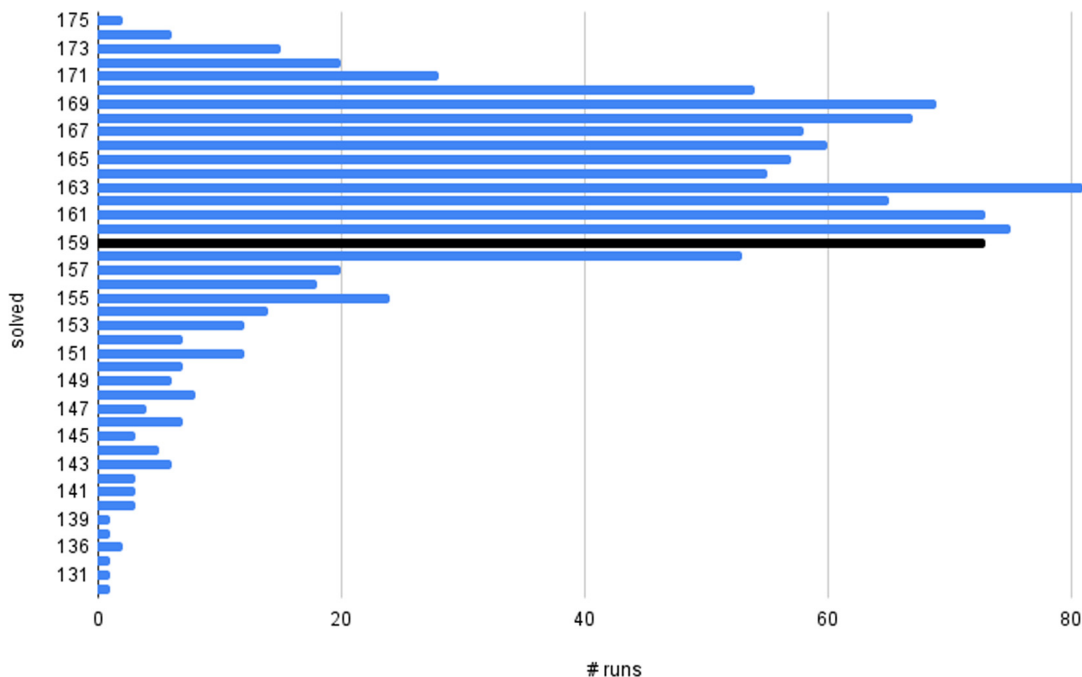


Fig. 1. The number of settings (and runs) corresponding to each number of solutions for the \mathcal{P}_{cat} grid search. The black bar is 159, the number of problems solved by $\mathcal{D}_{\text{large}}$. Only 154 (20%) of the runs interfere with $\mathcal{D}_{\text{large}}$'s performance and solve fewer problems. These runs largely consist of the thresholds, $\{0.3, 0.4, 0.5\}$, but the only parameter whose majority of runs score below $\mathcal{D}_{\text{large}}$ is a threshold of 0.5. The outliers tend to be larger models.

We only tune the parental threshold and context and query values, keeping the 2-phase threshold fixed at 0.1. The best result is again obtained by setting the parental threshold to 0.01, context = 768 and query = 256. This solves 1631 resp. **1632** of the development resp. holdout problems in 30 s CPU time. This is our ultimate result, which is exactly 60% higher than the 1020 problems solved by E's auto-schedule in 30 s CPU time. It is also 17.4% higher than the best ENIGMA result prior to this work (1390 by standalone $\mathcal{D}_{\text{large}}$).

6 Conclusion and Examples

We have described several additions to the ENIGMA system. The new methods combine fast(er) and smart(er) clause evaluation using ENIGMA's parameterizable learning-based setting. The GPU server allows much faster runs of the neurally-guided ENIGMA, improving its real-time performance by about 10%. The parental guidance allows one to train clause evaluation differently from standard ENIGMA, providing an improvement of 11.7% on the holdout set. Both when training on small and on large datasets, the 2-phase methods provide good improvements on the holdout sets (9% and 7.3%) over the strongest standalone methods. The methods are adjustable and they will likely lead to even higher improvements in longer runtimes, due to the typically quadratic growth of the

set of generated clauses in saturation-style ATPs. Our strongest 3-phase method improves E’s auto-schedule on the holdout set by 60% in 30 s and our best prior ENIGMA result by 17.4%.

Several examples of the new proofs produced only by the methods developed here are available on our project’s web page. Theorem `INTEGR13:27`¹⁵ about the differentiation of $-\cot(\ln(x))$ needed 3904 nontrivial given clause loops and 38826 nontrivial generated clauses, taking only 18 s with the 2-phase ENIGMA. This can be compared to the previous related theorem `FDIFF_7:36`¹⁶ (differentiation of $\exp(\cos(x))$) done in the old setting, taking 28.4 s to do only 1284 nontrivial given clause loops and 13287 nontrivial generated clauses. Other examples include a 486-long proof¹⁷ of a theorem about integrals done only in 41 s with the 2-phase ENIGMA evaluating 100k clauses, or a 259-long computational proof¹⁸ about Fermat primes found in 11 s while evaluating 52k clauses. Such proofs are found despite hundreds of redundant axioms, by using new combinations of faster and smarter trained ENIGMAS that efficiently guide the search.

Acknowledgments. This work was partially supported by the ERC Consolidator grant *AI4REASON* no. 649043 (ZG, JJ, and JU), the European Regional Development Fund under the Czech project *AI&Reasoning* no. CZ.02.1.01/0.0/0.0/15_003/0000466 (ZG, JU, KC), the ERC Starting Grant *SMART* no. 714034 (JJ, MO), and by the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902 (JJ).

References

1. Abadi, M., et al.: TensorFlow: large-scale machine learning on heterogeneous distributed systems. arXiv preprint [arXiv:1603.04467](https://arxiv.org/abs/1603.04467) (2016)
2. Chen, T., Guestrin, C.: XGBoost: a scalable tree boosting system. In: Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, KDD 2016, pp. 785–794. ACM, New York (2016)
3. Chvalovský, K., Jakubův, J., Suda, M., Urban, J.: ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In: Fontaine, P. (ed.) CADE 2019. LNCS (LNAI), vol. 11716, pp. 197–215. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_12
4. Fan, R.-E., Chang, K.-W., Hsieh, C.-J., Wang, X.-R., Lin, C.-J.: LIBLINEAR: a library for large linear classification. *J. Mach. Learn. Res.* **9**, 1871–1874 (2008)
5. Gittins, J.C.: Bandit processes and dynamic allocation indices. *J. Roy. Stat. Soc. Ser. B (Methodol.)* **41**, 148–177 (1979)
6. Goertzel, Z., Jakubův, J., Schulz, S., Urban, J.: ProofWatch: watchlist guidance for large theories in E. In: Avigad, J., Mahboubi, A. (eds.) ITP 2018. LNCS, vol. 10895, pp. 270–288. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-94821-8_16

¹⁵ https://github.com/ai4reason/ATP_Proofs/#differentiation---cot--ln-x--1--x--sin-ln-x2-.

¹⁶ https://github.com/ai4reason/ATP_Proofs/#differentiation-exp_r--cos--x----exp_r--cos--x--sin-x.

¹⁷ https://github.com/ai4reason/ATP_Proofs/#integral-chi-aa-is-integrable--integral-chi-aa--vol-a-486-long-atp-proof-from-63-premises.

¹⁸ https://github.com/ai4reason/ATP_Proofs/#17-is-prime.


7. Goertzel, Z., Jakubův, J., Urban, J.: ENIGMAWatch: ProofWatch meets ENIGMA. In: Cerrito, S., Popescu, A. (eds.) TABLEAUX 2019. LNCS (LNAI), vol. 11714, pp. 374–388. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29026-9_21
8. Gottlob, G., Sutcliffe, G., Voronkov, A. (eds.): Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, 16–19 October 2015, Volume 36 of EPiC Series in Computing. EasyChair (2015)
9. Hillenbrand, T.: Citius altius fortius: lessons learned from the theorem prover WALDMEISTER. ENTCS **86**(1), 9–21 (2003)
10. Jakubův, J., Chvalovský, K., Olšák, M., Piotrowski, B., Suda, M., Urban, J.: ENIGMA anonymous: symbol-independent inference guiding machine (system description). In: Peltier, N., Sofronie-Stokkermans, V. (eds.) IJCAR 2020. LNCS (LNAI), vol. 12167, pp. 448–463. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-51054-1_29
11. Jakubův, J., Urban, J.: ENIGMA: efficient learning-based inference guiding machine. In: Geuvers, H., England, M., Hasan, O., Rabe, F., Teschke, O. (eds.) CICM 2017. LNCS (LNAI), vol. 10383, pp. 292–302. Springer, Cham (2017). https://doi.org/10.1007/978-3-319-62075-6_20
12. Jakubův, J., Urban, J.: Enhancing ENIGMA given clause guidance. In: Rabe, F., Farmer, W.M., Passmore, G.O., Youssef, A. (eds.) CICM 2018. LNCS (LNAI), vol. 11006, pp. 118–124. Springer, Cham (2018). https://doi.org/10.1007/978-3-319-96812-4_11
13. Jakubův, J., Urban, J.: Hierarchical invention of theorem proving strategies. AI Commun. **31**(3), 237–250 (2018)
14. Jakubův, J., Urban, J.: Hammering MizAR by learning clause guidance. In: Harrison, J., O’Leary, J., Tolmach, A. (eds.) 10th International Conference on Interactive Theorem Proving, ITP 2019, Portland, OR, USA, 9–12 September 2019, LIPIcs, vol. 141, pp. 34:1–34:8. Schloss Dagstuhl - Leibniz-Zentrum für Informatik (2019)
15. Kaliszyk, C.: Efficient low-level connection tableaux. In: De Nivelle, H. (ed.) TABLEAUX 2015. LNCS (LNAI), vol. 9323, pp. 102–111. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-24312-2_8
16. Kaliszyk, C., Urban, J.: MizAR 40 for MizAR 40. J. Autom. Reasoning **55**(3), 245–256 (2015)
17. Kaliszyk, C., Urban, J., Michalewski, H., Olšák, M.: Reinforcement learning of theorem proving. In: Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, Montréal, Canada, 3–8 December 2018, pp. 8836–8847 (2018)
18. Ke, G., et al.: LightGBM: a highly efficient gradient boosting decision tree. In: NIPS, pp. 3146–3154 (2017)
19. Kinyon, M., Veroff, R., Vojtěchovský, P.: Loops with abelian inner mapping groups: an application of automated deduction. In: Bonacina, M.P., Stickel, M.E. (eds.) Automated Reasoning and Mathematics. LNCS (LNAI), vol. 7788, pp. 151–164. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-36675-8_8
20. Kovács, L., Voronkov, A.: First-order theorem proving and VAMPIRE. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 1–35. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-39799-8_1
21. Loos, S.M., Irving, G., Szegedy, C., Kaliszyk, C.: Deep network guided proof search. In: Eiter, T., Sands, D. (eds.) LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, 7–12 May 2017, EPiC Series in Computing, vol. 46, pp. 85–105. EasyChair (2017)

22. McCune, W.: Experiments with discrimination-tree indexing and path indexing for term retrieval. *J. Autom. Reasoning* **9**(2), 147–167 (1992)
23. Olsák, M., Kaliszyk, C., Urban, J.: Property invariant embedding for automated reasoning. In: De Giacomo, G., et al. (eds.) *ECAI 2020–24th European Conference on Artificial Intelligence, 29 August–8 September 2020, Santiago de Compostela, Spain, 29 August–8 September 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020), Frontiers in Artificial Intelligence and Applications*, vol. 325, pp. 1395–1402. IOS Press (2020)
24. Overbeek, R.A.: A new class of automated theorem-proving algorithms. *J. ACM* **21**(2), 191–200 (1974)
25. Rath, T., Otten, J.: randoCoP: randomizing the proof search order in the connection calculus. In: Konev, B., Schmidt, R.A., Schulz, S. (eds.) *Proceedings of the First International Workshop on Practical Aspects of Automated Reasoning, Sydney, Australia, 10–11 August 2008, CEUR Workshop Proceedings*, vol. 373. CEUR-WS.org (2008)
26. Ruhdorfer, C., Schulz, S.: Efficient implementation of large-scale watchlists. In: Fontaine, P., Korovin, K., Kotsireas, I.S., Rümmer, P., Tourret, S. (eds.) *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 Co-Located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June–July 2020 (Virtual), CEUR Workshop Proceedings*, vol. 2752, pp. 120–133. CEUR-WS.org (2020)
27. Schäfer, S., Schulz, S.: Breeding theorem proving heuristics with genetic algorithms. In: Gottlob et al. [8], pp. 263–274
28. Schulz, S.: Fingerprint indexing for paramodulation and rewriting. In: Gramlich, B., Miller, D., Sattler, U. (eds.) *IJCAR 2012. LNCS (LNAI)*, vol. 7364, pp. 477–483. Springer, Heidelberg (2012). https://doi.org/10.1007/978-3-642-31365-3_37
29. Schulz, S.: System description: E 1.8. In: McMillan, K., Middeldorp, A., Voronkov, A. (eds.) *LPAR 2013. LNCS*, vol. 8312, pp. 735–743. Springer, Heidelberg (2013). https://doi.org/10.1007/978-3-642-45221-5_49
30. Schulz, S., Cruanes, S., Vukmirović, P.: Faster, higher, stronger: E 2.3. In: Fontaine, P. (ed.) *CADE 2019. LNCS (LNAI)*, vol. 11716, pp. 495–507. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-29436-6_29
31. Stickel, M.E.: The path-indexing method for indexing terms. Technical report, SRI International Menlo Park CA Artificial Intelligence Center (1989)
32. Sutcliffe, G., Suttner, C.B.: The state of CASC. *AI Commun.* **19**(1), 35–48 (2006)
33. Sutcliffe, G., Urban, J.: The CADE-25 automated theorem proving system competition - CASC-25. *AI Commun.* **29**(3), 423–433 (2016)
34. Urban, J.: MPTP 0.2: design, implementation, and initial experiments. *J. Autom. Reasoning* **37**(1–2), 21–43 (2006)
35. Urban, J.: BliStr: the blind strategymaker. In: Gottlob et al. [8], pp. 312–319
36. Urban, J., Vyskočil, J., Štěpánek, P.: MaLeCoP machine learning connection prover. In: Brunnler, K., Metcalfe, G. (eds.) *TABLEAUX 2011. LNCS (LNAI)*, vol. 6793, pp. 263–277. Springer, Heidelberg (2011). https://doi.org/10.1007/978-3-642-22119-4_21
37. Veroff, R.: Using hints to increase the effectiveness of an automated reasoning program: case studies. *J. Autom. Reasoning* **16**(3), 223–239 (1996)
38. Voronkov, A.: The anatomy of Vampire implementing bottom-up procedures with code trees. *J. Autom. Reasoning* **15**(2), 237–265 (1995)

The Isabelle ENIGMA

Zarathustra A. Goertzel 

Czech Technical University in Prague, Czech Republic

Jan Jakubův 

Czech Technical University in Prague, Czech Republic

Universität Innsbruck, Austria

Cezary Kaliszyk 

Universität Innsbruck, Austria

Miroslav Olšák 

Institut des Hautes Études Scientifiques, Bures-sur-Yvette, France

Jelle Piepenbrock 

Czech Technical University in Prague, Czech Republic

Radboud University, Nijmegen, The Netherlands

Josef Urban 

Czech Technical University in Prague, Czech Republic

Abstract

We significantly improve the performance of the E automated theorem prover on the Isabelle Sledgehammer problems by combining learning and theorem proving in several ways. In particular, we develop targeted versions of the ENIGMA guidance for the Isabelle problems, targeted versions of neural premise selection, and targeted strategies for E. The methods are trained in several iterations over hundreds of thousands untyped and typed first-order problems extracted from Isabelle. Our final best single-strategy ENIGMA and premise selection system improves the best previous version of E by 25.3% in 15 seconds, outperforming also all other previous ATP and SMT systems.

2012 ACM Subject Classification Theory of computation → Automated reasoning

Keywords and phrases E Prover, ENIGMA, Premise Selection, Isabelle/Sledgehammer

Digital Object Identifier 10.4230/LIPIcs.ITP.2022.16

Supplementary Material *Dataset*: https://github.com/ai4reason/isa_enigma_paper

Funding This work was partially supported by the European Regional Development Fund under the Czech project AI&Reasoning no. CZ.02.1.01/0.0/0.0/15_003/0000466 (ZG, JP, JU), the ERC Starting Grant *SMART* no. 714034 (JJ, CK), Amazon Research Awards (JP, JU) and by the Czech MEYS under the ERC CZ project *POSTMAN* no. LL1902 (JJ, JP).

1 Introduction

Formal verification in interactive theorem provers (ITPs) increasingly benefits from general proof automation in the form of *hammers* [7] and guided tactical provers [4, 13, 36]. In particular, the Sledgehammer system [8] for Isabelle is today perhaps the most widely used strong general proof automation system in ITP. In the recent years, machine learning and related AI methods for proof automation have also been significantly developed [48]. Such methods are relevant for hammers in at least three ways: (i) learning-based *premise selection* [2, 3, 12, 37, 39, 40] usually improves the heuristic filters used by the hammers, (ii) learning-based *internal guidance* of the automated theorem provers (ATPs) used for the heavy lifting in the hammers usually improves on heuristic guidance of ATPs [16, 22, 24, 26, 27, 29, 41, 49], and (iii) targeted theorem proving strategies developed by automated strategy invention systems often improve on manually designed ATP strategies [20, 23, 42, 47].



© Zarathustra A. Goertzel, Jan Jakubův, Cezary Kaliszyk, Miroslav Olšák, Jelle Piepenbrock, and Josef Urban;

licensed under Creative Commons License CC-BY 4.0

13th International Conference on Interactive Theorem Proving (ITP 2022).

Editors: June Andronick and Leonardo de Moura; Article No. 16; pp. 16:1–16:21

Leibniz International Proceedings in Informatics



LIPICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

Most recent versions of such AI/TP methods have been developed mainly on a fixed Mizar/MPTP corpus [28], to allow easy comparisons with previously developed methods. In particular, there the strongest 3-phase single-strategy version of the ENIGMA system (based on E [43, 44]) proves 56.35% of the holdout (test) toplevel theorems in 30s when using human-selected premises [16]. In higher time limits and by combining human and learning-based premise selection, ENIGMA and Vampire [32] today prove 75% of the toplevel Mizar theorems.¹ These are good reasons for transferring the methods to other ITP hammers.

A direct motivation for developing such AI/TP methods for Isabelle was a recent request from the Sledgehammer developers for an optimized version of ENIGMA for their GRUNGE-style [9] evaluation of multiple ATP systems and formats [11]. While it was not possible to do the work described in this paper on a two-week’s notice, it prompted us to start exporting and analyzing the Isabelle datasets and developing suitable methods and systems for them.

1.1 Contributions

We significantly improve the performance of the E automated theorem prover on the Isabelle Sledgehammer problems by combining learning and theorem proving in several ways. First, in Section 2 we extract two large datasets of untyped first-order (FOF) and many-sorted first-order (TFF, TF0) Isabelle Sledgehammer problems, using the Isabelle tool Mirabelle. This results in almost 300000 aligned problems in each of the exports, spanning in total 1902 Isabelle theory files and covering a large number of topics in mathematics and formal verification. To our knowledge, these are so far the largest corpora of Isabelle Sledgehammer problems available today for training and evaluation of AI/TP systems. Section 2.1 analyzes the corpora, showing that they significantly differ from other large AI/TP datasets such as the Mizar/MPTP toplevel theorems [28] and the HOL4/GRUNGE toplevel theorems [9].

