EPiC Series in Computing
Volume 73, 2020, Pages 138–150
LPAR23. LPAR-23: 23rd International Conference on Logic for Programming, Artificial Intelligence and Reasoning

Tactic Learning and Proving for the Coq Proof Assistant*

Lasse Blaauwbroek[1,2], Josef Urban[1], and Herman Geuvers[2]

[1] Czech Technical University, Prague, Czech Republic
[2] Radboud University, Nijmegen, The Netherlands

lasse@blaauwbroek.eu, josef.urban@gmail.com, herman@cs.ru.nl

Abstract

We present a system that utilizes machine learning for tactic proof search in the Coq Proof Assistant. In a similar vein as the TacticToe project for HOL4, our system predicts appropriate tactics and finds proofs in the form of tactic scripts. To do this, it learns from previous tactic scripts and how they are applied to proof states. The performance of the system is evaluated on the Coq Standard Library. Currently, our predictor can identify the correct tactic to be applied to a proof state 23.4% of the time. Our proof searcher can fully automatically prove 39.3% of the lemmas. When combined with the CoqHammer system, the two systems together prove 56.7% of the library's lemmas.

1 Introduction

The Coq Proof Assistant [37] is an Interactive Theorem Prover in which one proves lemmas using a tactic language. This language contains a wide range of tactics, from simple actions like `intros` and `apply` to complicated decision procedures such as `ring` and `tauto` and search procedures like `auto` and `firstorder`. Although the second and third options provide a large amount of automation, they still require human intervention to use. (1) Decision procedures only apply to particular domains, requiring the user to know when they are appropriate, and (2) Coq's search tactics require careful construction of hint databases to be performant.

The system we present in this paper learns from previously written proof scripts and how they are applied to proof states. This knowledge can then be used to suggest a tactic to apply on a previously unseen proof state or to perform a full proof search and prove the current lemma automatically. Learning on the tactic level has some advantages over learning from low-level proof terms in Gallina, Coq's version of the Calculus of Inductive Constructions [31]: (1) Tactics represent coarser proof steps than the individual identifiers in a proof term. They are also more forgiving; a minor modification to a tactic script is more likely to be nondestructive than a minor modification to a proof term, making the machine learning task easier. (2) By learning on the tactic level, we allow the user to introduce domain-specific knowledge to the system by writing custom tactics. By using these tactics in hand-written proofs, the system will automatically learn of their existence and start to use them.

Similar to the TacticToe project for HOL4 [11], our system has the following components.

*This work was supported by the *AI4REASON* ERC Consolidator grant nr. 649043 and by the European Regional Development Fund under the project AI&Reasoning (reg. no. CZ.02.1.01/0.0/0.0/15_003/0000466)

Proof recording First, we create a database of tactics that are executed in existing (human-written) tactic scripts, recording in which proof states they are executed. We do not constrain the type of tactic that can be recorded, which allows the system to discover user-defined tactics dynamically.

Tactic prediction Using this database of pairs of tactics and proof states, we use machine learning to generate a list of tactics likely applicable to the current proof state. This list can either be used as a tactic recommendation system for the user, or as input to the next component.

Proof search Finally, for a given proof state, we perform a proof search by repeatedly applying predicted tactics to the proof state and subsequent states in the search tree until all proof obligations are discharged.

These components are all implemented in OCaml (the implementation language of Coq)—without external dependencies—and integrated directly with Coq, enabling us to eventually create a user-facing tool that is easy to use, install, and maintain.

In this paper, we present the technical implementation of the components above and an evaluation of the quality of tactic predictions and proof search. In Sections 2 to 4, we take a deeper dive into the recording, prediction, and search components of our system. Section 5 gives an overview of related work. Finally, Section 6 contains an evaluation of our system on the Coq Standard Library. The dataset used for our evaluation, together with the software that generated it is publicly available [4].

2 Proof Recording

The first component of the system is the recording of existing proofs. As said, this is done on the level of tactics. Conceptually speaking, when a tactic script is executed, we record the proof state before and after the execution of each tactic. The difference between these two states represents the action performed by the tactic, while the state before the tactic represents the context in which it was useful. By recording many such instances for a tactic, we create a dataset representing an approximation of the semantic meaning of that tactic. In practice, we currently only record the proof state before the execution of a tactic because, during proof search, we do not yet evaluate the proof state's quality after the execution of a tactic.

An important question is what exactly constitutes a tactic in Coq. On a low level, Coq utilizes a proof monad in which tactics can be written on the OCaml level [25]. This monad supports backtracking, among other things. It is, however, not immediately accessible by end-users. For that, several tactic languages exist that are interpreted into the proof monad [20, 28, 14, 32]. The most used language is Ltac1 [10]. In an ideal world, we would record tactics on the level of the proof monad since that would allow us to record tactics from all existing tactic languages. However, it does not appear to be possible to put recording instrumentation in place on the monadic level. Therefore, we have chosen to only record tactics from the Ltac1 language.

Within the Ltac language, it is also not immediately clear what a tactic is. One option is to decompose a script into a series of the most primitive tactic invocations and record those. On the other side of the extreme, one could view every whole vernacular expression as one tactic. The first option greatly diminishes the advantages of the system because such low-level recording would not allow domain-specific decision procedures or custom tactics to be recorded. The second option means that many tactics will be unique and rather specific to a single proof state. A good solution will lie somewhere in between. Striking this balance is further complicated by

the possibility of tactics to backtrack. It is unclear whether we should record only the successful trace of a tactic, or keep the backtracking procedure as one building block.

At the moment, we see every vernacular command as one tactic, except for tactic composition (`tac1; tac2`) and tactic dispatching (`tac1; [tac2 | tac3]`). We treat tactics and their arguments as one unit. The lack of global lemma prediction for tactic arguments is, in our opinion, not of crucial importance in Coq due to the nature of its tactics. Tactics in Coq usually accept a single lemma as an argument, rather than a list of premises like the `metis` tactic in HOL4. Tactics that do accept multiple lemmas as arguments are usually a shorthand that invokes a simpler tactic for every lemma. The semantics of tactics that accept a single lemma as an argument, such as `apply lem` and `rewrite lem`, are mainly determined by the given lemma. Therefore, it makes sense to predict these tactics and their arguments as one unit. However, as a consequence, new lemmas will only be picked up by the system when it learns from a tactic that mentions those lemmas. Finally, we acknowledge that the prediction of arguments pointing to the local context is important. This is left as future work.

To record a tactic script, conceptually, we replace a tactic `t` with a custom recording tactic `r t` that receives the original tactic as an argument. Tactic `r t` first records the current proof state before executing `t`. Then this proof state is added to the database together with tactic `t`. As an example of a tactic decomposition and recording, the tactic expression `tac1; [tac2 | tac3]; tac4` will be converted to `r tac1; [r tac2 | r tac3]; r tac4`.

3 Tactic Prediction

After creating a database of tactics and recorded proof states, we want to predict the correct tactic to make progress in a new, unseen proof state. For this, we must find a good characterization of the sentences present in a proof state. Currently, this characterization is a set of features. The features of a sentence are all of its one-shingles and two-shingles, meaning they are all identifiers and pairs of adjacent identifiers in the abstract syntax tree. For example, the sentence $(x + y) + z = x + (y + z)$ will be converted to the feature set

$$\{eq, plus, x, y, z, eq\_plus, plus\_x, plus\_y, plus\_z, eq\_z, eq\_x\}$$

modulo the fact that variables are represented by de Bruijn indices. For the characterization of a proof state, we take the union of the feature set of all sentences in its hypotheses and the goal.

To find a list of likely matches for a new proof state, we run a k-nearest neighbor algorithm on the vectors. We compare vectors using several standard metrics and similarities (specialized for feature sets instead of feature vectors):

$$\text{cosine}(f_1, f_2) = \frac{|f_1 \cap f_2|}{\sqrt{|f_1||f_2|}} \quad \text{euclid}(f_1, f_2) = \sqrt{|f_1 \cup f_2| - |f_1 \cap f_2|} \quad \text{jaccard}(f_1, f_2) = \frac{|f_1 \cap f_2|}{|f_1 \cup f_2|}$$

Finally, we would like to weigh features based on their relative importance in a given proof state with respect to the entire database. For this, we use the TfIdf statistic [19] (which is the same as the Idf statistic since we use feature sets rather than bags). We then use a generalized version of the Jaccard index to incorporate these weights into the predictions.

$$\text{tfidf}(x) = \log \frac{N}{|x|_N} \qquad \text{jaccard}_w(f_1, f_2) = \frac{\sum_{x \in f_1 \cap f_2} \text{tfidf}(x)}{\sum_{x \in f_1 \cup f2} \text{tfidf}(x)}$$

Here N is the database size, and $|x|_N$ is the number of times feature x occurs in the database.