In Section 3, we find optimized E strategies and parameters for the corpora, which already improve on standard E on the problems. They are suitable also for combinations with the ENIGMA guidance, which is introduced in Section 4. We also describe there several extensions to ENIGMA that were developed to handle the Isabelle untyped and typed problems. Section 5 discusses the neural premise selection that we use and its extensions for the typed Isabelle setting. Section 6 evaluates the methods in several loops interleaving proving and learning from the proofs. Our ultimate performance results are: (i) improving in 15s the original E auto-schedule with the MePo filter by 25.3%, when using a single ENIGMA strategy with the best neural predictor, (ii) considerably improving over all other ATPs and SMTs by a single ENIGMA strategy combined with the best neural predictor, (iii) improving the performance of all other systems by using the neural predictor, and (iv) outperforming with ENIGMA all other ATPs and SMTs even when they are combined with our predictor.

2 Isabelle Problems

To train and evaluate the Isabelle ENIGMA, we need a dataset of Sledgehammer problems, which correspond to the proof obligations that users encounter when using Isabelle as an interactive prover. We decided to focus on all proof-intermediate goals visible to the users. This task has been tried as early as in the first versions of the MPTP system [46]. In Isabelle, it has been known as the “Judgement Day” evaluation, based on the paper with that title [8]. We have used the Isabelle/Mirabelle infrastructure to export all the problems encountered

¹ https://github.com/ai4reason/ATP_Proofs/blob/master/75percent_announce.md

when building 179 Isabelle sessions. These sessions originate from 75 sessions distributed with Isabelle 2021-1, 80 selected sessions from the AFP [6], as well as 24 sessions distributed as part of IsaFoR [45]. All the sessions include in total 1902 Isabelle theory files. The sessions with most problems can be categorized as Analysis, Algebra, Java Semantics, Category Theory, Protocols, Term Rewriting, and Probability Theory with the largest 26 sessions listed in Table 1.

■ **Table 1** The largest included sessions and their respective problem numbers.

HOL-Nonstandard-Analysis	1699	Groebner-Macaulay	4227
Category2	1776	HOL-ODE-Numerics	4422
Poincare-Bendixson	1983	HOL-MicroJava	5183
HOL-Number-Theory	2071	HOL-Auth	5304
MonoidalCategory	2238	HOL-Complex-Analysis	5489
HOL-Cardinals	2268	Groebner-Bases	5710
Core-DOM	2280	HOL-Computational-Algebra	6280
HOL-IMP	2324	Jordan-Normal-Form	6786
HOL-Data-Structures	2353	Category3	6818
Dirichlet-Series	2435	HOL-Probability	6954
Slicing	2517	HOL-Decision-Proc	7103
HOLCF	2524	CR	7341
Formal-SSA	2899	HOL-Bali	7804
HOL-UNITY	2938	HOL	7818
HOL-Homology	3022	Goedel-HFSet-Semanticless	8697
HOL-ex	3047	HOL-Algebra	9674
CTRS	3328	HRB-Slicing	10052
HOL-Hoare-Parallel	3733	Jinja	11520
Signature-Groebner	3762	HOL-Library	15627
Valuation	3786	Bicategory	16965
Ordinary-Differential-Equations	3885	HOL-Nominal-Examples	17145
Smith-Normal-Form	4045	Group-Ring-Module	19718
Differential-Dynamic-Logic	4158	HOL-Analysis	44172

The Sledgehammer export allows multiple encodings of types, lambdas, and other options [5]. Since we are interested in the performance of learning-based first-order ATPs, we exported the problems in two first-order formats: TFF (also called TF0), i.e., many-sorted first-order logic, and FOF, i.e. untyped first-order logic. For all problems we pre-selected 512 relevant premises using the heuristic MePo filter [35] before the translation. This slightly overshoots the best performance (256 premises) obtained by most of the top systems² on the FOF and TFF problems in the recent Sledgehammer evaluation [11]. We use 512 premises because the heuristic MePo filter is known to be weaker than state-of-the-art selection systems (possibly pruning out some good premises too early), and also because the 512-premise results of the best systems in [11] are nearly identical³ to the 256-premise results.⁴

² Vampire is an exception: in [11] it is best with 512 premises, likely due to its optimized SInE filter [19].

³ In particular, CVC5 - the winner in [11] - is only 3.7% (2626/2533) stronger with 256 premises.

⁴ We could have used also 1024 premises, however already with 512 premises the datasets are becoming very large, making also the training of the ML systems technically challenging.

For the other parameters, for E and Vampire we used the ones corresponding to the slice selected when no-slicing is used for a particular prover. Additionally, when extracting the FOF problems, we used the parameters used for such a slice in first-order E in the previous Isabelle version. These parameters have been optimized by the Isabelle/Sledgehammer developers based on experiments described in previous papers, e.g., [5]. To ease comparison with [11], we use the polymorphic $g??$ [11] encoding together with lambda-lifting [21] for FOF and the native monomorphic encoding with lambda-lifting for TFF0.

Since the Mirabelle export has occasional problems with some theories and encodings (theory compilation fails or does not terminate with a particular export), we initially get different numbers of problems for the FOF (293587) and TFF (386619) exports. To align the two exports, we remove the non-overlapping problems, thus obtaining 276363 problems both in FOF and TFF that correspond to each other. As usual in machine learning, we then divide this dataset into the *training*, *development* (validation) and *holdout* (ultimate testing) parts. This is done by randomly shuffling the list of the problems and dividing the shuffled list 90:5:5. This means that we have 248727 problems to train our systems on, 13818 development problems for controlling the hyperparameters of the learning and building the best portfolios, and 13818 holdout problems on which the trained systems are ultimately evaluated. We also sometimes use a 13818-big subset of the training set (*small trains*). The total size of the FOF dataset is about 50G compressed by gzip to 5.4G, while for the TFF dataset it is about 90G, compressed by gzip to 7.7G. The complete datasets are publicly released at our accompanying repository.⁵

The translation of the Isabelle/HOL problems to TPTP does not preserve the names across the problems. The naming inconsistency can be as simple as the naturals being given the constant name `nat` or `nat2` in an encoded TPTP problem (this one happens because the projection `int-to-nat` is also called `nat` in Isabelle), depending on the order of defined constants in a given problem. Additionally, Isabelle mangles names as part of the encoding. For example in the basic theorem `List.distinct`, which states that an empty list is not equal to an applied list constructor, an instance of the empty list can look like `nil_Pr1308055047at_nat` for an empty list of products of pairs of naturals. This motivates our use of anonymous methods for ENIGMA and premise selection in this work (Section 4,5).

2.1 Differences to Related ITP/ATP Datasets

The FOF and TFF Isabelle exports we use are intended to be sound but generally sacrifice completeness to optimize ATP performance. The possible sources of incompleteness include:

- The heuristic premise filter [35] pre-selecting only a fixed number of premises that are generally not guaranteed to justify the conjecture in Isabelle.
- In the encodings, polymorphic types (such as `'a list`) are heuristically pre-instantiated (*monomorphized*) by ground types. This is an established optimization going back at least to Harrison's implementation of the MESON tactic [18] in HOL Light [17], which can be seen as a particular kind of an abstraction step when reasoning in large theories. Without a full abstraction-refinement loop [34], this is an obvious source of incompleteness, in a similar way as premise selection with a fixed premise limit.
- Limited treatment of higher-order constructs such as lambda abstraction, typically not fully encoded in the FOF and TFF problems. The encodings employ lambda-lifting, which is usually improving the ATP performance in practice, but is generally incomplete.

⁵ https://github.com/ai4reason/isa_enigma_paper

When developing new strategies, ATPs and premise selection methods, such optimizations may be premature, having different beneficial or adverse effects on the methods. In particular, in the experiments conducted by us, we detect small amount of incompleteness already with the baseline systems. For example, CVC5 reports 256 problems in the whole TFF dataset to be countersatisfiable. On the other hand, once a proof is found, it is typically comparatively easy to replay from the minimized set of premises by any ATP.

In this sense, the monomorphized Isabelle datasets considerably differ from other datasets used for large AI/TP experiments such as the toplevel theorems in the Mizar and HOL 4 libraries [9]. There, replaying the minimized proofs may still be quite hard for ATPs, and the exports are typically striving for completeness, fully delegating various abstraction-refinement methods such as monomorphization and premise selection to the AI/TP systems that may implement more complicated procedures for them.

We measure this in more detail by comparing the clasified premise-minimized ATP problems solved by Vampire and E on the Isabelle FOF dataset (88888 problems) and the Mizar dataset (113332 problems) using several metrics computed in Table 2. The table

■ **Table 2** Statistics of the Isabelle and Mizar clasified premise-minimized FOF problems solvable by E and Vampire. AC is the average number of clauses per problem, VC is the average number of clauses with variables per problem, EC is that for clauses with equality, iProver-10s is the number of problems solved by iProver limited to inst-gen calculus in 10s, and iProver-10s ratio is the ratio of that to the total number of problems.

Dataset	Problems	AC	VC	EC	iProver-10s	iProver-10s ratio
Isabelle FOF	88888	10.15	4.51	2.63	83015	0.93
Mizar	113332	35.55	23.16	10.31	65679	0.58
Ratio Miz/Isa		3.50	5.14	3.92		0.62

shows that the number of clauses per minimized problem is 3.5 times higher in Mizar. This may indicate the difference between the (generally harder) toplevel ITP problems and the intermediate goals. The most interesting difference is that about two thirds of the clauses in the Mizar problems contain variables, while in Isabelle this is only 44.4% of the clauses. Combined with the much higher number of clauses in the Mizar problems, this leads to 5.14 times more clauses with variables in the Mizar problems. For clauses with equality, this ratio is 3.92, i.e., also slightly higher than the ratio of the clauses. This means that the Isabelle problems are (after minimization) much more ground and non-equational, and thus likely much more amenable to instantiation-based methods than the Mizar problems. We confirm this by running iProver [31] on both sets of minimized problems using only its Inst-Gen calculus. In Mizar it solves 58% of the problems while in Isabelle 93%, i.e., 60% more.

3 Strategy Optimization for E and ENIGMA

ATP *strategies* play a critical role when proving theorems. Their targeted invention, optimization, and construction of their portfolios (*schedules*) may significantly improve the performance of the ATPs in different domains. We have also found that some ATP strategies behave better in combination with learning-based guidance of the ATPs than others, and that it often seems preferable to use a single strategy to produce the training data for ENIGMA.⁶

⁶ The use of single vs multiple strategies in combination with ENIGMA is not yet strongly experimentally explored. See, e.g., [15] for a recent related analysis.

Our initial goals are thus to (i) find a strong set of E strategies for the datasets, and in particular, to (ii) find a single strong E strategy that behaves well in combination with the ENIGMA guidance. We start exploring this on the FOF dataset, evaluating our 550 BliStr/Tune [23, 47] strategies previously invented on the Mizar, Sledgehammer, HOL, AIM and TPTP problems. This is done in two rounds. In the first round, we run all the 553 strategies on a smaller sample of 500 randomly selected FOF problems solvable by Vampire’s CASC mode in 30 seconds.⁷ After that, the 76 most performing and orthogonal strategies from the first run are evaluated on a bigger sample of 2000 Vampire-solvable problems. This yields the following top 2 strategies in the greedy cover:

```
protokoll_X---_auto_sine13 :995
protocol_eprøver_f171197f65f27d1ba69648a20c844832c84a5dd7 :198
```

The first strategy uses E’s auto-mode with a strong SInE filter, selecting up to 100 premises. Unlike in the Mizar problems, the `hypos` parameter of SInE is used here, giving the same importance to the local assumptions (TPTP role `hypothesis`) as to the conjecture. We have confirmed that this performs better than SInE without the parameter on the problems. This leads us to construct the ENIGMA features differently for Isabelle problems in Section 4.

The second strategy in the greedy cover (`f1711`) is the one working best in the Mizar/MPTP setting, where it also performs well when combined with the ENIGMA guidance. It is however significantly weaker (921 vs 995 solved problems) than the first auto-mode strategy. We conjecture that this is because it does not use SInE. Adding a strong SInE filter (with the “hypos” parameter) indeed improves its performance to 1022 problems, making it the strongest E strategy on the problems. Since it is also well behaved with the ENIGMA guidance, we use it in all further experiments. The base strategy (`f1711`) without any SInE filter will be denoted as $\mathcal{B}_{\text{base}}$, while the version with the SInE as $\mathcal{B}_{\text{sine}}$. With the clausification changes explained next we obtain two more strategies $\mathcal{B}_{\text{base3}}$ and $\mathcal{B}_{\text{sine3}}$.

3.1 Clausification

Clausification can have a large influence on the operation and performance of ATPs. In a setting with many complicated formulas, naive clausification can lead to exponential blow-ups. State-of-the-art ATPs counter that by introducing definitions for subformulas. E’s classifier uses heuristic counting of the occurrences of each subformula to decide when to introduce a new definition. The default factor (called `definitional-cnf`, `dc` for short) for this used by E has been experimentally optimized to be 24 many years ago on the TPTP benchmark. This may be however suboptimal for newer large-theory corpora, especially in encodings with type guards. Also, a possible explanation for the relatively large improvement of E by the aggressive SInE filter is that the clausification explodes quite frequently on the unfiltered problems. We investigate this in several ways.

First, we simply try to clausify all FOF problems with the default E options and a timeout of 60s. This results in a gzipped total size of 21G, i.e. four times the size of the gzipped FOF problems. This is however without 28212 (10% of all) problems that fail to get clausified within 60s. This is a lot, because ITP hammers typically give the ATPs a timeout of 15-30s to solve the whole problem.

⁷ We use here Vampire as a quick pre-filter for targeting the solvable problems by E strategies because in our preliminary experiments Vampire performed significantly better than E.

■ **Table 3** Influence of the dc values on the clausification timeouts and size of the clausal problems.

definitional-cnf (dc)	1	2	3	4	24
clausifications timed out in 60s (out of 1000)	0	0	0	51	125
gzipped size of all clausified problems (MB)	36	47	163	120	77

■ **Table 4** 15s $\mathcal{B}_{\text{base}}$ runs with/out SInE with different dc values on 1000 sample problems.

definitional-cnf (dc)	1	2	3	4	24
problems solved with SInE	242	268	271	266	263
problems solved without SInE	219	251	243	241	218

This leads us to an experiment with smaller values for the definitional-cnf (dc) parameter on a sample of 1000 training problems. We use a 60s timeout for the clausification, measure the total size of gzipped cnfs, and the number of files where the clausification timed out. The results are shown in Table 3. The dc value of 3 is the last one where there are no timeouts, but it already gives a 4-time blowup over $dc = 2$.

Both more aggressive premise selection and more aggressive introduction of new definitions can be used to counter the clausification blowup on the Isabelle problems. Since the SInE filter is only heuristic and usually inferior to trained premise selection, we prefer more aggressive use of new definitions. To measure how much the two methods interact, we evaluate our chosen strategy $\mathcal{B}_{\text{base}}$ with and without SInE and with various dc values in 15s on our sample of 1000 problems. The results are summarized in Table 4. They confirm that the two methods interact a lot. Setting $dc = 2$ replaces a lot of the improvement obtained by SInE with the default $dc = 24$. Since the SInE and non-SInE versions peak at $dc = 3$ and $dc = 2$ respectively, we experiment with these values of dc in our further experiments. We denote $\mathcal{B}_{\text{base3}}$ and $\mathcal{B}_{\text{SInE3}}$ the strategies obtained from $\mathcal{B}_{\text{base}}$ and $\mathcal{B}_{\text{SInE}}$ by setting $dc = 3$.

4 ENIGMA for Isabelle

State-of-the-art automated theorem provers (ATP), such as E, Prover9, and Vampire [32], are based on the saturation loop paradigm and the *given clause algorithm* [38]. The input problem, in first-order logic (FOF), is translated into a refutationally equivalent set of clauses, and a search for contradiction is initiated. The ATP maintains two sets of clauses: *processed* (initially empty) and *unprocessed* (initially the input clauses). At each iteration, one unprocessed clause is selected (*given*), and all of the possible inferences with all the processed clauses are generated (typically using resolution, paramodulation, etc.), extending the unprocessed clause set. The selected clause is then moved to the processed clause set. Hence the invariant holds that all inferences among processed clauses have been computed.

The selection of the “right” given clause is known to be vital for the success of the proof search. The first ENIGMA systems [14, 24–26] successfully implemented various ways of machine learning guidance for the clause selection based on gradient boosting decision trees (GBDT). Next generation ENIGMA [10, 22] abstracts from symbol names with anonymization methods and additionally employs graph neural network models (GNN) for clause selection. The latest ENIGMA [16] additionally implements clause filtering of generated clauses (*parental guidance*), and overcomes a slower speed of GNN models with amortizing evaluation server.

4.1 Model Training and Given Clause Guidance

The training of ENIGMA models is usually done in a training/evaluation loop. This general approach applies both to clause guidance and when filtering the generated clauses.

1. The training data \mathcal{T} are gathered from a number of previous successful proof searches. From each proof search, the training data consists of clauses processed during the proof search, labeled by flags *positive* or *negative* depending on whether they appear in the final proof. These labeled clauses are translated to a suitable format for the underlying selection model (vectors for GBDT models, and tensors for GNNs).
2. Based on data \mathcal{T} , a GBDT (or a GNN model) \mathcal{M} is trained. This model is capable of recognizing *positive* clauses from *negatives* by assigning a score to an arbitrary clause.
3. The model \mathcal{M} can be combined with an ordinary E's strategy \mathcal{S} in a *cooperative* way, yielding the ENIGMA strategy $\mathcal{S} \oplus \mathcal{M}$. The ENIGMA strategy $\mathcal{S} \oplus \mathcal{M}$ uses the model \mathcal{M} to guide the given clause selection inside E, and it inherits other behaviour from \mathcal{S} . In the cooperative setting, about 50% of the given clauses are selected as suggested by \mathcal{M} , while the remaining clauses are selected by the standard clause selection mechanism inherited from \mathcal{S} . Thus, ENIGMA compensates for a possible mistaken predictions of \mathcal{M} .
4. With new training data from new strategies, this process can be iterated.

4.2 Parental Guidance and Generated Clause Filtering

ENIGMA models are applied within E in two capacities: (1) given clause selection and (2) parental guidance for filtering of the generated clauses. Clausal parental guidance evaluates a new clause C based only on the features of the parents of C . Parental guidance thus serves as a fast rejection filter: generated clauses with scores below a chosen threshold are put into the *freezer* set and are only revived if E runs out of unprocessed clauses. Furthermore, such frozen clauses are never evaluated by other (possibly more expensive) heuristics. This mechanism thus effectively (and in a complete way) curbs the typically quadratic growth of the set of generated clauses. Full details can be found in previous work [16] where it was found that the parental guidance is most effective when the concatenated feature vectors of the parents are used as an input to the machine learning model. The data for training parental guidance is generated by classifying parents of proof clauses as *positive* and all other generated clauses during a proof search as *negative*. To balance the data, the ratio of negative to positive examples is a valuable hyperparameter.

4.3 Experiments with ENIGMA

ENIGMA was so far used only with first-order logic (FOF) data in the TPTP format. In this work, we extend the usability of ENIGMA models also to simply typed first-order formulas (TFF) of the TPTP format. In the case of GBDTs models, we simply forget the type annotations. Because GBDT ENIGMA models perform symbol name anonymization by replacing symbol names by their arities, all the simple type names would get translated to the same name anyway. In the case of GNN models, we embed the type information in the clause graphs by giving nodes representing variables of the same type by the same trained numerical representation (see Section 5).

ENIGMA models embed information about the conjecture being proved inside clause vectors/tensors. In this way, ENIGMA provides conjecture-specific suggestions. The conjectures are marked in the input format with the TPTP role `conjecture`. In these experiments, we additionally treat clauses with the TPTP role `hypothesis` just like conjectures. This helps to further differentiate among various Isabelle problems.

In this work, we use ENIGMA GBDT models for clause guidance inside E (for given clause selection and filtering of generated clauses), and we use the GNN models only for the task of premise selection. Section 5 describes how the GNN models are used for premise selection. The experimental evaluation described in Section 6 presents the results of training ENIGMA models for clause selection and parental guidance.

5 Premise Selection for Isabelle via Graph Neural Networks

A number of learning-based premise selection methods have been developed for large ITP corpora and hammers in the last two decades. See [2, 7, 33, 48] for their overviews. In a large evaluation done over the Mizar corpus,⁸ the strongest method turned out to be a property-invariant graph neural network (GNN) based on the architecture previously used in several settings [22, 37, 50]. We use this algorithm also for the Sledgehammer problems here.

GNNs, and in particular this architecture preserve several invariants of theorem proving data, such as insensitivity to clause ordering and literal ordering. The inference (decisions) about which premises are relevant for a conjecture are based on several rounds of neural message passing in a special graph constructed from the clauses corresponding to the formulas. The property invariant architecture also strives to be fully anonymous, in the sense that it is invariant to all symbol names: the representations of symbols are only based on their connectivity with other elements in the formula. It also has a specific encoding for argument order that allows the network to partially preserve this information and it has a special handling of negation: terms of opposite polarity are related by the corresponding operation $* - 1$ in the float based representation of the network.

This set of properties allows the architecture to perform well in various theorem-proving settings. On our Isabelle datasets, the symbol and name anonymity of the GNN is particularly important. As mentioned in Section 2, the symbol names and the formula names are not used consistently here, which would make the use of non-anonymous premise selection methods difficult. In this work, a 10-layer GNN was used. The sizes of the first layer embeddings were 4, 1, 4 for the *term*, *symbol* and *clause* nodes respectively. For the rest of the layers, the term, symbol and clause nodes were represented by vectors of size 32, 64 and 32 respectively. The last, non-message passing layer that has the task of predicting a probability for each premise had 128 neurons.

The GNN was newly modified to parse and make use of the typed TFF input. To take advantage of the type information, we train separate embeddings for all types (2539 in Section 6.4) that occur more than 10.000 times in the data. The GNN uses this type embedding when reading in a variable, and the type embedding can contain information about the type of the variable. Here, for simplicity, we chose to directly learn the embeddings (initial GNN values before the start of the message passing) for the typed variable nodes. This however does not fully preserve the anonymity of the symbols in the GNN, which is one the core design principles of this neural architecture. Adding instead an extra node in the GNN for each type would allow us to preserve the anonymity also for types. In this setting the GNN would learn to understand the types based only on their use in the current problem, possibly thus generalizing better. This approach is however more complicated than our current solution and is left as future work here.

The Isabelle problems are big and their classification by our GNN parser may result in graphs with many clauses, even when we heuristically pre-reduce the initial set of formulas proposed by the MePo filter. This poses problems with the GPU memory (32 GB on our machines) both during training the GNN and when using it for predicting the relevant clauses.

⁸ https://github.com/ai4reason/ATP_Proofs

To counter that, we have introduced several limits related to the number of nodes in the clause graphs that allow us to skip very large classified problems. The limit that we currently use skips any problem that contains more than 50000 term nodes after clausification (this corresponds roughly to the 95th percentile for the amount of term nodes in the problems).

6 Evaluation

We experiment with four variants of Isabelle problems. The first two are (1) FOF and (2) TFF without premise selection. Then there are two versions result from the GNN premise selector applied to the TFF data: (3) PRE_1 and (4) PRE_2 .

First, Section 6.1 describes experiments with given clause guidance, and Section 6.2 describes experiments with adding parental guidance. These two experiments were partially used to obtain the training data for premise selection described in Section 6.3 and Section 6.4.

6.1 Evaluation of ENIGMA Given Clause Guidance

We perform three separate evaluations of the GBDT (LightGBM [30]) ENIGMA clause selection models on three different presentations of Isabelle problems. (1) On the FOF translation (without premise selection) in Section 6.1.1, (2) on the TFF translation (without premise selection) in Section 6.1.2, and (3) on the TFF translation with GNN premise selection in Section 6.1.3. The second premise selection dataset PRE_2 is not used here.

We experiment with combining training samples from different strategies. Different E strategies might use different term orderings affecting the clause normalization. Since the ENIGMA models are syntax based, we only combine training samples from *compatible* strategies, which perform equivalent clause normalization. At this point, we consider strategies to be *compatible* when they use the same term ordering and literal selection function.

6.1.1 Experiment FOF: First-Order Translation

Setup. First, we experiment with the FOF translations of Isabelle problems without any premise selection method applied. E supports *sine* filters to reduce the number of axioms of large problems. Since the problems have no premise selection applied, we use two versions of the E strategy to obtain training problems: $\mathcal{B}_{\text{sine}}$ uses a manually selected sine filter⁹ and $\mathcal{B}_{\text{base}}$ does not use a sine filter. We perform three training/evaluation loops as follows.

1. *Initial training data* \mathcal{T}_0 : Evaluation of $\mathcal{B}_{\text{base}}$ and $\mathcal{B}_{\text{sine}}$ on the training problems.
2. Train the model \mathcal{L} on the current data \mathcal{T} .
3. Evaluate $\mathcal{B}_{\text{base}} \oplus \mathcal{L}$ and $\mathcal{B}_{\text{sine}} \oplus \mathcal{L}$ on the training problems.
4. Extend data \mathcal{T} and continue with step 2.

We combine the two base strategies with model \mathcal{L} in a cooperative way. With model \mathcal{L} we obtain two strategies with ENIGMA guidance, that is, $\mathcal{B}_{\text{base}} \oplus \mathcal{L}$ and $\mathcal{B}_{\text{sine}} \oplus \mathcal{L}$.

Learning Statistics. Table 5 presents training data statistics and models evaluation for the three training/evaluation loops performed in this FOF experiments. There is:

- **training:** The column *probs* is the number of training problems in the training data, while the column *proofs* is the number of different successful proof runs, where we can have multiple proofs for a single problem. The column *rows* signifies the number of vectors in the training data, each vector corresponding to one clause in the proofs. The column *filesize* is the file size of the *compressed* training samples.

⁹ `-sine='GSine(CountFormulas,hypos,1.1,03,20000,1.0)'`

■ **Table 5** Experiment FOF: Learning statistics (Section 6.1.1).

l	notation		training				accuracy[%]			model	
	$trains$	$model$	$probs$	$proofs$	$rows$	$filesize$	acc	pos	neg	$time$	$filesize$
0	\mathcal{T}_0^{FOF}	\mathcal{L}_0^{FOF}	70K	114K	8M	1.1G	92.8	89.8	93.4	0:12	54.8M
1	\mathcal{T}_1^{FOF}	\mathcal{L}_1^{FOF}	81K	255K	16M	2.3G	87.8	82.1	89.0	0:20	54.9M
2	\mathcal{T}_2^{FOF}	\mathcal{L}_2^{FOF}	84K	400K	23M	3.2G	85.6	81.9	86.5	0:31	55.1M

■ **Table 6** Experiment FOF: ATP performance (Section 6.1.1).

l	strategy		trains solved by				devels solved by			
	$base$	$sine$	$base$	$sine$	$both$	$total$	$base$	$sine$	$both$	$total$
-	\mathcal{S}_{base}	\mathcal{S}_{size}	56 921	65 124	75 080	75 080	3114	3567	4084	4084
0	$\mathcal{S}_* \oplus \mathcal{L}_0^{FOF}$		77 084	72 869	85 903	86 661	3888	3886	4552	4784
1	$\mathcal{S}_* \oplus \mathcal{L}_1^{FOF}$		80 613	74 191	87 734	89 886	3933	3851	4516	4947
2	$\mathcal{S}_* \oplus \mathcal{L}_2^{FOF}$		81 640	74 878	88 566	91 261	3963	3894	4558	5036

- **accuracy:** Columns acc , pos , neg show testing accuracies of each model on the testing set in percents. Column acc show the overall model accuracy, while columns pos and neg show testing accuracy on positive and negative testing samples separately.
- **model:** The column $time$ shows the time needed for model training (in hours and minutes), and the column $size$ shows the LightGBM model file size. Model file size is an important suggestion of the model ATP performance, since the model size influences the model loading time and prediction times in E.

When training a model, we set aside 5% of the training data in order to compute the testing **accuracy**. The model is trained on the remaining 95%.¹⁰ This split is done on the level of solved problem names rather than on proofs or vectors so that all the proofs of a single problem will appear either in the 95% training subset, or all in the 5% testing subset. This is important to keep the testing set unbiased. Otherwise, the testing data can partially overlap with the training data, since two proofs of the same problem tend to be quite similar. This split on solved problem names is computed independently in every loop iteration. This split is done only on the training problems of the global training/development/holdout split used for further experiment in this paper.

From the numbers in Table 5, we can see that the number of solved problems (column $probs$) in the data increases with every loop iteration but much more slowly than the value in the column $proofs$. This means that we are obtaining duplicate proofs for already solved problems, since we include all the proofs for all solved problems in the training data in this experiment. Note that the testing accuracies decrease with increasing training data size. All the models have been built in less than 30 minutes and result in a similarly sized model file. Also note that number of $proofs$ grows much faster than the problems solved ($probs$). It shows that we often prove the same problems.

¹⁰The numbers in the **training** columns are only on the training 95% subset.

■ **Table 7** Experiment TFF: Learning statistics (Section 6.1.2).

l	notation		training				accuracy[%]			model	
	<i>trains</i>	<i>model</i>	<i>probs</i>	<i>proofs</i>	<i>rows</i>	<i>size</i>	<i>acc</i>	<i>pos</i>	<i>neg</i>	<i>time</i>	<i>size</i>
0	$\mathcal{T}_0^{\text{TFF}}$	$\mathcal{L}_0^{\text{TFF}}$	108K	186K	10,3M	1.2G	89.6	86.2	90.2	12:36	54.8M
1	$\mathcal{T}_1^{\text{TFF}}$	$\mathcal{L}_1^{\text{TFF}}$	114K	383K	19,6M	2.2G	85.0	78.8	86.2	20:29	55.0M
2	$\mathcal{T}_2^{\text{TFF}}$	$\mathcal{L}_2^{\text{TFF}}$	117K	587K	27,9M	3.1G	82.6	77.4	83.8	20:52	55.1M
3	$\mathcal{T}_3^{\text{TFF}}$	$\mathcal{L}_3^{\text{TFF}}$	122K	822K	39,3M	4.3G	81.4	77.8	82.2	23:17	55.2M
4	$\mathcal{T}_4^{\text{TFF}}$	$\mathcal{L}_4^{\text{TFF}}$	123K	1.03M	48,6M	5.3G	80.9	77.6	81.7	29:46	55.3M

ATP Evaluation. Table 6 shows the ENIGMA models performance separately on training (**trains**) and on development problems (**devel**). Since the development problems were not used during the training in any way, this evaluation tells how much are the ENIGMA model over-fitting on the training files.

Every row describes the performance of two strategies specified in the column **strategy**. Problems solved by the two strategies individually are in the first two **bold** columns. *Italics* values display a total cover of set of strategies. The column *both* shows the number of problems solved both by the two strategies together. This is helpful to estimate the complementarity of *base* and *sine* strategies. Two strategies are *complementary*, when they solve different problems. The column *total* shows the cumulative number of problems solved by all the current strategies (above in the table).

In Table 6, we see that the *sine* strategy performed better than *base* initially. However, from the first learning the *base* strategy dominates. This suggests that ENIGMA learns to do premise selection on its own to some extent (when trained on the samples from the *sine* strategy). All *base* and *sine* strategies are, however, quite complementary. In total, we start with 75 080 solved problems and we end up with 91 261 after the learning, almost 22% improvement on trains (23% on devals). The best single strategy is improved by 25% on trains (and by 11% on devals).

It is interesting to observe, how the *base* strategies in one iteration improves on both *base* and *sine* from the previous iteration, as if merging the two strategies into one. It suggests that additional proof samples from compatible but complementary strategies could lead to an additional improvement. We further investigate this in the next experiment (Section 6.1.2).

6.1.2 Experiment TFF: Typed First-Order Formulae

Setup. We perform a similar experiment as for the FOF, but this time targeted to the TFF Isabelle translation.

1. Again, we start with the training data obtained by the evaluation of $\mathcal{B}_{\text{base}}$ and $\mathcal{B}_{\text{sine}}$.
2. We run three iterations of the training/evaluation loop.
3. After the three iterations, we additionally evaluate two more pure E strategies $\mathcal{B}_{\text{base3}}$ and $\mathcal{B}_{\text{sine3}}$ which improve on $\mathcal{B}_{\text{base}}$ and $\mathcal{B}_{\text{sine}}$ by adjusting E’s classification algorithm (switch E’s option “**definitional-cnf**” from 24 to 3).
4. We perform two more training/evaluation loops with the expanded training data.

■ **Table 8** Experiment TFF: ATP performance (Section 6.1.2).

l	strategy		trains solved by				devels solved by			
	<i>base</i>	<i>sine</i>	<i>base</i>	<i>sine</i>	<i>both</i>	<i>total</i>	<i>base</i>	<i>sine</i>	<i>both</i>	<i>total</i>
-	$\mathcal{S}_{\text{base}}$	$\mathcal{S}_{\text{size}}$	100 259	98 317	114 838	114 838	5532	5403	6347	6347
0	$\mathcal{S}_* \oplus \mathcal{L}_0^{\text{TFF}}$		108 377	101 271	118 262	121 353	5468	5347	6222	6692
1	$\mathcal{S}_* \oplus \mathcal{L}_1^{\text{TFF}}$		113 729	103 382	121 995	124 795	5788	5471	6466	6894
2	$\mathcal{S}_* \oplus \mathcal{L}_2^{\text{TFF}}$		115 790	104 270	123 400	126 344	5934	5505	6547	6894
*	$\mathcal{S}_{\text{base3}}$	$\mathcal{S}_{\text{sine3}}$	106 132	100 904	118 925	132 552	5881	5522	6515	7160
3	$\mathcal{S}_{*3} \oplus \mathcal{L}_3^{\text{TFF}}$		122 492	107 035	127 955	133 222	6293	5673	6785	7280
4	$\mathcal{S}_{*3} \oplus \mathcal{L}_4^{\text{TFF}}$		122 931	107 316	128 339	133 762	6277	5704	6812	7326

Learning Statistics. Table 7 presents the machine learning evaluation (in the same format as Table 5 described in Section 6.1.1). Before the fourth loop ($l = 3$), we additionally evaluate all the strategies $\mathcal{S} \oplus \mathcal{L}$, for \mathcal{S} ranging over $\mathcal{B}_{\text{base3}}$ and $\mathcal{B}_{\text{sine3}}$, and for \mathcal{L} ranging over the models of the first three loops. This gives us additional training data for the fourth iteration, reflected in the table by a sudden increase in both solved problems (*probs*) and *proof* count (in the row $l = 3$). We see similar training times and model sizes as in the FOF experiment.

ATP Evaluation. Table 8 presents the ATP evaluation (in the same format as Table 6 described in Section 6.1.1). As opposed to the FOF experiment, the *base* strategies dominate from the beginning. Both strategies are still highly complementary. The evaluation of $\mathcal{B}_{\text{base3}}$ and $\mathcal{B}_{\text{sine3}}$ strategies boosts the number of solved trains from 126 344 to 132 552. This highly improves the performance of the best strategy (*base*) in the fourth iteration ($l = 3$) from 115 790 to 122 492, that is, by 5.8%. It shows that additional external training data can be quite useful during the training. We further investigate this issue in the next experiment (Section 6.1.3).

6.1.3 Experiment PRE₁: First GNN Premise Selection

Setup. Here we experiment with GNN premise selection data PRE₁ obtained by applying GNN premise selection to the TFF problems. The GNN premise selection produces several collections of the training problems (called *slices*) with a slightly different clause selection criterion. We experiment with two slices PRE₁⁻¹ and PRE₁⁶⁴, which were experimentally found well performing and complementary. Our first experiment is aimed at generating a large collection of training samples.

1. We perform three loops of training/evaluation, just as in the TFF experiment, separately on PRE₁⁻¹ and PRE₁⁶⁴. We loop with the base strategies $\mathcal{B}_{\text{base3}}$ and $\mathcal{B}_{\text{sine3}}$.
2. We merge the training data from the previous two separate experiments and perform three more loops on the merged data. However, we drop the *sine* strategies and evaluate only the strategy $\mathcal{B}_{\text{base3}} \oplus \mathcal{L}$ on the two PRE₁ slices.
3. From the above we collect a large database of 108K proved training problems. Since the collection can contain duplicate proofs of a single problem, we select just three proofs per problem. We use the proof pos/neg ratios as a measure of proof similarity, and select proofs thusly different.
4. The training data from the last step, denoted $\mathcal{T}_{\text{three}}^{\text{PRE}_1}$, gives us one final model $\mathcal{L}_{\text{three}}^{\text{PRE}_1}$.

■ **Table 9** Experiment PRE_1 : Learning statistics (Section 6.1.3).

notation		training				accuracy[%]			model	
<i>trains</i>	<i>model</i>	<i>probs</i>	<i>proofs</i>	<i>rows</i>	<i>size</i>	<i>acc</i>	<i>pos</i>	<i>neg</i>	<i>time</i>	<i>size</i>
$\mathcal{T}_{\text{three}}^{\text{PRE}_1}$	$\mathcal{L}_{\text{three}}^{\text{PRE}_1}$	108K	186K	10,3M	1.2G	89.6	86.2	90.2	12:36	54.8M
$\mathcal{T}_{\text{six}}^{\text{PRE}_1}$	$\mathcal{L}_{\text{six}}^{\text{PRE}_1}$	133K	763K	28,6M	3.26G	77.6	76.8	78.0	25:44	77.9M

■ **Table 10** Experiment PRE_1 : ATP performance (Section 6.1.3).

<i>strategy</i>	trains solved by				devels solved by			
	PRE_1^{-1}	PRE_1^{64}	<i>both</i>	<i>total</i>	PRE_1^{-1}	PRE_1^{64}	<i>both</i>	<i>total</i>
$\mathcal{S} = \mathcal{S}_{\text{base3}}$	122 196	117 341	126 323	126 323	6706	6462	6955	6955
$\mathcal{S} \oplus \mathcal{L}_{\text{three}}^{\text{PRE}_1}$	127 606	120 248	129 971	132 431	6800	6495	6994	7251
$\mathcal{S} \oplus \mathcal{L}_{\text{six}}^{\text{PRE}_1}$	132 063	123 229	134 544	135 823	6994	6591	7153	7380

Next experiment tries to gather even more training samples.

1. We additionally consider training data from the previous TFF experiments.
2. We gather even more valuable training samples from ENIGMA parental guidance experiments on slices PRE_1^{-1} and PRE_1^{64} .
3. We select three proofs per problem from TFF samples.
4. We select three proofs per problem from PRE_1 samples.
5. The training data $\mathcal{T}_{\text{sixes}}^{\text{PRE}_1}$ contain six proofs per problem and yield model $\mathcal{L}_{\text{sixes}}^{\text{PRE}_1}$.

Learning Statistics. Table 9 presents the machine learning evaluation (in the same format as Table 5 described in Section 6.1.1). Note the huge difference in the number of *proofs*, resulting in much larger *training data size*. The second training data include proofs of more than 25K additional problems (*probs*). Training times and model sizes clearly reflect the training file size.