Our k-nearest neighbor algorithm outputs an ordered list l of pairs $\langle(d_i, t_i)\rangle_k$, where t_i is a tactic and d_i the similarity between the current proof state and the proof state on which t_i was applied. If l contains duplicate tactics, we select only the pair with the best score. The list is ordered by the similarities d and can, therefore, immediately be used as a list of tactic recommendations.

A major downside of the k-NN approach above is that it requires an exhaustive search of the entire tactic database to obtain a list of recommendations. Therefore, the time complexities of the algorithms grow linearly with the size of the database, which can be quite large. To eliminate slow queries, we include an approximate k-NN algorithm that employs Locally Sensitive Hashing [12]. LSH is a technique that uses hashing functions that map similar items to the same integer with high probability (as opposed to the normal hashing function in which collisions are avoided). These functions are then used with hashtables to map similar items to the same bucket, at which point they can be retrieved in constant time. As long as the notion of similarity of the hashing function agrees with the intended similarity, this can be used as a substitute for the corresponding exhaustive k-NN algorithm. The downside of this method is that it is approximate and can thus miss crucial tactics. In our implementation, we use a technique called LSH Forest [3] to approximate the Jaccard index described above.

4 Proof Search

It is unlikely that the tactic prediction system will predict the correct tactic every time. For this reason, a proof search must be performed, where not only the first predicted tactic is chosen. We recursively explore the proof tree by executing a predicted tactic and then predicting a new list of tactics based on the new proof state. We explore this tree using a skewed form of breadth-first search, which we call diagonal search. Given a list of predictions, $\langle(d_i, t_i)\rangle_k$, we always explore the subtree starting from tactic t_i one step deeper than the subtree starting from tactic t_{i+1}. This diagonal exploration is continued recursively. If a tactic fails or does not modify the proof state, the resulting subtree is pruned.

Diagonal search captures the idea that tactics that are predicted as better should be explored more vigorously than other tactics. In the future, we intend to refine this search style by using the similarity scores d_i to determine how "diagonal" the search should be. Another interesting approach is to take inspiration from AlphaGo Zero [35] and use a Monte Carlo Tree Search algorithm for proof search. We should note, however, that such strategies have not born much fruit for the TacticToe system of HOL4.

5 Related Work

The high level of integration of our system with Coq and its lack of external dependencies is a distinguishing feature compared to existing learning-guided systems for Coq. The ML4PG system [26] provides tactic predictions and other statistics but is instead integrated with the Proof General [1] proof editor and requires connections to Matlab or Weka to function. The SEPIA system [15] provides proof search using tactic predictions and is also integrated with Proof General. Note, however, that its proof search is only based on tactic traces and does not make predictions based on the proof state.

Efforts to use neural architectures for Coq have also been made. GamePad [16] is a framework that integrates with the Coq codebase and allows machine learning to be performed in Python. Currently, the platform supports proof search and position evaluation (estimating the number of

steps required to finish a proof). A limited proof search evaluation was performed on a synthetic algebraic rewriting dataset. CoqGym [40] also extracts tactic and proof state information on a large scale and uses it to construct a deep learning model capable of generating tactics and their arguments. Although CoqGym's approach is interesting, it performs significantly worse than CoqHammer (introduced below) while using a much higher, impractical time limit. Another downside is that tactics are generated using a fixed grammar, which precludes the system from learning custom, domain-specific tactics as our system does.

The main inspiration for our system is the TacticToe [11] system for HOL4 [36]. Our work is similar both in doing a learning-guided tactic search and by its complete integration in the underlying proof assistant without relying on external machine learning libraries and specialized hardware. More recent TacticToe-inspired systems that do not aim at complete independence of external learning toolkits include HOList [2] and similar work by Wu et al. for HOL4 [39]. Although rather similar to TacticToe on the surface, differences in the logic of HOL4 and Coq and the design of their tactic languages result in significant differences in implementation details. For example, premise selection is important in HOL4 for automation tactics that require lists of premises to be provided. As explained in Section 2, this is much less important for our system. Coq's automation tactics generally rely on hint databases instead of lists of premises. On the other hand, a complicating feature of Coq is the notion of a (named) local context, while this does not exist in HOL4. Finally, the technical implementation of proof recording and search is quite different because Coq has a proof monad, while tactics in HOL4 are written directly in Standard ML.