ATP Evaluation. Table 10 presents the ATP evaluation (in the same format as Table 6 described in Section 6.1.1). Here, however, we evaluate the single strategy $\mathcal{B}_{\text{base3}} \oplus \mathcal{L}$ on slices PRE_1^{-1} and PRE_1^{64} , instead of using two *base* and *sine* strategies.

Firstly, we note the effect of the premise selection itself. The performance on $\mathcal{B}_{\text{sine3}}$ improved by more than 15% from the previous experiment (from 106 132 to 122 196). The model $\mathcal{L}_{\text{three}}^{\text{PRE}_1}$ performs quite well, being trained on proofs 108K problems, it solves almost 128K problems. The model $\mathcal{L}_{\text{six}}^{\text{PRE}_1}$ further boosts the performance, showing that combining of training data from various compatible sources might be beneficial. Comparing the performance on trains with the performance on devels, we can conclude that ENIGMA LightGBM clause selection models slightly overfit but they are still capable of generalization.

6.2 Evaluation of the Parental Guidance

Setup. Parental guidance models are co-trained with clause selection models in a series of loops over the training data. In each loop iteration, the LightGBM parameters for parental guidance models are tuned using a series of grid searches with Optuna [1]. These are the

■ **Table 11** Parental guidance iterations on small trains, devel, and holdout (13 818 problems in 15s). Loops \mathcal{L}_1 and \mathcal{L}_2 are run on TFF data, \mathcal{L}_3 to \mathcal{L}_8 on PRE_1^{-1} , and \mathcal{L}_9 and \mathcal{L}_{10} on PRE_2^{-1} .

	\mathcal{L}_1	\mathcal{L}_2	\mathcal{L}_3	\mathcal{L}_4	\mathcal{L}_5	\mathcal{L}_6	\mathcal{L}_7	\mathcal{L}_8	\mathcal{L}_9	\mathcal{L}_{10}
small trains	6475	6718	7081	7140	7312	7351	7407	7417	7647	7705
devel	6241	6462	6928	6892	6566	6850	7070	7115	7277	7379
holdout	6251	6459	6886	6843	6581	6816	7015	7062	7352	7395

number of leaves, the bagging fraction and frequency, the minimum number of samples to create a new leaf, and L1 and L2 regularization. The learning rate is fixed at 0.15, the maximum tree depth is capped at 256 and the number of trees is 250. The number of leaves is varied between 256 and 3333. The best result of each grid search is used for the next parameter’s grid search. Accuracy on positive training examples is considered twice as important as the accuracy on negatives when choosing which parameters perform best. There are multiple reasons for this. A primary reason is that the confidence in positive examples is higher than confidence for the classification of negatives because a negative clause in one successful proof search could be positive in another proof search. The resulting model is evaluated with the nine parental filtering thresholds, $\{0.03, 0.05, 0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7\}$, over a set of 300 problems from the development set for 30 seconds. This is done for the vanilla TFF problems as well as the problems with premise selection slices. The best run (as evaluated by a greedy cover) on each version of the problems is then run on the full training set. Then the problems from runs in the total greedy cover are used as data for the next iteration of looping. This means that some problems can have over 10 proofs in the training data.

Iterations. The training of parental guidance was done with the aim to develop as strong a performance as possible, using diverse data. The models for loops \mathcal{L}_1 and \mathcal{L}_2 are run and trained on the TFF data that do not use premise selection. Models \mathcal{L}_3 and \mathcal{L}_4 are only run on the small trains set. The models \mathcal{L}_3 to \mathcal{L}_8 are run on PRE_1 . Finally, models \mathcal{L}_9 and \mathcal{L}_{10} are run on PRE_2^{-1} . The largest performance jumps correspond to the addition of premise selection (Table 11). The strongest parental guidance models are always on the PRE^{-1} premise selection data and the PRE_1^{64} slices provide fewer complementary problems than the baseline TFF problems.

The best model \mathcal{L}_{10} , with the second premise selection slices, PRE_2^{-1} , proves 168 problems (56%) in 30s on the parameter tuning development set, and 137 893 problems (55.4%) on the training set in 15s. In 30s, \mathcal{L}_{10} proves 7472 problems (54.1%) on the development set and 7466 problems (54%) on the holdout. Without premise selection, \mathcal{L}_{10} proves 133 390 training problems (53.6%), which indicates that training on the premise selection data transfers back to the original problems. The remaining results are presented in Table 11. This parental+ENIGMA model is our final product. It solves 7395 holdout problems in 15s, thus significantly improving over unguided E and also over all other ATPs and SMTs. It also outperforms all other ATPs and SMTs even when they use our best premises (Table 12).

6.3 First Training of Premise Selection on TFF Problems (PRE_1)

We have done several large experiments with the GNN-based premise selection (Section 5), first in the untyped and then in the typed setting. For lack of space we include below only the two final experiments on the TFF data, where most of the ENIGMA runs were done.

For the first round of training the GNN on the TFF data we are using the proof data produced only by the base sine/nosine TFF runs of unguided E and the first three ENIGMA iterations on the TFF training set. Altogether these runs produce 1701284 proof dependencies. These dependencies are first deduplicated to 353875, and then we also for every problem P remove all premise sets subsumed by a smaller premise set. This further decreases the size of the dataset to 242432 proof dependencies, for 131309 unique solved problems. Most of the solved problems (80993) have after this redundancy elimination only one solution, while for the remaining ones we get from 2 to 16 different solutions. Since problems with 1 to 3 solutions dominate the dataset (163650 out of the total 242432 solutions), we do not do further pruning of over-represented proofs for the training (as in the next training run).

For this first training and prediction we do not yet use the new typed extensions of the GNN. Instead, all TFF formulas are stripped of their type information and given to the network as untyped FOF. Each problem uses its original conjecture, the positives are the premises used in a given proof and the negatives are all other premises for that problem (i.e., all the MePo premises). This sometimes leads to large training inputs, so we normalize them to have size at most 500KB by randomly removing negatives. The whole training dataset has size 46GB. We then train the GNN on it with batch size 10, learning rate 0.005, and with balancing the loss on the positive and negative premises.

The training for two full epochs on an NVIDIA Volta 100 takes about 12 hours, saving the weights 16 times. The balanced accuracy increases from 0.8533 to the final 0.9067 (0.9061 vs 0.9073 on positives vs negatives) in our final snapshot, which we then use for prediction over all 276363 problems. This is parallelized over four GPUs, taking several hours. For each problem we use the predictions to produce 5 premise selections based on the GNN score threshold (1,0,-1,-2,-3,-4), and 5 premise selections based on old-style top slices of the ranked premises (16,32,64,128,256). We do a small search with 200 development problems and the base strategy over this grid, which is won by the -1-based predictions, best complemented by the 64-based predictions. These premise selections are denoted PRE_1^{-1} and PRE_1^{64} in the other parts of this paper. E/ENIGMA are then evaluated on both of them, while we also evaluate other systems only on the -1-based predictions (Table 12).

6.4 Second Training of Premise Selection on TFF Problems (PRE_2)

The second premise selection training is done by the typed version of the GNN (Section 5), using explicitly 2539 types that occur with frequency higher than 10000 in the training data. The remaining types (over 300000 in the training set) are mapped to the same generic embedding, which means that the GNN treats them all as the same type. The overhead for the 2539 distinguished most frequent types increases the size of the GNN only by 100kb. The training uses again a batch size of 20 and a learning rate of 0.0005. The training dataset is created from all TFF training problems solved in the previous loops, both by E/ENIGMA and CVC5 and Vampire. This gives 823141 unique premise selections for 146576 solved problems. The 823141 unique premise selections are again minimized with respect to subsumption, reducing them to 488186 minimal premise selections. To address the imbalance caused by having various numbers of proofs for a single problem in the training set, we keep at most three proofs for each problem. This further reduces the set to 292080 examples. The examples are again all reduced to a size of at most 500KB.

■ **Table 12** Final comparison with non-ENIGMA systems: E2.6 with its auto-schedule, CVC5, and Vampire-CASC (master 4909). Each run standalone (MePo predictions) and with the first/second -1 GNN TFF predictions. The last entry is the final/best $Loop_{10}$ (parental) ENIGMA (Section 6.2).

method	E auto-sched.	CVC5	Vampire	\mathcal{L}_{10} ENIGMA
15s devel, no premsel	5891	7053	6452	7133
15s holdout, no premsel	5903	7051	6454	7139
30s holdout, no premsel	6089	7140	6945	7170
15s devel, preds -1 (1st round)	6968	7211	7023	7191
15s holdout, preds -1 (1st round)	6956	7158	6978	7155
15s devel, preds -1 (2nd round)	7074	7394	7132	7379
15s holdout, preds -1 (2nd round)	7066	7372	7118	7395
30s holdout, preds -1 (2nd round)	7139	7398	7397	7466

This results in our final training set with an overall size of about 60GB. Since the reduction of the TFF inputs does not generally guarantee to prevent a blow-up during the clausification, we also further use here a size limit of 50000 nodes inside the GNN parser (Section 5) and filter out such large graphs which may otherwise deplete the GPU memory. The GNN is trained for full two epochs on the data, taking about one day on a single NVIDIA V100 GPU and storing the weight files 15 times per epoch. For producing the final predictions, we take the 28th weights with the highest balanced accuracy of 0.9221 (0.9391 / 0.9051 for positives/negatives). We produce the same grid of predictions as in the first round for all problems. The -1-based predictions are again the winner, best complemented by the 0-based predictions. These premise selections are denoted PRE_2^{-1} and PRE_2^0 in the other parts of this paper. Table 12 shows that also all non-ENIGMA systems benefit from the GNN predictions, and that the second round improves over the first round of predictions for all of them.

7 Conclusion

We have developed versions of the ENIGMA systems and neural premise selectors for the Isabelle Sledgehammer problems. Our best single-strategy system using the parental ENIGMA guidance and the typed GNN premise selection solves 7395 holdout problems in 15s, improving on original E’s auto-schedule performance (5903) by 25.3%. It also improves on all other ATPs and SMTs, both when used standalone and when used in conjunction with our best neural premise selection. To achieve this, we have produced large corpora of Isabelle problems for training and evaluation of the AI/TP methods, and developed new extensions of our systems, especially for the typed setting.

References

- 1 Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.
- 2 Jesse Alama, Tom Heskens, Daniel Kühlwein, Evgeni Tsivtsivadze, and Josef Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014. doi:10.1007/s10817-013-9286-5.
- 3 Alexander A. Alemi, François Chollet, Niklas Eén, Geoffrey Irving, Christian Szegedy, and Josef Urban. DeepMath - deep sequence models for premise selection. In Daniel D. Lee, Masashi Sugiyama, Ulrike V. Luxburg, Isabelle Guyon, and Roman Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2235–2243, 2016. URL: <http://papers.nips.cc/paper/6280-deepmath-deep-sequence-models-for-premise-selection>.

- 4 Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. The tactician - A seamless, interactive tactic learner and prover for coq. In *CICM*, volume 12236 of *Lecture Notes in Computer Science*, pages 271–277. Springer, 2020.
- 5 Jasmin Christian Blanchette, Sascha Böhme, Andrei Popescu, and Nicholas Smallbone. Encoding monomorphic and polymorphic types. In Nir Piterman and Scott A. Smolka, editors, *TACAS*, volume 7795 of *LNCS*, pages 493–507. Springer, 2013. doi:10.1007/978-3-642-36742-7_34.
- 6 Jasmin Christian Blanchette, Max W. Haslbeck, Daniel Matichuk, and Tobias Nipkow. Mining the archive of formal proofs. In Manfred Kerber, Jacques Carette, Cezary Kaliszyk, Florian Rabe, and Volker Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2015. doi:10.1007/978-3-319-20615-8_1.
- 7 Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016. doi:10.6092/issn.1972-5787/4593.
- 8 Sascha Böhme and Tobias Nipkow. Sledgehammer: Judgement Day. In Jürgen Giesl and Reiner Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010. doi:10.1007/978-3-642-14203-1_9.
- 9 Chad E. Brown, Thibault Gauthier, Cezary Kaliszyk, Geoff Sutcliffe, and Josef Urban. GRUNGE: A grand unified ATP challenge. In *CADE*, volume 11716 of *Lecture Notes in Computer Science*, pages 123–141. Springer, 2019.
- 10 Karel Chvalovský, Jan Jakubův, Martin Suda, and Josef Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In 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*, pages 197–215. Springer, 2019. doi:10.1007/978-3-030-29436-6_12.
- 11 Martin Desharnais, Petar Vukmirović, Jasmin Blanchette, and Makarius Wenzel. Seventeen provers under the hammer, 2022. URL: <https://matryoshka-project.github.io/pubs/seventeen.pdf>.
- 12 Michael Färber and Cezary Kaliszyk. Random forests for premise selection. In Carsten Lutz and Silvio Ranise, editors, *Frontiers of Combining Systems - 10th International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings*, volume 9322 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2015. doi:10.1007/978-3-319-24246-0_20.
- 13 Thibault Gauthier, Cezary Kaliszyk, Josef Urban, Ramana Kumar, and Michael Norrish. Tactictoe: Learning to prove with tactics. *J. Autom. Reason.*, 65(2):257–286, 2021.
- 14 Zarathustra Goertzel, Jan Jakubův, and Josef Urban. Enigmawatch: Proofwatch meets ENIGMA. In Serenella Cerrito and Andrei Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2019. doi:10.1007/978-3-030-29026-9_21.
- 15 Zarathustra Amadeus Goertzel. Make E smart again (short paper). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 408–415. Springer, 2020.
- 16 Zarathustra Amadeus Goertzel, Karel Chvalovský, Jan Jakubův, Miroslav Olsák, and Josef Urban. Fast and slow enigmas and parental guidance. 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 173–191. Springer, 2021. doi:10.1007/978-3-030-86205-3_10.
- 17 John Harrison. HOL Light: A tutorial introduction. In Mandayam K. Srivas and Albert John Camilleri, editors, *FMCAD*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996. doi:10.1007/BFb0031814.

- 18 John Harrison. Optimizing Proof Search in Model Elimination. In M. McRobbie and J.K. Slaney, editors, *Proceedings of the 13th International Conference on Automated Deduction*, number 1104 in LNAI, pages 313–327. Springer, 1996.
- 19 Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCIS*, pages 299–314. Springer, 2011. doi:10.1007/978-3-642-22438-6_23.
- 20 Edvard K. Holden and Konstantin Korovin. Heterogeneous heuristic optimisation and scheduling for first-order theorem proving. In *CICM*, volume 12833 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 2021.
- 21 R. J. M. Hughes. Super-combinators a new implementation method for applicative languages. In *Proceedings of the 1982 ACM Symposium on LISP and Functional Programming, LFP '82*, pages 1–10, New York, NY, USA, 1982. Association for Computing Machinery. doi:10.1145/800068.802129.
- 22 Jan Jakubův, 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. doi:10.1007/978-3-030-51054-1_29.
- 23 Jan Jakubův and Josef Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Yves Bertot and Viktor Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017. doi:10.1145/3018610.3018619.
- 24 Jan Jakubův 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. doi:10.1007/978-3-319-62075-6_20.
- 25 Jan Jakubův and Josef Urban. Enhancing ENIGMA given clause guidance. In Florian Rabe, William M. Farmer, Grant O. Passmore, and Abdou Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018. doi:10.1007/978-3-319-96812-4_11.
- 26 Jan Jakubův and Josef Urban. Hammering MizAR by learning clause guidance. 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. doi:10.4230/LIPICs.ITP.2019.34.
- 27 Cezary Kaliszyk and Josef Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In Martin Davis, Ansgar Fehnker, Annabelle McIver, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 88–96. Springer, 2015. doi:10.1007/978-3-662-48899-7_7.
- 28 Cezary Kaliszyk and Josef Urban. MizAR 40 for MizAR 40. *J. Autom. Reasoning*, 55(3):245–256, 2015. doi:10.1007/s10817-015-9330-8.
- 29 Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 8836–8847, 2018. URL: <http://papers.nips.cc/paper/8098-reinforcement-learning-of-theorem-proving>.

- 30 Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, pages 3146–3154, 2017.
- 31 Konstantin Korovin. iprover - an instantiation-based theorem prover for first-order logic (system description). In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008. doi:10.1007/978-3-540-71070-7_24.
- 32 Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013. doi:10.1007/978-3-642-39799-8_1.
- 33 Daniel Kühlwein, Twan van Laarhoven, Evgeni Tsivtsivadze, Josef Urban, and Tom Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In Bernhard Gramlich, Dale Miller, and Uli Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 378–392. Springer, 2012. doi:10.1007/978-3-642-31365-3_30.
- 34 Julio César López-Hernández and Konstantin Korovin. An abstraction-refinement framework for reasoning with large theories. In *IJCAR*, volume 10900 of *Lecture Notes in Computer Science*, pages 663–679. Springer, 2018.
- 35 Jia Meng and Lawrence C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009. doi:10.1016/j.jal.2007.07.004.
- 36 Yutaka Nagashima and Ramana Kumar. A proof strategy language and proof script generation for isabelle/hol. In *CADE*, volume 10395 of *Lecture Notes in Computer Science*, pages 528–545. Springer, 2017.
- 37 Miroslav Olsák, Cezary Kaliszzyk, and Josef Urban. Property invariant embedding for automated reasoning. In Giuseppe De Giacomo, Alejandro Catalá, Bistra Dilkina, Michela Milano, Senén Barro, Alberto Bugarín, and Jérôme Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020. doi:10.3233/FAIA200244.
- 38 Ross A. Overbeek. A new class of automated theorem-proving algorithms. *J. ACM*, 21(2):191–200, April 1974. doi:10.1145/321812.321814.
- 39 Bartosz Piotrowski and Josef Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In Didier Galmiche, Stephan Schulz, and Roberto Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 2018. doi:10.1007/978-3-319-94205-6_37.
- 40 Bartosz Piotrowski and Josef Urban. Stateful premise selection by recurrent neural networks. In *LPAR*, volume 73 of *EPiC Series in Computing*, pages 409–422. EasyChair, 2020.
- 41 Michael Rawson and Giles Reger. lazycop: Lazy paramodulation meets neurally guided search. In *TABLEAUX*, volume 12842 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2021.
- 42 Simon Schäfer and Stephan Schulz. Breeding theorem proving heuristics with genetic algorithms. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*, pages 263–274. EasyChair, 2015. URL: http://www.easychair.org/publications/paper/Breeding_Theorem_Proving_Heuristics_with_Genetic_Algorithms, doi:10.29007/gms9.
- 43 Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013. doi:10.1007/978-3-642-45221-5_49.

- 44 Stephan Schulz, Simon Cruanes, and Petar Vukmirovic. Faster, higher, stronger: E 2.3. In 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*, pages 495–507. Springer, 2019. doi:10.1007/978-3-030-29436-6_29.
- 45 René Thiemann and Christian Sternagel. Certification of termination proofs using ceta. In Stefan Berghofer, Tobias Nipkow, Christian Urban, and Makarius Wenzel, editors, *TPHOLS*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009. doi:10.1007/978-3-642-03359-9.
- 46 Josef Urban. MPTP - Motivation, Implementation, First Experiments. *J. Autom. Reasoning*, 33(3-4):319–339, 2004. doi:10.1007/s10817-004-6245-1.
- 47 Josef Urban. BliStr: The Blind Strategymaker. In Georg Gottlob, Geoff Sutcliffe, and Andrei Voronkov, editors, *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*, pages 312–319. EasyChair, 2015. URL: http://www.easychair.org/publications/paper/BliStr_The_Blind_Strategymaker, doi:10.29007/8n7m.
- 48 Josef Urban. ERC project AI4Reason final scientific report, 2021. URL: http://grid01.cii.rc.cvut.cz/~mptp/ai4reason/PR_CORE_SCIENTIFIC_4.pdf.
- 49 Robert Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reasoning*, 16(3):223–239, 1996. doi:10.1007/BF00252178.
- 50 Zsolt Zombori, Josef Urban, and Miroslav Olsák. The role of entropy in guiding a connection prover. In *TABLEAUX*, volume 12842 of *Lecture Notes in Computer Science*, pages 218–235. Springer, 2021.

6 Conclusion and Contributions

6.1 Contributions

The contributions of each paper are listed here. I also discuss my share of co-authorship. Josef Urban and Jan Jakubův read the thesis and offered many helpful suggestions.

- “ProofWatch” (Section 5.1) is done in collaboration with Jan Jakubův, Josef Urban, and Stephan Schulz. Schulz implemented the core watchlist functionality in E. Jakubův did most of the work to implement the *dynamic watchlist* features. Urban ran the k-nearest neighbor algorithm to suggest clauses and proofs for the watchlists. I implemented an option to modify watchlist subsumption checks in E to treat Skolem symbols of equal arity as equal and a feature that overrides all priority functions with **PreferWatchlist** when a watchlist clause is subsumed. I did almost all experimental evaluations. (Estimated personal contribution: 35%)
- “ProofWatch Meets ENIGMA: First Experiments” (Section 5.2) is done in collaboration with Jan Jakubův and Josef Urban. Jakubův implemented 80-95% of the code to pass the proof progress vectors to the ENIGMA machine learning model. I helped to fine-tune the implementation and scripts. I ran almost all of the experimental evaluations. (Contributions: 45%)
- “ENIGMAWatch: ProofWatch Meets ENIGMA” (Section 5.2) is done in collaboration with Jan Jakubův and Josef Urban. Jakubův implemented multi-index subsumption indexing and feature hashing to allow ENIGMAWatch to scale to large ITP libraries, such as the full Mizar Mathematical Library. I evaluated four methods of selecting reduced-dimensionality proof-state vectors to use for guidance. I ran almost all of the experiments. (Contributions: 50%)
- “Make E Smart Again” (Section 5.4) is done almost entirely by me. Jan Jakubův and Josef Urban proposed the topic. Jakubův and Stephan Schulz discussed options for implementing a minimal term ordering within E. (Contributions: 95%)
- “Fast and Slow Enigmas and Parental Guidance” (Section 5.5) is done in collaboration with Karel Chvalovský, Jan Jakubův, Miroslav Olšák, and Josef Urban. Olšák developed the graph-neural network (GNN). Jakubův integrated the GNN into E and ENIGMA. Chvalovský implemented the GPU server-based evaluation to speed up neural guidance. Jakubův implemented the 2-phase ENIGMA. I implemented the Parental Guidance feature in E. Jakubův and Urban ran the experiments on the GPU server evaluation and on the 2-phase ENIGMA. I ran the experiments on Parental Guidance, and Urban ran the 3-phase ENIGMA evaluation. (Contributions: 40%)

- “The Isabelle ENIGMA” (Section 5.6) is done in collaboration with Jan Jakubův, Cezary Kaliszyk, Miroslav Olšák, Jelle Piepenbrock, and Josef Urban. Kaliszyk extracted the training problems from Isabelle sessions and translated them to appropriate TPTP formats. Jakubův and Urban optimized the E strategies for the Isabelle Sledgehammer problems. Jakubův and I modified the ENIGMA feature scripts to ignore types so that the TFF format could be used. Olšák and Piepenbrock adapted the GNN to work with the typed data for premise selection, and Urban did the premise selection. Jakubův ran the ENIGMA evaluations, and I ran the Parental Guidance evaluations. (Contributions: 30%)

6.2 Concluding Remarks

This thesis covers the development of three learning-based ATP proof search guidance methods. ProofWatch, ENIGMAWatch, and Parental Guidance all improve the state-of-the-art on datasets from the Mizar Mathematical Library: ProofWatch improves the single-strategy performance by 26.5% and five-strategy ensemble performance by 7%, ENIGMAWatch accelerates training and in the first iteration improved upon ENIGMA by 8.8%, Parental Guidance as part of the 3-phase ENIGMA culminates in a 60% improvement over E’s auto-schedule and 17.4% over the best previous result by ENIGMA. The fundamental investigations with E0 in the “Make E Smart Again” paper indicate that learning can replace even more of E’s search algorithms; however, this potential will require future research to be realized. In “The Isabelle ENIGMA”, we demonstrate the transferability of our results from the Mizar Mathematical Library to the Isabelle Sledgehammer problems, attaining competitive performance that slightly surpasses two of the strongest state-of-the-art ATPs: Vampire and CVC5.

The methods all focus on using additional semantic information and integrating learning models. The proof vectors in ProofWatch contain logical semantic information about which proofs the current proof state resembles. ENIGMAWatch demonstrates performance benefits in integrating semantic learning with statistical machine learning models. Parental Guidance includes the parent clause features as new information, and with the 2 and 3-phase ENIGMAs, demonstrates further the value of integrating multiple machine learning models and methods. The graph neural network adds more semantic representations of mathematical formulas (than the hashed ENIGMA features) and can even include (part of) a proof state as context clauses. In “The Isabelle ENIGMA” paper, iterating learning for proof search guidance and premise selection is also seen to help. The anecdotally observed faster learning of ENIGMA *coop* methods where the machine-learned guidance shares the role with automatically developed strategies [132, 239] is one more point in favor of combining many learning methods in one automated theorem proving system. The Make E Smart Again experiments with E0 suggest that machine learning methods can cope without simplification orderings; however, a more effective approach might be to learn the precedences for the Knuth-Bendix ordering on a problem-specific basis [25, 26].

The road forward is clear: find methods to incorporate additional information into the theorem proving loop and integrate these models with the rest. The field is still full of promising research avenues that have never before been made to work.

A Author Publications

Conference articles

1. Z. Goertzel, J. Jakubův, S. Schulz, and J. Urban. ProofWatch: Watchlist guidance for large theories in E. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018
2. Z. Goertzel, J. Jakubův, and J. Urban. ENIGMAWatch: ProofWatch meets ENIGMA. In S. Cerrito and A. Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2019
3. Z. A. Goertzel, K. Chvalovský, J. Jakubův, M. Olsák, and J. Urban. Fast and slow Enigmas and Parental Guidance. In B. Konev and G. 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 173–191. Springer, 2021
4. Z. A. Goertzel, J. Jakubův, C. Kaliszyk, M. Olsák, J. Piepenbrock, and J. Urban. The Isabelle ENIGMA. In J. Andronick and L. de Moura, editors, *13th International Conference on Interactive Theorem Proving (ITP 2022)*, volume 237 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:21, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik

Short conference articles

1. Z. Goertzel, J. Jakubův, and J. Urban. ProofWatch meets ENIGMA: First experiments. In G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, and M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 15–22. EasyChair, 2018
2. Z. A. Goertzel. Make E Smart Again (short paper). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 408–415. Springer, 2020

Extended abstracts

1. Z. Goertzel, J. Jakubův, and J. Urban. First experiments with watchlist guidance on Mizar. *AITP 2018*, 2018
2. Z. Goertzel and J. Urban. Usefulness of lemmas via graph neural networks. *AITP 2019*, 2019

3. Z. A. Goertzel. Make E Smart Again. *AITP 2020*, 2020
4. Z. A. Goertzel, J. Jakubuv, and J. Urban. Parental guidance in E. *AITP 2021*, 2021
5. Z. A. Goertzel, A. Pease, and J. Urban. Project proposal: Formal ethics ontology in SUMO. *AITP 2022*, 2022

Unrelated publications

1. Z. Goertzel. Offer networks simulation and dynamics. Master's thesis, University of Copenhagen, Computer Science Dept., 2017
2. B. Goertzel, T. Goertzel, and Z. Goertzel. The global brain and the emerging economy of abundance: Mutualism, open collaboration, exchange networks and the automated commons. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 114:65–73, JAN 2017

A.1 Citations of Author’s Publications

The citations of the author’s work are extracted from the Web of Science and are supplemented with interesting citations from Google Scholar. Where there are no citations on Web of Science, preprints are included instead. First-order self-citations are excluded. The data were gathered on April 28th, 2023.

Z. Goertzel, J. Jakubův, S. Schulz, and J. Urban. ProofWatch: Watchlist guidance for large theories in E. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018

- S. Schulz, S. Cruanes, and P. Vukmirovic. Faster, higher, stronger: E 2.3. In P. 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*, pages 495–507. Springer, 2019
- K. Chvalovský, J. Jakubův, M. Suda, and J. Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In P. 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*, pages 197–215. Springer, 2019
- M. S. Nawaz, M. Sun, and P. Fournier-Viger. Proof guidance in PVS with sequential pattern mining. In H. Hojjat and M. Massink, editors, *Fundamentals of Software Engineering*, pages 45–60, Cham, 2019. Springer International Publishing
- C. Ruhdorfer and S. Schulz. Efficient implementation of large-scale watchlists. In P. Fontaine, K. Korovin, I. S. Kotsireas, P. Rümmer, and S. Tournet, editors, *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual)*, volume 2752 of *CEUR Workshop Proceedings*, pages 120–133. CEUR-WS.org, 2020
- P. Roshon and F.-J. Yang. A study on deep learning approach to optimize solving construction problems. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 170–174, 2021

Z. Goertzel, J. Jakubův, and J. Urban. ProofWatch meets ENIGMA: First experiments. In G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, and M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 15–22. EasyChair, 2018

- K. Chvalovský, J. Jakubův, M. Suda, and J. Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In P. 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*, pages 197–215. Springer, 2019
- I. Abdelaziz, V. Thost, M. Crouse, and A. Fokoue. An experimental study of formula embeddings for automated theorem proving in first-order logic. *ArXiv*, abs/2002.00423, 2020

Z. Goertzel, J. Jakubův, and J. Urban. ENIGMAWatch: ProofWatch meets ENIGMA. In S. Cerrito and A. Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2019

- C. Ruhdorfer and S. Schulz. Efficient implementation of large-scale watchlists. In P. Fontaine, K. Korovin, I. S. Kotsireas, P. Rümmer, and S. Tourret, editors, *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual)*, volume 2752 of *CEUR Workshop Proceedings*, pages 120–133. CEUR-WS.org, 2020
- J. Jakubův, K. Chvalovský, M. Olsák, B. Piotrowski, M. Suda, and J. Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In N. Peltier and V. 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
- Z. Zombori, J. Urban, and C. E. Brown. Prolog technology reinforcement learning prover. In N. Peltier and V. Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 489–507, Cham, 2020. Springer International Publishing

- P. Roshon and F.-J. Yang. A study on deep learning approach to optimize solving construction problems. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 170–174, 2021
- M. Rawson. *Applications of machine learning to automated reasoning*. PhD thesis, University of Manchester, 2021
- D. El Ouraoui. *Méthodes pour le raisonnement d’ordre supérieur dans SMT*. PhD thesis, Université de Lorraine, 2021

Z. A. Goertzel. Make E Smart Again (short paper). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 408–415. Springer, 2020

- M. Rawson. *Applications of machine learning to automated reasoning*. PhD thesis, University of Manchester, 2021
- M. Rawson and G. Reger. Automated theorem proving, fast and slow. EasyChair Preprint no. 4433, EasyChair, 2021
- E. Aygün, L. Orseau, A. Anand, X. Glorot, V. Firoiu, L. M. Zhang, D. Precup, and S. Mourad. Proving theorems using incremental learning and hindsight experience replay. *CoRR*, abs/2112.10664, 2021

Z. A. Goertzel, K. Chvalovský, J. Jakubuv, M. Olsák, and J. Urban. Fast and slow Enigmas and Parental Guidance. In B. Konev and G. 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 173–191. Springer, 2021

- A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022
- E. K. Holden and K. Korovin. Graph sequence learning for premise selection. *arXiv preprint arXiv:2303.15642*, 2023

Z. Goertzel and J. Urban. Usefulness of lemmas via graph neural networks. *AITP 2019*, 2019

- W. Li, L. Yu, Y. Wu, and L. C. Paulson. Isarstep: a benchmark for high-level mathematical reasoning. In *International Conference on Learning Representations*, 2021

Z. Goertzel. Offer networks simulation and dynamics. Master's thesis, University of Copenhagen, Computer Science Dept., 2017

- B. Goertzel, S. Giacomelli, D. Hanson, C. Pennachin, and M. Argentieri. Singularitynet: A decentralized, open market and inter-network for ais. *Thoughts, Theories Stud. Artif. Intell. Res.*, 2017
- G. A. Montes and B. Goertzel. Distributed, decentralized, and democratized artificial intelligence. *Technological Forecasting and Social Change*, 141:354–358, 2019

B. Goertzel, T. Goertzel, and Z. Goertzel. The global brain and the emerging economy of abundance: Mutualism, open collaboration, exchange networks and the automated commons. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 114:65–73, JAN 2017

- Y. K. Dwivedi, A. Sharma, N. P. Rana, M. Giannakis, P. Goel, and V. Dutot. Evolution of artificial intelligence research in technological forecasting and social change: Research topics, trends, and future directions. *Technological Forecasting and Social Change*, 192:122579, 2023
- Y. Qin, Z. Xu, X. Wang, and M. Skare. Artificial intelligence and economic development: An evolutionary investigation and systematic review. *JOURNAL OF THE KNOWLEDGE ECONOMY*
- S. Abbate, P. Centobelli, R. Cerchione, E. Oropallo, and E. Riccio. Kick-start your scientific journey into the metaverse. *KNOWLEDGE MANAGEMENT & E-LEARNING-AN INTERNATIONAL JOURNAL*, 15(1, SI):103–114, MAR 2023
- A. Melkonyan, T. Gruchmann, F. Lohmar, and R. Bleischwitz. Decision support for sustainable urban mobility: A case study of the rhine-ruhr area. *SUSTAINABLE CITIES AND SOCIETY*, 80, MAY 2022
- S. Faustino. Deleuze in the wild: making philosophy matter for fintech. *JOURNAL OF CULTURAL ECONOMY*, 15(1, SI):93–102, JAN 2 2022
- P. C. Bhatt, V. Kumar, T.-C. Lu, and T. Daim. Technology convergence assessment: Case of blockchain within the ir 4.0 platform. *TECHNOLOGY IN SOCIETY*, 67, NOV 2021
- P. Centobelli, R. Cerchione, E. Esposito, and E. Oropallo. Surfing blockchain wave, or drowning? shaping the future of distributed ledgers and decentralized technologies. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 165, APR 2021

- L. Ante. Smart contracts on the blockchain - a bibliometric analysis and review. *TELEMATICS AND INFORMATICS*, 57, MAR 2021
- D. Shin and W. T. Bianco. In blockchain we trust: Does blockchain itself generate trust? *SOCIAL SCIENCE QUARTERLY*, 101(7):2522–2538, DEC 2020
- R. Alkhudary, X. Brusset, and P. Fenies. Blockchain in general management and economics: a systematic literature review. *EUROPEAN BUSINESS REVIEW*, 32(4):765–783, AUG 9 2020
- D. Shin and Y. Hwang. The effects of security and traceability of blockchain on digital affordance. *ONLINE INFORMATION REVIEW*, 44(4):913–932, AUG 10 2020
- A. Klarin. The decade-long cryptocurrencies and the blockchain rollercoaster: Mapping the intellectual structure and charting future directions. *RESEARCH IN INTERNATIONAL BUSINESS AND FINANCE*, 51, JAN 2020
- C. Last. *Global Brain: Foundations of a Distributed Singularity*, pages 363–375. Springer International Publishing, Cham, 2020
- X. Shi, Q. Zhang, and Z. Zheng. The double-edged sword of external search in collaboration networks: embeddedness in knowledge networks as moderators. *JOURNAL OF KNOWLEDGE MANAGEMENT*, 23(10):2135–2160, DEC 9 2019
- M. J. Meirino, M. P. Mexas, A. d. V. Faria, R. P. Mexas, and G. D. Meirelles. Blockchain technology applications: A literature review. *BRAZILIAN JOURNAL OF OPERATIONS & PRODUCTION MANAGEMENT*, 16(4):672–684, DEC 2019
- K. Al-Htaybat, K. Hutaibat, and L. von Alberti-Alhtaybat. Global brain-reflective accounting practices forms of intellectual capital contributing to value creation and sustainable development. *JOURNAL OF INTELLECTUAL CAPITAL*, 20(6):733–762, NOV 28 2019
- P. Grover, A. K. Kar, and M. Janssen. Diffusion of blockchain technology insights from academic literature and social media analytics. *JOURNAL OF ENTERPRISE INFORMATION MANAGEMENT*, 32(5):735–757, SEP 4 2019
- S. E. Chang, Y.-C. Chen, and M.-F. Lu. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 144:1–11, JUL 2019

- P. Grover, A. K. Kar, M. Janssen, and P. V. Ilavarasan. Perceived usefulness, ease of use and user acceptance of blockchain technology for digital transactions - insights from user-generated content on twitter. *ENTERPRISE INFORMATION SYSTEMS*, 13(6):771–800, JUL 3 2019
- M. A. Nasir, T. L. D. Huynh, S. P. Nguyen, and D. Duong. Forecasting cryptocurrency returns and volume using search engines. *FINANCIAL INNOVATION*, 5(1), JAN 10 2019
- Y. Fu and J. Zhu. Operation mechanisms for intelligent logistics system: A blockchain perspective. *IEEE ACCESS*, 7:144202–144213, 2019
- S. Cai, M. Xu, and L. Zhang. Automatic information disclosure with value chains based on blockchain technology. In B. Xu, editor, *PROCEEDINGS OF 2019 IEEE 8TH JOINT INTERNATIONAL INFORMATION TECHNOLOGY AND ARTIFICIAL INTELLIGENCE CONFERENCE (ITAIC 2019)*, pages 1534–1538. IEEE; IEEE Beijing Sect; Chongqing Global Union Acad Sci & Technol; Chongqing Univ Technol; Chengdu Global Union Acad Sci & Technol; Chongqing Geeks Educ Technol Co Ltd, 2019. IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, PEOPLES R CHINA, MAY 24-26, 2019
- K. Al-Htaybat, K. Hutaibat, and L. von Alberti-Alhtaybat. Global brain-reflective accounting practices: Forms of intellectual capital contributing to value creation and sustainable development. *Journal of Intellectual Capital*, 20:733–762, 11 2019
- S. E. Chang, Y.-C. Chen, and M.-F. Lu. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. *Technological Forecasting and Social Change*, 144:1–11, 2019
- B. N. Joao. Blockchain and the potential of new business models: A systematic mapping. *REVISTA DE GESTAO E PROJETOS*, 9(3):33–48, SEP-DEC 2018
- D. Alonso-Martinez. Social progress and international patent collaboration. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 134:169–177, SEP 2018
- M. Moreno Munoz. Technological mediation of social interaction and the risks of its instrumentalization. the case of the facebook platform. *GAZETA DE ANTROPOLOGIA*, 34(2), JUL-DEC 2018
- S. Prasad, R. Shankar, R. Gupta, and S. Roy. A tism modeling of critical success factors of blockchain based cloud services. *JOURNAL OF ADVANCES IN MANAGEMENT RESEARCH*, 15(4):434–456, 2018