Significant work on learning-based guidance has been done in the context of connection-based and saturation-based Automated Theorem Provers (ATPs) such as leanCoP [30] and E [34]. In the leanCoP setting, (reinforcement) learning of clausal steps based on the proof state is done in systems such as rlCoP [24, 29], and (FE)MaLeCoP [22, 38]. In E, given-clause selection has been guided by learning in systems such as ENIGMA and its variants [17, 27, 18, 7, 13].

ATPs are also used in *hammers* [6, 21, 5, 23], where learning-based premise selection is used to translate an ITP problem to the ATP's target language (typically First Order Logic). A proof found by the ATP can then be reconstructed in the proof assistant. A particular hammer instance for Coq is the CoqHammer system [8].

6 Evaluation

We use Coq's standard library to evaluate the effectiveness of our system. The standard library is an impartial choice that should represent the de facto proof script style (although we acknowledge that many proof styles are available to users). The library contains modules with a variety of complexity. Basic modules regarding logic and arithmetic contain relatively simple proofs, while more complicated developments like the reals are also included. Evaluating the standard library also allows us to compare our performance to CoqHammer. The dataset for the evaluation and the software that generated it is publicly available [4].

6.1 Methodology

For our evaluation, we try to simulate how a hypothetical user might develop the standard library. We assume that when the user tries to prove a new lemma, all previous lemmas are already proven and can be used for learning. Hence, when evaluating a Coq source code file, the tactic database starts with only information inherited from the file's dependencies, if any. When we evaluate a lemma in the file, we are only allowed to use state-tactic pairs from dependency

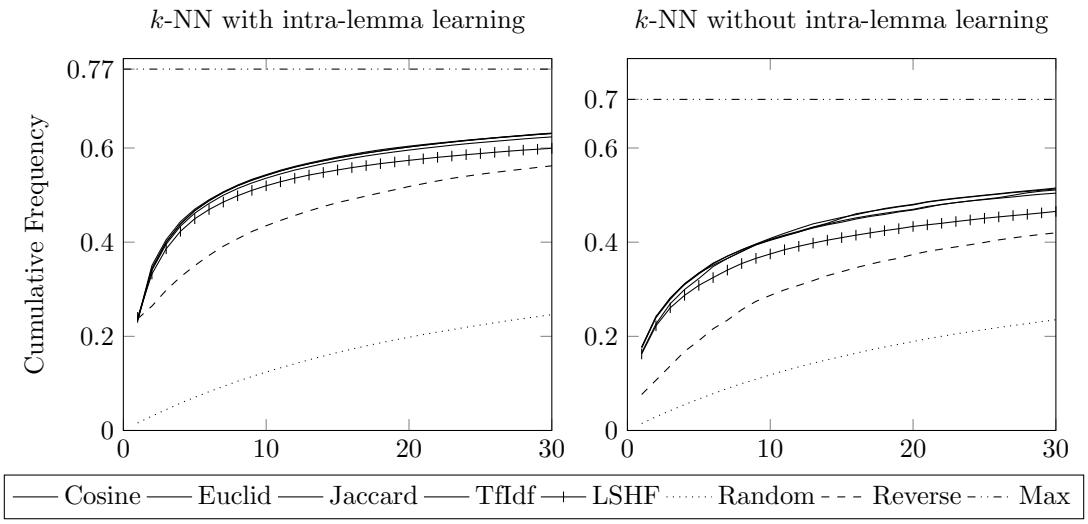

Figure 1: Evaluation of different metrics used for k-nearest neighbor prediction. For every k, the proportion of state-tactic pairs for which the correct tactic is in the candidate list is plotted.

files and pairs recorded in the file's previous lemmas. In contrast, Hammer evaluations typically use the less faithful approach of sorting the files topologically and letting all previous proofs (not just those in the dependencies) inform the current search, giving them more learning data.

We start with an offline evaluation of the feature quality and tactic prediction quality. Then follows a full evaluation of the search strategy and a comparison to CoqHammer. The standard library of Coq v8.10 is used in the evaluations. All the experiments were run on a 32 cores Intel(R) Xeon(R) CPU E5-2698 v3 @ 2.30GHz server with 256 GB memory.