- J. Reveley. Embracing the humanistic vision: Recurrent themes in peter roberts' recent writings. *EDUCATIONAL PHILOSOPHY AND THEORY*, 50(3):312–321, 2018
- M.-L. Marsal-Llacuna and M. Oliver-Riera. The standards revolution: Who will first put this new kid on the blockchain ? In *2017 ITU KALEIDOSCOPE: CHALLENGES FOR A DATA-DRIVEN SOCIETY (ITU K)*. Ins Elect & Elect Engineers; ITU Kaleidoscope; Jiangsu Inst Commun; Nanjing Fiberhome Starrysky; Nanjing Ironhorse Informat Technol; HBC; New H3C Technologies; IEEE Commun Soc; Int Conf Standardizat & Innovat Informat Technol, 2017. ITU Kaleidoscope Conference - Challenges for a Data-Driven Society (ITU K), Nanjing, PEOPLES R CHINA, NOV 27-29, 2017
- F. Heylighen and M. Lenartowicz. The global brain as a model of the future information society: An introduction to the special issue. *Technological Forecasting and Social Change*, 114:1–6, 2017
- C. Last. Global commons in the global brain. *Technological Forecasting and Social Change*, 114:48–64, 2017

References

- [1] The QED Manifesto. In A. Bundy, editor, *CADE*, volume 814 of *LNCS*, pages 238–251. Springer, 1994.
- [2] S. Abbate, P. Centobelli, R. Cerchione, E. Oropallo, and E. Riccio. Kick-start your scientific journey into the metaverse. *KNOWLEDGE MANAGEMENT & E-LEARNING-AN INTERNATIONAL JOURNAL*, 15(1, SI):103–114, MAR 2023.
- [3] I. Abdelaziz, V. Thost, M. Crouse, and A. Fokoue. An experimental study of formula embeddings for automated theorem proving in first-order logic. *ArXiv*, abs/2002.00423, 2020.
- [4] K. Al-Htaybat, K. Hutaibat, and L. von Alberti-Alhtaybat. Global brain-reflective accounting practices forms of intellectual capital contributing to value creation and sustainable development. *JOURNAL OF INTELLECTUAL CAPITAL*, 20(6):733–762, NOV 28 2019.
- [5] K. Al-Htaybat, K. Hutaibat, and L. von Alberti-Alhtaybat. Global brain-reflective accounting practices: Forms of intellectual capital contributing to value creation and sustainable development. *Journal of Intellectual Capital*, 20:733–762, 11 2019.
- [6] J. Alama, T. Heskes, D. Kühlwein, E. Tsivtsivadze, and J. Urban. Premise selection for mathematics by corpus analysis and kernel methods. *J. Autom. Reasoning*, 52(2):191–213, 2014.
- [7] A. A. Alemi, F. Chollet, N. Eén, G. Irving, C. Szegedy, and J. Urban. DeepMath - deep sequence models for premise selection. In D. D. Lee, M. Sugiyama, U. V. Luxburg, I. Guyon, and R. Garnett, editors, *Advances in Neural Information Processing Systems 29: Annual Conference on Neural Information Processing Systems 2016, December 5-10, 2016, Barcelona, Spain*, pages 2235–2243, 2016.
- [8] R. Alkhudary, X. Brusset, and P. Fenies. Blockchain in general management and economics: a systematic literature review. *EUROPEAN BUSINESS REVIEW*, 32(4):765–783, AUG 9 2020.
- [9] D. Alonso-Martinez. Social progress and international patent collaboration. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 134:169–177, SEP 2018.
- [10] P. Andrews. *An Introduction to Mathematical Logic and Type Theory*. Applied Logic Series. Springer Netherlands, 2002.
- [11] L. Ante. Smart contracts on the blockchain - a bibliometric analysis and review. *TELEMATICS AND INFORMATICS*, 57, MAR 2021.

- [12] K. Appel and W. Haken. Every planar map is four colorable. *Bulletin of the American mathematical Society*, 82(5):711–712, 1976.
- [13] K. I. Appel and W. Haken. *Every planar map is four colorable*, volume 98. American Mathematical Soc., 1989.
- [14] M. Armand, G. Faure, B. Grégoire, C. Keller, L. Théry, and B. Werner. A modular integration of sat/smt solvers to coq through proof witnesses. In J.-P. Jouannaud and Z. Shao, editors, *Certified Programs and Proofs*, pages 135–150, Berlin, Heidelberg, 2011. Springer Berlin Heidelberg.
- [15] A. Assaf, G. Burel, R. Cauderlier, D. Delahaye, G. Dowek, C. Dubois, F. Gilbert, P. Halmagrand, O. Hermant, and R. Saillard. *Dedukti : a logical framework based on the λ π -calculus modulo theory*, 2016.
- [16] E. Aygün, L. Orseau, A. Anand, X. Glorot, V. Firoiu, L. M. Zhang, D. Precup, and S. Mourad. Proving theorems using incremental learning and hindsight experience replay. *CoRR*, abs/2112.10664, 2021.
- [17] L. Bachmair and H. Ganzinger. Rewrite-based equational theorem proving with selection and simplification. *Journal of Logic and Computation*, 4(3):217–247, 1994.
- [18] L. Bachmair, H. Ganzinger, and U. Waldmann. Superposition with simplification as a decision procedure for the monadic class with equality. In *Kurt Gödel Colloquium on Computational Logic and Proof Theory*, pages 83–96. Springer, 1993.
- [19] G. Bancerek. Automatic translation in formalized mathematics. 2006.
- [20] G. Bancerek, C. Bylinski, A. Grabowski, A. Kornilowicz, R. Matuszewski, A. Naumowicz, and K. Pak. The role of the Mizar Mathematical Library for interactive proof development in Mizar. *J. Autom. Reason.*, 61(1-4):9–32, 2018.
- [21] G. Bancerek, A. Naumowicz, and J. Urban. System description: XSL-based translator of Mizar to LaTeX. In F. Rabe, W. M. Farmer, G. O. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 1–6. Springer, 2018.
- [22] K. Bansal, S. Loos, M. Rabe, C. Szegedy, and S. Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In K. Chaudhuri and R. Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463. PMLR, 09–15 Jun 2019.

- [23] H. Barbosa, C. W. Barrett, M. Brain, G. Kremer, H. Lachnitt, M. Mann, A. Mohamed, M. Mohamed, A. Niemetz, A. Nötzli, A. Ozdemir, M. Preiner, A. Reynolds, Y. Sheng, C. Tinelli, and Y. Zohar. cvc5: A versatile and industrial-strength SMT solver. In D. Fisman and G. Rosu, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 28th International Conference, TACAS 2022, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2022, Munich, Germany, April 2-7, 2022, Proceedings, Part I*, volume 13243 of *Lecture Notes in Computer Science*, pages 415–442. Springer, 2022.
- [24] C. W. Barrett, R. Sebastiani, S. A. Seshia, and C. Tinelli. Handbook of satisfiability. In *Handbook of Satisfiability*, 2021.
- [25] F. Bártek and M. Suda. Learning precedences from simple symbol features. In P. Fontaine, K. Korovin, I. S. Kotsireas, P. Rümmer, and S. Tourret, editors, *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual)*, volume 2752 of *CEUR Workshop Proceedings*, pages 21–33. CEUR-WS.org, 2020.
- [26] F. Bártek and M. Suda. Neural precedence recommender. In A. Platzer and G. 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*, pages 525–542. Springer, 2021.
- [27] R. Bayardo and R. Schrag. Using csp look-back techniques to solve real-world sat instances. *Proceedings of the National Conference on Artificial Intelligence*, 03 1998.
- [28] C. Benzmüller and B. W. Paleo. Gödel’s god in isabelle/hol. *Archive of Formal Proofs*, November 2013. <https://isa-afp.org/entries/Goede1God.html>, Formal proof development.
- [29] P. C. Bhatt, V. Kumar, T.-C. Lu, and T. Daim. Technology convergence assessment: Case of blockchain within the ir 4.0 platform. *TECHNOLOGY IN SOCIETY*, 67, NOV 2021.
- [30] W. Bibel. Matings in Matrices. *Communications of the ACM*, 26(11):844–852, 1983.
- [31] W. Bibel. *Automated theorem proving, 2nd Edition*. Artificial intelligence. Vieweg, 1987.
- [32] W. Bibel. *Deduction - automated logic*. Academic Press, 1993.

- [33] L. Blaauwbroek, J. Urban, and H. Geuvers. Tactic learning and proving for the Coq proof assistant. In E. Albert and L. Kovacs, editors, *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73 of *EPiC Series in Computing*, pages 138–150. EasyChair, 2020.
- [34] L. Blaauwbroek, J. Urban, and H. Geuvers. The Tactician - A seamless, interactive tactic learner and prover for coq. In *CICM*, volume 12236 of *Lecture Notes in Computer Science*, pages 271–277. Springer, 2020.
- [35] J. C. Blanchette, D. Greenaway, C. Kaliszyk, D. Kühlwein, and J. Urban. A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning*, 57(3):219–244, 2016.
- [36] J. C. Blanchette, M. W. Haslbeck, D. Matichuk, and T. Nipkow. Mining the archive of formal proofs. In M. Kerber, J. Carette, C. Kaliszyk, F. Rabe, and V. Sorge, editors, *Intelligent Computer Mathematics - International Conference, CICM 2015, Washington, DC, USA, July 13-17, 2015, Proceedings*, volume 9150 of *Lecture Notes in Computer Science*, pages 3–17. Springer, 2015.
- [37] J. C. Blanchette, C. Kaliszyk, L. C. Paulson, and J. Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016.
- [38] F. Blanqui, G. Dowek, É. Grienerberger, G. Hondet, and F. Thiré. Some axioms for mathematics. *CoRR*, abs/2111.00543, 2021.
- [39] S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *IJCAR*, volume 6173 of *LNCS*, pages 107–121. Springer, 2010.
- [40] S. Böhme and T. Nipkow. Sledgehammer: Judgement Day. In J. Giesl and R. Hähnle, editors, *Proc. of the 5th IJCAR, Edinburgh*, volume 6173 of *LNAI*, pages 107–121. Springer, 2010.
- [41] B. E. Boser, I. M. Guyon, and V. N. Vapnik. A training algorithm for optimal margin classifiers. In *Proceedings of the Fifth Annual Workshop on Computational Learning Theory, COLT '92*, pages 144–152, New York, NY, USA, 1992. ACM.
- [42] P. Braselmann and P. Koepke. Gödel’s completeness theorem. *Formalized Mathematics*, 13(1):49–53, 2005.
- [43] C. Brown and C. Rizkallah. Glivenko and kuroda for simple type theory. *The Journal of Symbolic Logic*, 79(2):485–495, 06 2014.
- [44] C. E. Brown, T. Gauthier, C. Kaliszyk, G. Sutcliffe, and J. Urban. GRUNGE: A grand unified ATP challenge. In *CADE*, volume 11716 of *Lecture Notes in Computer Science*, pages 123–141. Springer, 2019.

- [45] C. E. Brown and K. Pałk. A tale of two set theories. In C. Kaliszzyk, E. Brady, A. Kohlhase, and C. Sacerdoti Coen, editors, *Intelligent Computer Mathematics*, pages 44–60, Cham, 2019. Springer International Publishing.
- [46] S. Cai, M. Xu, and L. Zhang. Automatic information disclosure with value chains based on blockchain technology. In B. Xu, editor, *PROCEEDINGS OF 2019 IEEE 8TH JOINT INTERNATIONAL INFORMATION TECHNOLOGY AND ARTIFICIAL INTELLIGENCE CONFERENCE (ITAIC 2019)*, pages 1534–1538. IEEE; IEEE Beijing Sect; Chongqing Global Union Acad Sci & Technol; Chongqing Univ Technol; Chengdu Global Union Acad Sci & Technol; Chongqing Geeks Educ Technol Co Ltd, 2019. IEEE 8th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Chongqing, PEOPLES R CHINA, MAY 24-26, 2019.
- [47] W. Carnielli, M. E. Coniglio, and A. Rodrigues. Recovery operators, paraconsistency and duality. *Logic Journal of the IGPL*, 28(5):624–656, 01 2019.
- [48] W. A. Carnielli and D. Fuenmayor. Gödel blooming: the incompleteness theorems from a paraconsistent perspective. 2020.
- [49] P. Centobelli, R. Cerchione, E. Esposito, and E. Oropallo. Surfing blockchain wave, or drowning? shaping the future of distributed ledgers and decentralized technologies. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 165, APR 2021.
- [50] D. J. Chalmers. *Constructing the World*. Oxford University Press, 2012.
- [51] S. E. Chang, Y.-C. Chen, and M.-F. Lu. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 144:1–11, JUL 2019.
- [52] S. E. Chang, Y.-C. Chen, and M.-F. Lu. Supply chain re-engineering using blockchain technology: A case of smart contract based tracking process. *Technological Forecasting and Social Change*, 144:1–11, 2019.
- [53] M. Chen, J. Tworek, H. Jun, Q. Yuan, H. P. d. O. Pinto, J. Kaplan, H. Edwards, Y. Burda, N. Joseph, G. Brockman, A. Ray, R. Puri, G. Krueger, M. Petrov, H. Khlaaf, G. Sastry, P. Mishkin, B. Chan, S. Gray, N. Ryder, M. Pavlov, A. Power, L. Kaiser, M. Bavarian, C. Winter, P. Tillet, F. P. Such, D. Cummings, M. Plappert, F. Chantzis, E. Barnes, A. Herbert-Voss, W. H. Guss, A. Nichol, A. Paino, N. Tezak, J. Tang, I. Babuschkin, S. Balaji, S. Jain, W. Saunders, C. Hesse, A. N. Carr, J. Leike, J. Achiam, V. Misra, E. Morikawa, A. Radford, M. Knight, M. Brundage, M. Murati, K. Mayer, P. Welinder, B. McGrew, D. Amodei, S. McCandlish, I. Sutskever, and W. Zaremba. Evaluating large language models trained on code, 2021.

- [54] T. Chen and C. Guestrin. XGBoost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pages 785–794, New York, NY, USA, 2016. ACM.
- [55] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, P. Schuh, K. Shi, S. Tsvyashchenko, J. Maynez, A. Rao, P. Barnes, Y. Tay, N. Shazeer, V. Prabhakaran, E. Reif, N. Du, B. Hutchinson, R. Pope, J. Bradbury, J. Austin, M. Isard, G. Gur-Ari, P. Yin, T. Duke, A. Levskaya, S. Ghemawat, S. Dev, H. Michalewski, X. Garcia, V. Misra, K. Robinson, L. Fedus, D. Zhou, D. Ippolito, D. Luan, H. Lim, B. Zoph, A. Spiridonov, R. Sepassi, D. Dohan, S. Agrawal, M. Omernick, A. M. Dai, T. S. Pillai, M. Pellat, A. Lewkowycz, E. Moreira, R. Child, O. Polozov, K. Lee, Z. Zhou, X. Wang, B. Saeta, M. Diaz, O. Firat, M. Catasta, J. Wei, K. Meier-Hellstern, D. Eck, J. Dean, S. Petrov, and N. Fiedel. Palm: Scaling language modeling with pathways, 2022.
- [56] K. Chvalovský, J. Jakubův, M. Suda, and J. Urban. ENIGMA-NG: efficient neural and gradient-boosted inference guidance for E. In P. 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*, pages 197–215. Springer, 2019.
- [57] K. Chvalovský, J. Jakubův, M. Olšák, and J. Urban. Learning theorem proving components. In A. Das and S. Negri, editors, *Automated Reasoning with Analytic Tableaux and Related Methods*, pages 266–278, Cham, 2021. Springer International Publishing.
- [58] P. Cintula, C. G. Fermüller, and C. Noguera. Fuzzy Logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2021 edition, 2021.
- [59] The Coq Proof Assistant. <http://coq.inria.fr>.
- [60] M. Cramer, B. Fisseni, P. Koepke, D. Kühlwein, B. Schröder, and J. Veldman. The Naproche Project: Controlled Natural Language Proof Checking of Mathematical Texts. In N. E. Fuchs, editor, *CNL*, volume 5972 of *LNCS*, pages 170–186. Springer, 2009.
- [61] Ł. Czajka. Practical proof search for coq by type inhabitation. In N. Peltier and V. Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 28–57, Cham, 2020. Springer International Publishing.
- [62] z. Czajka and C. Kaliszyk. Hammer for coq: Automation for dependent type theory. *J. Autom. Reason.*, 61(1–4):423–453, jun 2018.

- [63] M. Davis, G. Logemann, and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, 5(7):394–397, 1962.
- [64] L. Demey, B. Kooi, and J. Sack. Logic and Probability. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2019 edition, 2019.
- [65] J. Denzinger, M. Fuchs, C. Goller, and S. Schulz. Learning from Previous Proof Experience. Technical Report AR99-4, Institut für Informatik, Technische Universität München, 1999.
- [66] M. Desharnais, P. Vukmirović, J. Blanchette, and M. Wenzel. Seventeen Provers Under the Hammer. In J. Andronick and L. de Moura, editors, *13th International Conference on Interactive Theorem Proving (ITP 2022)*, volume 237 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 8:1–8:18, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik. <https://matryoshka-project.github.io/pubs/seventeen.pdf>.
- [67] R. Di Cosmo and D. Miller. Linear Logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2019 edition, 2019.
- [68] G. Dowek. From the universality of mathematical truth to the interoperability of proof systems. In J. Blanchette, L. Kovács, and D. Pattinson, editors, *Automated Reasoning*, pages 8–11, Cham, 2022. Springer International Publishing.
- [69] Y. K. Dwivedi, A. Sharma, N. P. Rana, M. Giannakis, P. Goel, and V. Dutot. Evolution of artificial intelligence research in technological forecasting and social change: Research topics, trends, and future directions. *Technological Forecasting and Social Change*, 192:122579, 2023.
- [70] H. Ebbinghaus, J. Flum, and W. Thomas. *Mathematical Logic*. Graduate Texts in Mathematics. Springer International Publishing, 2021.
- [71] M. Eberl and L. C. Paulson. The prime number theorem. *Archive of Formal Proofs*, September 2018. https://isa-afp.org/entries/Prime_Number_Theorem.html, Formal proof development.
- [72] D. El Ouraoui. *Méthodes pour le raisonnement d'ordre supérieur dans SMT*. PhD thesis, Université de Lorraine, 2021.
- [73] R.-E. Fan, K.-W. Chang, C.-J. Hsieh, X.-R. Wang, and C.-J. Lin. Liblinear: A library for large linear classification. *J. Mach. Learn. Res.*, 9:1871–1874, June 2008.
- [74] M. Färber and C. Kaliszky. Random forests for premise selection. In C. Lutz and S. Ranise, editors, *Frontiers of Combining Systems - 10th*

- International Symposium, FroCoS 2015, Wroclaw, Poland, September 21-24, 2015. Proceedings*, volume 9322 of *Lecture Notes in Computer Science*, pages 325–340. Springer, 2015.
- [75] M. Färber, C. Kaliszyk, and J. Urban. Monte Carlo tableau proof search. In L. de Moura, editor, *Automated Deduction - CADE 26 - 26th International Conference on Automated Deduction, Gothenburg, Sweden, August 6-11, 2017, Proceedings*, volume 10395 of *Lecture Notes in Computer Science*, pages 563–579. Springer, 2017.
- [76] W. M. Farmer. The seven virtues of simple type theory. *Journal of Applied Logic*, 6(3):267–286, 2008.
- [77] S. Faustino. Deleuze in the wild: making philosophy matter for fintech. *JOURNAL OF CULTURAL ECONOMY*, 15(1, SI):93–102, JAN 2 2022.
- [78] Y. Fu and J. Zhu. Operation mechanisms for intelligent logistics system: A blockchain perspective. *IEEE ACCESS*, 7:144202–144213, 2019.
- [79] J. Garson. Modal Logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Summer 2021 edition, 2021.
- [80] T. Gauthier. Deep reinforcement learning for synthesizing functions in higher-order logic. In E. Albert and L. Kovacs, editors, *LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning*, volume 73 of *EPiC Series in Computing*, pages 230–248. EasyChair, 2020.
- [81] T. Gauthier. Tree neural networks in hol4. In C. Benzmüller and B. Miller, editors, *Intelligent Computer Mathematics*, pages 278–283, Cham, 2020. Springer International Publishing.
- [82] T. Gauthier, C. Kaliszyk, and J. Urban. TacticToe: Learning to reason with HOL4 tactics. In T. Eiter and D. Sands, editors, *LPAR-21, 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning, Maun, Botswana, May 7-12, 2017*, volume 46 of *EPiC Series in Computing*, pages 125–143. EasyChair, 2017.
- [83] T. Gauthier, C. Kaliszyk, J. Urban, R. Kumar, and M. Norrish. TacticToe: Learning to prove with tactics. *J. Autom. Reason.*, 65(2):257–286, 2021.
- [84] T. Gauthier and J. Urban. Learning program synthesis for integer sequences from scratch, 2022.
- [85] K. Gödel. Die vollständigkeit der axiome des logischen funktionenkalküls. *Monatshefte für Mathematik und Physik*, 37:349–360, 1930.
- [86] B. Goertzel, S. Giacomelli, D. Hanson, C. Pennachin, and M. Argentieri. Singularitynet: A decentralized, open market and inter-network for ais. *Thoughts, Theories Stud. Artif. Intell. Res.*, 2017.

- [87] B. Goertzel, T. Goertzel, and Z. Goertzel. The global brain and the emerging economy of abundance: Mutualism, open collaboration, exchange networks and the automated commons. *TECHNOLOGICAL FORECASTING AND SOCIAL CHANGE*, 114:65–73, JAN 2017.
- [88] Z. Goertzel. Offer networks simulation and dynamics. Master’s thesis, University of Copenhagen, Computer Science Dept., 2017.
- [89] Z. Goertzel, J. Jakubův, S. Schulz, and J. Urban. ProofWatch: Watchlist guidance for large theories in E. In J. Avigad and A. Mahboubi, editors, *Interactive Theorem Proving - 9th International Conference, ITP 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 9-12, 2018, Proceedings*, volume 10895 of *Lecture Notes in Computer Science*, pages 270–288. Springer, 2018.
- [90] Z. Goertzel, J. Jakubův, and J. Urban. ProofWatch meets ENIGMA: First experiments. In G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, and M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 15–22. EasyChair, 2018.
- [91] Z. Goertzel, J. Jakubův, and J. Urban. ENIGMAWatch: ProofWatch meets ENIGMA. In S. Cerrito and A. Popescu, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 28th International Conference, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2019.
- [92] Z. Goertzel, J. Jakubuv, and J. Urban. First experiments with watchlist guidance on Mizar. *AITP 2018*, 2018.
- [93] Z. Goertzel, J. Jakubuv, and J. Urban. ProofWatch meets ENIGMA: First experiments. In G. Barthe, K. Korovin, S. Schulz, M. Suda, G. Sutcliffe, and M. Veanes, editors, *LPAR-22 Workshop and Short Paper Proceedings*, volume 9 of *Kalpa Publications in Computing*, pages 15–22. EasyChair, 2018.
- [94] Z. Goertzel and J. Urban. Usefulness of lemmas via graph neural networks. *AITP 2019*, 2019.
- [95] Z. A. Goertzel. Make E Smart Again. *AITP 2020*, 2020.
- [96] Z. A. Goertzel. Make E Smart Again (short paper). In *IJCAR (2)*, volume 12167 of *Lecture Notes in Computer Science*, pages 408–415. Springer, 2020.
- [97] Z. A. Goertzel, K. Chvalovský, J. Jakubuv, M. Olsák, and J. Urban. Fast and slow Enigmas and Parental Guidance. In B. Konev and G. 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 173–191. Springer, 2021.
- [98] Z. A. Goertzel, J. Jakubův, C. Kaliszyk, M. Olšák, J. Piepenbrock, and J. Urban. The Isabelle ENIGMA. In J. Andronick and L. de Moura, editors, *13th International Conference on Interactive Theorem Proving (ITP 2022)*, volume 237 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 16:1–16:21, Dagstuhl, Germany, 2022. Schloss Dagstuhl – Leibniz-Zentrum für Informatik.
- [99] Z. A. Goertzel, J. Jakubuv, and J. Urban. Parental guidance in E. *AITP 2021*, 2021.
- [100] Z. A. Goertzel, A. Pease, and J. Urban. Project proposal: Formal ethics ontology in SUMO. *AITP 2022*, 2022.
- [101] G. Gonthier. The four colour theorem: Engineering of a formal proof. In D. Kapur, editor, *Computer Mathematics, 8th Asian Symposium, ASCM 2007, Singapore, December 15-17, 2007. Revised and Invited Papers*, volume 5081 of *LNCS*, page 333. Springer, 2007.
- [102] G. Gonthier, A. Asperti, J. Avigad, Y. Bertot, C. Cohen, F. Garillot, S. L. Roux, A. Mahboubi, R. O’Connor, S. O. Biha, I. Pasca, L. Rideau, A. Solovyev, E. Tassi, and L. Théry. A machine-checked proof of the Odd Order Theorem. In S. Blazy, C. Paulin-Mohring, and D. Pichardie, editors, *ITP*, volume 7998 of *LNCS*, pages 163–179. Springer, 2013.
- [103] G. Gonthier et al. Formal proof—the four-color theorem. *Notices of the AMS*, 55(11):1382–1393, 2008.
- [104] G. Gottlob, G. Sutcliffe, and A. Voronkov, editors. *Global Conference on Artificial Intelligence, GCAI 2015, Tbilisi, Georgia, October 16-19, 2015*, volume 36 of *EPiC Series in Computing*. EasyChair, 2015.
- [105] A. Grabowski, A. Kornilowicz, and A. Naumowicz. Mizar in a nutshell. *J. Formalized Reasoning*, 3(2):153–245, 2010.
- [106] P. Grover, A. K. Kar, and M. Janssen. Diffusion of blockchain technology insights from academic literature and social media analytics. *JOURNAL OF ENTERPRISE INFORMATION MANAGEMENT*, 32(5):735–757, SEP 4 2019.
- [107] P. Grover, A. K. Kar, M. Janssen, and P. V. Ilavarasan. Perceived usefulness, ease of use and user acceptance of blockchain technology for digital transactions - insights from user-generated content on twitter. *ENTERPRISE INFORMATION SYSTEMS*, 13(6):771–800, JUL 3 2019.
- [108] C. Hahn, F. Schmitt, J. U. Kreber, M. N. Rabe, and B. Finkbeiner. Teaching temporal logics to neural networks. In *International Conference on Learning Representations*, 2021.

- [109] T. Hales, M. Adams, G. Bauer, T. D. Dang, J. Harrison, H. Le Truong, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, et al. A formal proof of the kepler conjecture. In *Forum of mathematics, Pi*, volume 5. Cambridge University Press, 2017.
- [110] T. C. Hales. A proof of the kepler conjecture. *Annals of mathematics*, pages 1065–1185, 2005.
- [111] T. C. Hales. Introduction to the Flyspeck project. In T. Coquand, H. Lombardi, and M.-F. Roy, editors, *Mathematics, Algorithms, Proofs*, number 05021 in Dagstuhl Seminar Proceedings, pages 1–11, Dagstuhl, Germany, 2006. Internationales Begegnungs- und Forschungszentrum für Informatik (IBFI), Schloss Dagstuhl, Germany.
- [112] T. C. Hales, M. Adams, G. Bauer, D. T. Dang, J. Harrison, T. L. Hoang, C. Kaliszyk, V. Magron, S. McLaughlin, T. T. Nguyen, T. Q. Nguyen, T. Nipkow, S. Obua, J. Pleso, J. Rute, A. Solovyev, A. H. T. Ta, T. N. Tran, D. T. Trieu, J. Urban, K. K. Vu, and R. Zumkeller. A formal proof of the Kepler conjecture. *CoRR*, abs/1501.02155, 2015.
- [113] J. M. Han, J. Rute, Y. Wu, E. Ayers, and S. Polu. Proof artifact co-training for theorem proving with language models. In *International Conference on Learning Representations*, 2022.
- [114] J. M. Han and F. van Doorn. A formal proof of the independence of the continuum hypothesis. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 353–366, New York, NY, USA, 2020. Association for Computing Machinery.
- [115] J. Harrison. HOL Light: A tutorial introduction. In M. K. Srivas and A. J. Camilleri, editors, *FMCAD*, volume 1166 of *LNCS*, pages 265–269. Springer, 1996.
- [116] J. Harrison, J. Urban, and F. Wiedijk. History of interactive theorem proving. In J. H. Siekmann, editor, *Computational Logic*, volume 9 of *Handbook of the History of Logic*, pages 135–214. Elsevier, 2014.
- [117] D. Hendrycks, C. Burns, S. Kadavath, A. Arora, S. Basart, E. Tang, D. Song, and J. Steinhardt. Measuring mathematical problem solving with the math dataset. *NeurIPS*, 2021.
- [118] F. Heylighen and M. Lenartowicz. The global brain as a model of the future information society: An introduction to the special issue. *Technological Forecasting and Social Change*, 114:1–6, 2017.
- [119] A. Heyting. *Intuitionism an Introduction*. Amsterdam, Netherlands: North-Holland, 1956.

- [120] D. Hilbert and W. Ackermann. *Grundzuge der Theoretischen Logik*. Springer Verlag, 1928.
- [121] K. Hoder, G. Reger, M. Suda, and A. Voronkov. Selecting the selection. In N. Olivetti and A. Tiwari, editors, *Proc. of the 8th IJCAR, Coimbra*, volume 9706 of *LNAI*, pages 313–329. Springer, 2016.
- [122] K. Hoder and A. Voronkov. Sine qua non for large theory reasoning. In N. Bjørner and V. Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.
- [123] E. K. Holden and K. Korovin. Heterogeneous heuristic optimisation and scheduling for first-order theorem proving. In *CICM*, volume 12833 of *Lecture Notes in Computer Science*, pages 107–123. Springer, 2021.
- [124] E. K. Holden and K. Korovin. Graph sequence learning for premise selection. *arXiv preprint arXiv:2303.15642*, 2023.
- [125] D. Huang, P. Dhariwal, D. Song, and I. Sutskever. Gamepad: A learning environment for theorem proving. In *International Conference on Learning Representations*, 2019.
- [126] L. Hupel and Y. Zhang. Cakeml. *Archive of Formal Proofs*, March 2018. <https://isa-afp.org/entries/CakeML.html>, Formal proof development.
- [127] M. Hutter. *Universal artificial intelligence: Sequential decisions based on algorithmic probability*. Springer Science & Business Media, 2004.
- [128] J. Jakubův, K. Chvalovský, M. Olsák, B. Piotrowski, M. Suda, and J. Urban. ENIGMA anonymous: Symbol-independent inference guiding machine (system description). In N. Peltier and V. 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.
- [129] J. Jakubův and J. Urban. BliStrTune: hierarchical invention of theorem proving strategies. In Y. Bertot and V. Vafeiadis, editors, *Proceedings of the 6th ACM SIGPLAN Conference on Certified Programs and Proofs, CPP 2017, Paris, France, January 16-17, 2017*, pages 43–52. ACM, 2017.
- [130] J. Jakubův and J. Urban. ENIGMA: efficient learning-based inference guiding machine. In H. Geuvers, M. England, O. Hasan, F. Rabe, and O. 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.

- [131] J. Jakubův and J. Urban. Enhancing ENIGMA given clause guidance. In F. Rabe, W. M. Farmer, G. O. Passmore, and A. Youssef, editors, *Intelligent Computer Mathematics - 11th International Conference, CICM 2018, Hagenberg, Austria, August 13-17, 2018, Proceedings*, volume 11006 of *Lecture Notes in Computer Science*, pages 118–124. Springer, 2018.
- [132] J. Jakubův and J. Urban. Hierarchical invention of theorem proving strategies. *AI Commun.*, 31(3):237–250, 2018.
- [133] J. Jakubův and J. Urban. Hammering Mizar by learning clause guidance. In J. Harrison, J. O’Leary, and A. 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.
- [134] J. Jakubuv and C. Kaliszyk. Relaxed weighted path order in theorem proving. *Mathematics in Computer Science*, 14, 09 2020.
- [135] A. Q. Jiang, W. Li, S. Tworkowski, K. Czechowski, T. Odrzygóźdź, P. Miłoś, Y. Wu, and M. Jamnik. Thor: Wielding hammers to integrate language models and automated theorem provers. In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- [136] B. N. Joao. Blockchain and the potential of new business models: A systematic mapping. *REVISTA DE GESTAO E PROJETOS*, 9(3):33–48, SEP-DEC 2018.
- [137] C. Kaliszyk and J. Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 53(2):173–213, 2014.
- [138] C. Kaliszyk and J. Urban. FEMaLeCoP: Fairly efficient machine learning connection prover. In M. Davis, A. Fehnker, A. McIver, and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 20th International Conference, LPAR-20 2015, Suva, Fiji, November 24-28, 2015, Proceedings*, volume 9450 of *Lecture Notes in Computer Science*, pages 88–96. Springer, 2015.
- [139] C. Kaliszyk and J. Urban. MizAR 40 for Mizar 40. *J. Autom. Reasoning*, 55(3):245–256, 2015.
- [140] C. Kaliszyk, J. Urban, H. Michalewski, and M. Olsák. Reinforcement learning of theorem proving. In *Advances in Neural Information Processing Systems 31: Annual Conference on Neural Information Processing Systems 2018, NeurIPS 2018, 3-8 December 2018, Montréal, Canada.*, pages 8836–8847, 2018.
- [141] C. Kaliszyk, J. Urban, and J. Vyskočil. Automating formalization by statistical and semantic parsing of mathematics. In M. Ayala-Rincón and

- C. A. Muñoz, editors, *Interactive Theorem Proving*, pages 12–27, Cham, 2017. Springer International Publishing.
- [142] C. Kaliszyk, J. Urban, and J. Vyskočil. Learning to parse on aligned corpora (rough diamond). In C. Urban and X. Zhang, editors, *Interactive Theorem Proving - 6th International Conference, ITP 2015, Nanjing, China, August 24-27, 2015, Proceedings*, volume 9236 of *Lecture Notes in Computer Science*, pages 227–233. Springer, 2015.
- [143] S. Kamin. Two generalizations of the recursive path ordering. *Unpublished manuscript*, 1980.
- [144] G. Ke, Q. Meng, T. Finley, T. Wang, W. Chen, W. Ma, Q. Ye, and T. Liu. Lightgbm: A highly efficient gradient boosting decision tree. In *NIPS*, pages 3146–3154, 2017.
- [145] M. K. Kinyon, R. Veroff, and P. Vojtechovský. Loops with abelian inner mapping groups: An application of automated deduction. In M. P. Bonacina and M. E. Stickel, editors, *Automated Reasoning and Mathematics - Essays in Memory of William W. McCune*, volume 7788 of *LNCS*, pages 151–164. Springer, 2013.
- [146] A. Klarin. The decade-long cryptocurrencies and the blockchain rollercoaster: Mapping the intellectual structure and charting future directions. *RESEARCH IN INTERNATIONAL BUSINESS AND FINANCE*, 51, JAN 2020.
- [147] T. Klenze and C. Sprenger. Isanet: Formalization of a verification framework for secure data plane protocols. *Archive of Formal Proofs*, June 2022. <https://isa-afp.org/entries/IsaNet.html>, Formal proof development.
- [148] D. Knuth and P. Bendix. Simple Word Problems in Universal Algebras. In J. Leech, editor, *Computational Algebra*, pages 263–297. Pergamon Press, 1970.
- [149] M. Kohlhase and F. Rabe. Qed reloaded: Towards a pluralistic formal library of mathematical knowledge. *Journal of Formalized Reasoning*, 9(1):201–234, Jan. 2016.
- [150] A. Kornilowicz. Jordan curve theorem. *Formalized Mathematics*, 13(4):481–491, 2005.
- [151] K. Korovin. iprover - an instantiation-based theorem prover for first-order logic (system description). In A. Armando, P. Baumgartner, and G. Dowek, editors, *Automated Reasoning, 4th International Joint Conference, IJCAR 2008, Sydney, Australia, August 12-15, 2008, Proceedings*, volume 5195 of *Lecture Notes in Computer Science*, pages 292–298. Springer, 2008.

- [152] J. Kotowicz. Convergent real sequences. Upper and lower bound of sets of real numbers. *Formalized Mathematics*, 1(**3**):477–481, 1990.
- [153] L. Kovács and A. Voronkov. First-order theorem proving and Vampire. In N. Sharygina and H. Veith, editors, *CAV*, volume 8044 of *LNCS*, pages 1–35. Springer, 2013.
- [154] A. Krishna, S. Riedel, and A. Vlachos. Proofver: Natural logic theorem proving for fact verification. *CoRR*, abs/2108.11357, 2021.
- [155] A. Kubo and Y. Nakamura. Angle and triangle in Euclidean topological space. *Formalized Mathematics*, 11(**3**):281–287, 2003.
- [156] D. Kühlwein, T. van Laarhoven, E. Tsivtsivadze, J. Urban, and T. Heskes. Overview and evaluation of premise selection techniques for large theory mathematics. In B. Gramlich, D. Miller, and U. Sattler, editors, *IJCAR*, volume 7364 of *LNCS*, pages 378–392. Springer, 2012.
- [157] C. Last. Global commons in the global brain. *Technological Forecasting and Social Change*, 114:48–64, 2017.
- [158] C. Last. *Global Brain: Foundations of a Distributed Singularity*, pages 363–375. Springer International Publishing, Cham, 2020.
- [159] L. A. Levin. Universal sequential search problems. *Problemy peredachi informatsii*, 9(3):115–116, 1973.
- [160] L. A. Levin. Randomness conservation inequalities; information and independence in mathematical theories. *Information and Control*, 61(1):15–37, 1984.
- [161] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. BART: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 7871–7880, Online, July 2020. Association for Computational Linguistics.
- [162] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *arXiv preprint arXiv:2206.14858*, 2022.
- [163] A. Lewkowycz, A. Andreassen, D. Dohan, E. Dyer, H. Michalewski, V. Ramasesh, A. Slone, C. Anil, I. Schlag, T. Gutman-Solo, Y. Wu, B. Neyshabur, G. Gur-Ari, and V. Misra. Solving quantitative reasoning problems with language models, 2022.
- [164] W. Li, L. Yu, Y. Wu, and L. C. Paulson. Isarstep: a benchmark for high-level mathematical reasoning. In *International Conference on Learning Representations*, 2021.