6.2 Tactic Prediction Evaluation

The quality of the tactic prediction directly influences the ability to synthesize entire proofs. As the tactic predictions improve, the proof search procedure will require less backtracking. To judge the prediction quality, we perform an offline evaluation. We have extracted a textual representation of the tactic database of every file. For every pair of proof state s and tactic t in a file, we use the previous pairs in that file and its dependencies to predict an ordered list of candidate tactics. We judge this prediction by how close to the top of the candidate list t occurs. Note that this offline evaluation is an under-approximation of the system's real predictive power. This is because we judge the system by its ability to predict exactly the original tactic t, while there might also be alternative tactics that perform the same or better than the original.

An interesting question is whether a prediction may use tactics that have been recorded in the lemma currently being proved. If so, the system could learn from a proof while the user writes it, letting the user benefit from this within the same proof. A problem with this is that the proof states within the same lemma are likely to be very similar, especially when using our feature characterization. Therefore, if we allow intra-lemma learning, we can expect to see the tactics from the current lemma predicted often.

We evaluate the different metrics we proposed for the k-nearest neighbor algorithm with and without the intra-lemma learning described above. The results are shown in Figure 1. It turns out that all our metrics perform quite similarly. Contrary to prior expectations, even the

TfIdf-weighted version of the Jaccard measure does not improve the predictions. To evaluate the level of locality enjoyed by tactics, we also include the "reverse-database" predictor, where we rank tactics by when they were last added to the database. When we allow intra-lemma learning, reverse-database does not perform much worse than the other predictors, suggesting that predictors frequently output the previous tactic as their best choice. Obviously, this will seldom be the correct tactic. A future line of research is to incorporate the local tactic execution history (*m*emory) into the predictors to combat this problem. However, with a larger k, the reverse-predictor's performance starts to go down, showing that our real predictors do nontrivial learning. Finally, the LSHF predictor is performing only about 15% worse than the rest. Given that it performs predictions in constant time, allowing for approximately a thousand predictions per second, this is very promising. Especially considering that the file with the largest database contains 90594 tactics, where an exhaustive prediction takes hundreds of milliseconds.

The main result of this evaluation is that the first predicted tactic is the correct one 23.4% of the time with intra-lemma learning and 17.7% without intra-lemma learning. This number goes up as we start to consider tactics that are predicted with lower similarity. For $k = 10$, the correct tactic is among the predicted tactics 54.3% and 40.3% of the time, respectively. These results are encouraging, considering that the theoretical best we can achieve is 77%. This limit exists because whenever a tactic is encountered for the first time, it does not yet exist in the tactic database. Therefore it is impossible for the system to give a correct prediction.

6.3 Proof Search Evaluation

For a full evaluation of the system, we want to know for how many of the lemmas in the library a proof can be synthesized automatically. These experiments are performed as follows. Every Coq source code file is compiled as usual. However, when the system encounters a lemma, it disregards the original proof and tries to synthesize a new proof based on the current tactic database. It then records whether the proof synthesis was successful and substitutes the original proof back, thus ensuring that it can progress through the entire file in case no proof is found. The tactics used in the original proofs are then added to the tactic database. Each proof search is given 40 seconds, equal to the total time CoqHammer used in its evaluation.

We experiment with several configurations of our system. Exhaustive k-NN search is performed using the TfIdf-weighted Jaccard measure. To keep the search speed under control, we limit the size of the tactic database. First, we evaluate with only tactics that have executed in the current file (no tactics from dependencies). Then we evaluate using only the last 1000, 2000 and 10000 recorded tactics at the moment a proof search is initiated. For completeness, we also evaluate using all available tactics. Finally, an evaluation using the approximate Jaccard predictor LSHF is performed.

Table 1 gives an overview of the performance of these evaluations, broken down for the different developments of the standard library. The different configurations can prove between 28.4% and 34.0% of the library lemmas, with the LSHF configuration being the best. Together, the configurations prove 39.3% of the library lemmas. From this, it is clear that the most crucial factor is the speed at which the tactic predictions can be made. The faster the predictions, the deeper the search tree can be explored. Since LSHF's speed is in the order of a thousand predictions per second when using the full tactic database, it defeats the brute force k-NN approaches despite missing some tactics due to its approximate nature. For the exhaustive configurations, the versions with smaller databases work the best. Therefore, we conclude that tactics exhibit good locality. That is, tactics that were recently executed are likely to be useful again. It should be noted that the exhaustive configuration with a full database still proves