- [165] C. Mangla, S. B. Holden, and L. C. Paulson. Bayesian ranking for strategy scheduling in automated theorem provers. In J. Blanchette, L. Kovács, and D. 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 559–577. Springer, 2022.
- [166] M. Manzano and V. Aranda. Many-Sorted Logic. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2022 edition, 2022.
- [167] J. Marques Silva and K. Sakallah. Grasp-a new search algorithm for satisfiability. In *Proceedings of International Conference on Computer Aided Design*, pages 220–227, 1996.
- [168] J. Marques-Silva and K. Sakallah. Grasp: a search algorithm for propositional satisfiability. *IEEE Transactions on Computers*, 48(5):506–521, 1999.
- [169] M.-L. Marsal-Llacuna and M. Oliver-Riera. The standards revolution: Who will first put this new kid on the blockchain ? In *2017 ITU KALEIDOSCOPE: CHALLENGES FOR A DATA-DRIVEN SOCIETY (ITU K)*. Ins Elect & Elect Engineers; ITU Kaleidoscope; Jiangsu Inst Commun; Nanjing Fiberhome Starrysky; Nanjing Ironhorse Informat Technol; HBC; New H3C Technologies; IEEE Commun Soc; Int Conf Standardizat & Innovat Informat Technol, 2017. ITU Kaleidoscope Conference - Challenges for a Data-Driven Society (ITU K), Nanjing, PEOPLES R CHINA, NOV 27-29, 2017.
- [170] W. W. McCune. Prover9 and Mace4. <http://www.cs.unm.edu/~mccune/prover9/>, 2005–2010. (accessed 2016-03-29).
- [171] M. McGrath and D. Frank. Propositions. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2020 edition, 2020.
- [172] M. J. Meirino, M. P. Mexas, A. d. V. Faria, R. P. Mexas, and G. D. Meirelles. Blockchain technology applications: A literature review. *BRAZILIAN JOURNAL OF OPERATIONS & PRODUCTION MANAGEMENT*, 16(4):672–684, DEC 2019.
- [173] A. Melkonyan, T. Gruchmann, F. Lohmar, and R. Bleischwitz. Decision support for sustainable urban mobility: A case study of the rhine-ruhr area. *SUSTAINABLE CITIES AND SOCIETY*, 80, MAY 2022.
- [174] E. Mendelson. *Introduction to Mathematical Logic*. Princeton: Van Nostrand, 1964.
- [175] J. Meng and L. C. Paulson. Lightweight relevance filtering for machine-generated resolution problems. *J. Applied Logic*, 7(1):41–57, 2009.

- [176] P. Minervini, M. Bosnjak, T. Rocktäschel, S. Riedel, and E. Grefenstette. Differentiable reasoning on large knowledge bases and natural language. In *The Thirty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2020, The Thirty-Second Innovative Applications of Artificial Intelligence Conference, IAAI 2020, The Tenth AAAI Symposium on Educational Advances in Artificial Intelligence, EAAI 2020, New York, NY, USA, February 7-12, 2020*, pages 5182–5190. AAAI Press, 2020.
- [177] P. Minervini, S. Riedel, P. Stenetorp, E. Grefenstette, and T. Rocktäschel. Learning reasoning strategies in end-to-end differentiable proving. In P. Hitzler and M. K. Sarker, editors, *Neuro-Symbolic Artificial Intelligence: The State of the Art*, volume 342 of *Frontiers in Artificial Intelligence and Applications*, pages 280–293. IOS Press, 2021.
- [178] G. A. Montes and B. Goertzel. Distributed, decentralized, and democratized artificial intelligence. *Technological Forecasting and Social Change*, 141:354–358, 2019.
- [179] M. Moreno Munoz. Technological mediation of social interaction and the risks of its instrumentalization. the case of the facebook platform. *GAZETA DE ANTROPOLOGIA*, 34(2), JUL-DEC 2018.
- [180] J. Moschovakis. Intuitionistic Logic. In E. N. Zalta and U. Nodelman, editors, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Winter 2022 edition, 2022.
- [181] L. d. Moura and S. Ullrich. The lean 4 theorem prover and programming language. In A. Platzer and G. Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 625–635, Cham, 2021. Springer International Publishing.
- [182] M. A. Nasir, T. L. D. Huynh, S. P. Nguyen, and D. Duong. Forecasting cryptocurrency returns and volume using search engines. *FINANCIAL INNOVATION*, 5(1), JAN 10 2019.
- [183] M. S. Nawaz, M. Sun, and P. Fournier-Viger. Proof guidance in PVS with sequential pattern mining. In H. Hojjat and M. Massink, editors, *Fundamentals of Software Engineering*, pages 45–60, Cham, 2019. Springer International Publishing.
- [184] I. Niles and A. Pease. Towards a standard upper ontology. In *FOIS*, pages 2–9, 2001.
- [185] T. Nipkow, L. C. Paulson, and M. Wenzel. *Isabelle/HOL: A Proof Assistant for Higher-Order Logic*, volume 2283 of *LNCS*. Springer, 2002.
- [186] P. Noce. Verification of a diffie-hellman password-based authentication protocol by extending the inductive method. *Archive of Formal Proofs*, January 2017. https://isa-afp.org/entries/Password_Authentication_Protocol.html, Formal proof development.

- [187] M. Olšák, C. Kaliszyk, and J. Urban. Property invariant embedding for automated reasoning. In G. D. Giacomo, A. Catalá, B. Dilkina, M. Milano, S. Barro, A. Bugarín, and J. Lang, editors, *ECAI 2020 - 24th European Conference on Artificial Intelligence, 29 August-8 September 2020, Santiago de Compostela, Spain, August 29 - September 8, 2020 - Including 10th Conference on Prestigious Applications of Artificial Intelligence (PAIS 2020)*, volume 325 of *Frontiers in Artificial Intelligence and Applications*, pages 1395–1402. IOS Press, 2020.
- [188] J. Otten. Clausal connection-based theorem proving in intuitionistic first-order logic. In B. Beckert, editor, *Automated Reasoning with Analytic Tableaux and Related Methods, International Conference, TABLEUX 2005, Koblenz, Germany, September 14-17, 2005, Proceedings*, volume 3702 of *LNCS*, pages 245–261. Springer, 2005.
- [189] J. Otten and W. Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003. <http://www.leancop.de/>.
- [190] S. H. Park. A formal cheri-c memory model. *Archive of Formal Proofs*, November 2022. https://isa-afp.org/entries/CHERI-C_Memory_Model.html, Formal proof development.
- [191] L. C. Paulson. Gödel’s incompleteness theorems. *Archive of Formal Proofs*, November 2013. <https://isa-afp.org/entries/Incompleteness.html>, Formal proof development.
- [192] A. Pease. *Ontology: A Practical Guide*. Articulate Software Press, 2011.
- [193] J. Piepenbrock, J. Urban, K. Korovin, M. Olšák, T. Heskes, and M. Janota. Machine learning meets the herbrand universe. *arXiv preprint arXiv:2210.03590*, 2022.
- [194] B. Piotrowski and J. Urban. ATPboost: Learning premise selection in binary setting with ATP feedback. In D. Galmiche, S. Schulz, and R. Sebastiani, editors, *Automated Reasoning - 9th International Joint Conference, IJCAR 2018, Held as Part of the Federated Logic Conference, FloC 2018, Oxford, UK, July 14-17, 2018, Proceedings*, volume 10900 of *Lecture Notes in Computer Science*, pages 566–574. Springer, 2018.
- [195] B. Piotrowski and J. Urban. Stateful premise selection by recurrent neural networks. In *LPAR*, volume 73 of *EPiC Series in Computing*, pages 409–422. EasyChair, 2020.
- [196] B. Piotrowski, J. Urban, C. E. Brown, and C. Kaliszyk. Can neural networks learn symbolic rewriting? *arXiv preprint arXiv:1911.04873*, 2019.
- [197] K. Pąk. Brouwer fixed point theorem in the general case. *Formalized Mathematics*, 19(3):151–153, 2011.

- [198] S. Prasad, R. Shankar, R. Gupta, and S. Roy. A tism modeling of critical success factors of blockchain based cloud services. *JOURNAL OF ADVANCES IN MANAGEMENT RESEARCH*, 15(4):434–456, 2018.
- [199] G. Priest, K. Tanaka, and Z. Weber. Paraconsistent Logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Spring 2022 edition, 2022.
- [200] Y. Qin, Z. Xu, X. Wang, and M. Skare. Artificial intelligence and economic development: An evolutionary investigation and systematic review. *JOURNAL OF THE KNOWLEDGE ECONOMY*.
- [201] A. Quaife. *Automated Development of Fundamental Mathematical Theories*. Kluwer Academic Publishers, 1992.
- [202] F. Rabe and M. Kohlhase. A scalable module system. *Information and Computation*, 230:1–54, 2013.
- [203] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, and I. Sutskever. Language models are unsupervised multitask learners. 2019.
- [204] M. Rawson. *Applications of machine learning to automated reasoning*. PhD thesis, University of Manchester, 2021.
- [205] M. Rawson and G. Reger. Dynamic strategy priority: Empower the strong and abandon the weak. In *PAAR@FLoC*, 2018.
- [206] M. Rawson and G. Reger. Old or heavy? decaying gracefully with age/weight shapes. In P. Fontaine, editor, *Automated Deduction – CADE 27*, pages 462–476, Cham, 2019. Springer International Publishing.
- [207] M. Rawson and G. Reger. lazycop: Lazy paramodulation meets neurally guided search. In *TABLEAUX*, volume 12842 of *Lecture Notes in Computer Science*, pages 187–199. Springer, 2021.
- [208] M. Rawson and G. Reger. Automated theorem proving, fast and slow. EasyChair Preprint no. 4433, EasyChair, 2021.
- [209] J. Reveley. Embracing the humanistic vision: Recurrent themes in peter roberts’ recent writings. *EDUCATIONAL PHILOSOPHY AND THEORY*, 50(3):312–321, 2018.
- [210] M. Richardson and P. Domingos. Markov logic networks. *Mach. Learn.*, 62(1–2):107–136, feb 2006.
- [211] N. Robertson, D. P. Sanders, P. Seymour, and R. Thomas. Efficiently four-coloring planar graphs. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC ’96, page 571–575, New York, NY, USA, 1996. Association for Computing Machinery.

- [212] J. A. Robinson and A. Voronkov, editors. *Handbook of Automated Reasoning (in 2 volumes)*. Elsevier and MIT Press, 2001.
- [213] T. Rocktäschel and S. Riedel. End-to-end differentiable proving. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.
- [214] P. Roshon and F.-J. Yang. A study on deep learning approach to optimize solving construction problems. In *2021 International Conference on Computational Science and Computational Intelligence (CSCI)*, pages 170–174, 2021.
- [215] T. Ross. *FUZZY LOGIC WITH ENGINEERING APPLICATIONS, 3RD ED.* Wiley India Pvt. Limited, 2011.
- [216] C. Ruhdorfer and S. Schulz. Efficient implementation of large-scale watchlists. In P. Fontaine, K. Korovin, I. S. Kotsireas, P. Rümmer, and S. Tourret, editors, *Joint Proceedings of the 7th Workshop on Practical Aspects of Automated Reasoning (PAAR) and the 5th Satisfiability Checking and Symbolic Computation Workshop (SC-Square) Workshop, 2020 co-located with the 10th International Joint Conference on Automated Reasoning (IJCAR 2020), Paris, France, June-July, 2020 (Virtual)*, volume 2752 of *CEUR Workshop Proceedings*, pages 120–133. CEUR-WS.org, 2020.
- [217] S. Schäfer and S. Schulz. Breeding theorem proving heuristics with genetic algorithms. In Gottlob et al. [104], pages 263–274.
- [218] J. Schmidhuber. Gödel machines: Fully self-referential optimal universal self-improvers. In B. Goertzel and C. Pennachin, editors, *Artificial General Intelligence*, Cognitive Technologies, pages 199–226. Springer, 2007.
- [219] S. Schulz. *Learning search control knowledge for equational deduction*, volume 230 of *DISKI*. Infix Akademische Verlagsgesellschaft, 2000.
- [220] S. Schulz. Learning Search Control Knowledge for Equational Theorem Proving. In F. Baader, G. Brewka, and T. Eiter, editors, *Proc. of the Joint German/Austrian Conference on Artificial Intelligence (KI-2001)*, volume 2174 of *LNAI*, pages 320–334. Springer, 2001.
- [221] S. Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.
- [222] S. Schulz. System description: E 1.8. In K. L. McMillan, A. Middeldorp, and A. Voronkov, editors, *LPAR*, volume 8312 of *LNCS*, pages 735–743. Springer, 2013.
- [223] S. Schulz, S. Cruanes, and P. Vukmirovic. Faster, higher, stronger: E 2.3. In P. 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*, pages 495–507. Springer, 2019.
- [224] S. Schulz and M. Möhrmann. Performance of clause selection heuristics for saturation-based theorem proving. In N. Olivetti and A. Tiwari, editors, *Proc. of the 8th IJCAR, Coimbra*, volume 9706 of *LNAI*, pages 330–345. Springer, 2016.
- [225] X. Shi, Q. Zhang, and Z. Zheng. The double-edged sword of external search in collaboration networks: embeddedness in knowledge networks as moderators. *JOURNAL OF KNOWLEDGE MANAGEMENT*, 23(10):2135–2160, DEC 9 2019.
- [226] D. Shin and W. T. Bianco. In blockchain we trust: Does blockchain itself generate trust? *SOCIAL SCIENCE QUARTERLY*, 101(7):2522–2538, DEC 2020.
- [227] D. Shin and Y. Hwang. The effects of security and traceability of blockchain on digital affordance. *ONLINE INFORMATION REVIEW*, 44(4):913–932, AUG 10 2020.
- [228] K. Slind and M. Norrish. A brief overview of HOL4. In O. A. Mohamed, C. A. Muñoz, and S. Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *LNCS*, pages 28–32. Springer, 2008.
- [229] N. J. A. Sloane. The on-line encyclopedia of integer sequences. In M. Kauers, M. Kerber, R. Miner, and W. Windsteiger, editors, *Towards Mechanized Mathematical Assistants*, pages 130–130, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [230] M. Suda. Improving enigma-style clause selection while learning from history. In A. Platzer and G. Sutcliffe, editors, *Automated Deduction – CADE 28*, pages 543–561, Cham, 2021. Springer International Publishing.
- [231] M. Suda. Vampire with a brain is a good itp hammer. In B. Konev and G. Reger, editors, *Frontiers of Combining Systems*, pages 192–209, Cham, 2021. Springer International Publishing.
- [232] G. Sutcliffe. The TPTP Problem Library and Associated Infrastructure. From CNF to TH0, TPTP v6.4.0. *Journal of Automated Reasoning*, 59(4):483–502, 2017.
- [233] G. Sutcliffe and Y. Puzis. SRASS - a semantic relevance axiom selection system. In F. Pfenning, editor, *CADE*, volume 4603 of *LNCS*, pages 295–310. Springer, 2007.

- [234] R. Thiemann and C. Sternagel. Certification of termination proofs using ceta. In S. Berghofer, T. Nipkow, C. Urban, and M. Wenzel, editors, *TPHOLs*, volume 5674 of *Lecture Notes in Computer Science*, pages 452–468. Springer, 2009.
- [235] A. Trybulec. Tarski Grothendieck set theory. *Formalized Mathematics*, 1(1):9–11, 1990.
- [236] F. Tuong and B. Wolff. Isabelle/c. *Archive of Formal Proofs*, October 2019. https://isa-afp.org/entries/Isabelle_C.html, Formal proof development.
- [237] J. Urban. MPTP 0.2: Design, implementation, and initial experiments. *J. Autom. Reasoning*, 37(1-2):21–43, 2006.
- [238] J. Urban. MaLAREa: a metasystem for automated reasoning in large theories. In G. Sutcliffe, J. Urban, and S. Schulz, editors, *ESARLT*, volume 257 of *CEUR Workshop Proceedings*. CEUR-WS.org, 2007.
- [239] J. Urban. BliStr: The Blind Strategymaker. In Gottlob et al. [104], pages 312–319.
- [240] J. Urban. ERC project AI4Reason final scientific report, 2021. http://grid01.ciirc.cvut.cz/~mptp/ai4reason/PR_CORE_SCIENTIFIC_4.pdf.
- [241] J. Urban and J. Jakubův. First neural conjecturing datasets and experiments. In C. Benzmüller and B. Miller, editors, *Intelligent Computer Mathematics*, pages 315–323, Cham, 2020. Springer International Publishing.
- [242] J. Urban and G. Sutcliffe. ATP cross-verification of the Mizar MPTP Challenge problems. In N. Dershowitz and A. Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning*, pages 546–560, Berlin, Heidelberg, 2007. Springer Berlin Heidelberg.
- [243] J. Urban, G. Sutcliffe, P. Pudlák, and J. Vyskočil. MaLAREa SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In A. Armando, P. Baumgartner, and G. Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008.
- [244] J. Urban, J. Vyskočil, and P. Štěpánek. MaLeCoP: Machine learning connection prover. In K. Brünner and G. Metcalfe, editors, *TABLEAUX*, volume 6793 of *LNCS*, pages 263–277. Springer, 2011.
- [245] J. Veness, K. S. Ng, M. Hutter, W. Uther, and D. Silver. A monte-carlo aixi approximation. *J. Artif. Int. Res.*, 40(1):95–142, jan 2011.
- [246] F. Verbeek, A. Bharadwaj, J. Bockenek, I. Roessle, T. Weerwag, and B. Ravindran. X86 instruction semantics and basic block symbolic execution. *Archive of Formal Proofs*, October 2021. https://isa-afp.org/entries/X86_Semantics.html, Formal proof development.

- [247] R. Veroff. Using hints to increase the effectiveness of an automated reasoning program: Case studies. *J. Autom. Reasoning*, 16(3):223–239, 1996.
- [248] P. Vukmirovic, J. Blanchette, S. Cruanes, and S. Schulz. Extending a brainiac prover to lambda-free higher-order logic. *Int. J. Softw. Tools Technol. Transf.*, 24(1):67–87, 2022.
- [249] J. Väänänen. Second-order and Higher-order Logic. In E. N. Zalta, editor, *The Stanford Encyclopedia of Philosophy*. Metaphysics Research Lab, Stanford University, Fall 2021 edition, 2021.
- [250] Q. Wang, C. Brown, C. Kaliszyk, and J. Urban. Exploration of neural machine translation in autoformalization of mathematics in mizar. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, page 85–98, New York, NY, USA, 2020. Association for Computing Machinery.
- [251] Q. Wang and C. Kaliszyk. Jeff: Joint embedding of formal proof libraries. In B. Konev and G. Reger, editors, *Frontiers of Combining Systems*, pages 154–170, Cham, 2021. Springer International Publishing.
- [252] C. Weidenbach. Combining superposition, sorts and splitting. In Robinson and Voronkov [212], pages 1965–2013.
- [253] R. Wilson. *Four Colors Suffice: How the Map Problem Was Solved-Revised Color Edition*, volume 30. Princeton university press, 2013.
- [254] Y. Wu, A. Q. Jiang, W. Li, M. N. Rabe, C. Staats, M. Jamnik, and C. Szegedy. Autoformalization with large language models, 2022.
- [255] Y. Wu, M. N. Rabe, W. Li, J. Ba, R. B. Grosse, and C. Szegedy. Lime: Learning inductive bias for primitives of mathematical reasoning. In M. Meila and T. Zhang, editors, *Proceedings of the 38th International Conference on Machine Learning*, volume 139 of *Proceedings of Machine Learning Research*, pages 11251–11262. PMLR, 18–24 Jul 2021.
- [256] A. Yamada, K. Kusakari, and T. Sakabe. A unified ordering for termination proving. *Science of Computer Programming*, 111:110–134, 2015. Special Issue on Principles and Practice of Declarative Programming (PPDP 2013).
- [257] L. Zhang, L. Blaauwbroek, B. Piotrowski, P. Černý, C. Kaliszyk, and J. Urban. Online machine learning techniques for Coq: A comparison. In F. Kamareddine and C. Sacerdoti Coen, editors, *Intelligent Computer Mathematics*, pages 67–83, Cham, 2021. Springer International Publishing.
- [258] K. Zheng, J. M. Han, and S. Polu. Minif2f: a cross-system benchmark for formal olympiad-level mathematics. *arXiv preprint arXiv:2109.00110*, 2021.

- [259] Z. Zombori, J. Urban, and C. E. Brown. Prolog technology reinforcement learning prover. In N. Peltier and V. Sofronie-Stokkermans, editors, *Automated Reasoning*, pages 489–507, Cham, 2020. Springer International Publishing.
- [260] Z. Zombori, J. Urban, and M. Olsák. The role of entropy in guiding a connection prover. In *TABLEAUX*, volume 12842 of *Lecture Notes in Computer Science*, pages 218–235. Springer, 2021.