		Proof Length			Exhaustive k-NN with db size n						
Development	#Lem	Q_2	Q_3	Max	File	1000	2000	10000	All	LSHF	Total
Arith	293	2	5	42	38.2	48.8	52.2	51.2	51.2	61.8	67.2
Bool	130	1	3	8	83.8	93.1	92.3	92.3	93.1	93.8	93.8
Classes	191	1	4	19	72.3	79.6	78.0	78.5	80.1	80.1	82.7
FSets	1,137	6	15	471	27.6	29.6	28.9	28.7	27.7	32.5	36.2
Init	166	3	6	257	62.0	71.1	71.7	71.7	71.7	68.7	72.9
Lists	388	8	15	74	36.1	36.6	35.8	34.0	34.3	40.5	43.8
Logic	329	4	9	121	24.3	29.8	29.5	29.8	29.5	33.7	35.0
MSets	830	5	15	164	34.7	36.1	34.6	34.3	33.5	36.9	42.3
NArith	288	5	10	54	34.7	36.1	37.8	34.7	34.4	38.5	45.5
Numbers	2,198	5	11	275	14.6	16.1	16.3	15.7	15.5	16.9	20.1
PArith	280	4	9	166	29.6	31.8	32.1	32.5	31.1	30.4	36.8
Program	28	2	5	11	42.9	82.1	78.6	78.6	78.6	71.4	82.1
QArith	295	5	9	80	31.5	34.6	39.3	34.6	33.2	38.3	47.5
Reals	1,975	8	21	1,121	23.2	24.3	23.4	21.0	18.3	25.0	31.4
Relations	37	8	11	25	32.4	32.4	29.7	29.7	29.7	32.4	37.8
Setoids	4	4	5	5	0.0	100.0	100.0	100.0	100.0	100.0	100.0
Sets	222	6	12	93	33.3	41.0	40.5	38.7	39.2	48.2	51.4
Sorting	136	8	15	100	23.5	25.7	24.3	21.3	19.1	24.3	32.4
Strings	74	9	117	1,539	13.5	17.6	18.9	27.0	24.3	21.6	29.7
Structures	390	3	7	71	33.3	40.8	39.2	39.7	39.2	50.0	54.1
Vectors	37	7	12	22	21.6	18.9	24.3	18.9	18.9	37.8	43.2
Wellfounded	36	8	22	49	8.3	19.4	11.1	11.1	11.1	16.7	19.4
ZArith	952	3	7	87	35.4	46.5	47.5	45.3	45.0	47.6	58.0
Total	10,416	5	12	1,539	28.4	32.0	31.9	30.7	29.9	34.0	39.3

Table 1: A breakdown of the performance of some configurations of the system against the developments in the standard library, measured by the percentage of lemmas automatically proved. This culminates in the Total column, where the performance of the union of all configurations is measured. The Proof Length columns summarize the original proof lengths, where Q_2 and Q_3 are the second and third quartiles.

lemmas that are not proven by any other configuration. In those cases, its slow proof exploration is compensated by the fact that it can find exactly the right tactic.

When looking at the different developments in the standard library, we see substantial differences in performance, mostly due to differences in proof styles, with longer and shorter proofs. For example, in `Arith`, we do quite well due to short proofs facilitated by existing automation, such as `auto`. The most problematic development is `Numbers` due to its size and low success rate. Although its proofs are not absurdly big long, they utilize complicated compound tactics that are difficult for the system to decompose.

Figure 2 compares the lengths of the proofs found by the LSHF configuration with the lengths of the original proofs, measured in the number of individual tactics executed. The system easily finds most of the short proofs. Longer proofs are increasingly rare. However, a proof that utilizes 19 individual tactics was found. One striking thing to note is that the standard library contains quite a few extremely long proofs (the graph cuts off at length 80, but a proof of length 1539 exists). These long proofs are not written manually by the user but are the result of tactic sharing. An excellent example of this is the `ascii` inductive datatype. This

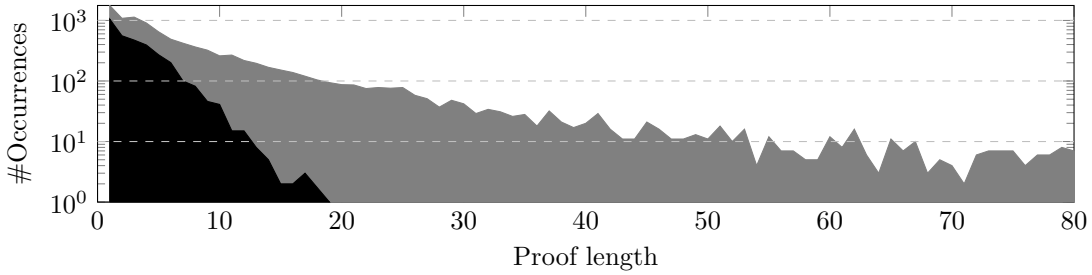

Figure 2: Logarithmic occurrence count of successfully synthesized proof lengths (black) vs. the original proof lengths in the library (grey), cut off at length 80.

type contains the obvious 256 cases. Proving a fact about `ascii` using case analysis requires one to prove all 256 cases individually. All cases can usually be proved using the same tactic, for example, `auto`. The human written proof then looks like `destruct c; auto`, while the system records 257 individual tactic executions. Making use of such tactic sharing is future work related to the issue of tactic recording granularity discussed in Section 2.

6.4 Comparison to CoqHammer

CoqHammer utilizes several automatic theorem provers, such as Vampire [33], E [34] and Z3 [9] with different settings and reconstruction tactics to generate proofs. Individually, these strategies are reported to prove between 17.5% and 28.8% of the Coq v8.5 standard library [8]. Combining all strategies leads to solving 40.8% library lemmas in total. Our individual configurations—and especially the LSHF configuration that solves 34.0% of the lemmas—are generally better than CoqHammer's strategies, while the hammer narrowly beats us in overall coverage.

More importantly, our system is quite complementary to CoqHammer. That is, we can prove many different lemmas. A direct comparison between the two systems is somewhat problematic due to the use of different versions of Coq's libraries and the technical difficulty of rerunning CoqHammer's original evaluation on the newer library. The v8.5 library used for the CoqHammer evaluation has 9276 lemmas, while the later v8.10 library used by us contains 10416 lemmas. Therefore, we compare only lemmas that exist in both versions. A straightforward method for aligning the lemmas in the two library versions yields 8231 common lemmas. Of those, 3240 (39.4%) are solved by the union of our configurations, and 3534 (42.9%) by the union of the CoqHammer strategies. The union of both systems yields 4666 (56.7%) proved lemmas, adding 32.0% more solutions to those obtained by CoqHammer. These results are a substantial improvement for Coq users who can employ both methods and then expect over 55% automation.

7 Conclusion

We have shown that machine learning for tactics is a useful enterprise. On its own, our system can fully automatically prove 39.3% of the standard library. Together with CoqHammer, 56.7% of the lemmas are proved, showing that these approaches complement each other very well. Although the system is already quite strong, many avenues of improvement exist. One can think of local parameter prediction, incorporating local tactic history in predictions, implementing better search strategies, and a proper user interface. We leave this as future work.

References

[1] David Aspinall. Proof General: A generic tool for proof development. In Susanne Graf and Michael I. Schwartzbach, editors, *Tools and Algorithms for Construction and Analysis of Systems, 6th International Conference, TACAS 2000, Held as Part of the European Joint Conferences on the Theory and Practice of Software, ETAPS 2000, Berlin, Germany, March 25 - April 2, 2000, Proceedings*, volume 1785 of *Lecture Notes in Computer Science*, pages 38–42. Springer, 2000. URL: https://doi.org/10.1007/3-540-46419-0_3, doi:10.1007/3-540-46419-0_3.

[2] Kshitij Bansal, Sarah M. Loos, Markus N. Rabe, Christian Szegedy, and Stewart Wilcox. HOList: An environment for machine learning of higher order logic theorem proving. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 454–463. PMLR, 2019. URL: http://proceedings.mlr.press/v97/bansal19a.html.

[3] Mayank Bawa, Tyson Condie, and Prasanna Ganesan. LSH forest: self-tuning indexes for similarity search. In Allan Ellis and Tatsuya Hagino, editors, *Proceedings of the 14th international conference on World Wide Web, WWW 2005, Chiba, Japan, May 10-14, 2005*, pages 651–660. ACM, 2005. URL: https://doi.org/10.1145/1060745.1060840, doi:10.1145/1060745.1060840.

[4] Lasse Blaauwbroek, Josef Urban, and Herman Geuvers. Artifacts of tactic learning and proving for the Coq proof assistant, March 2020. URL: https://doi.org/10.5281/zenodo.3693760, doi:10.5281/zenodo.3693760.

[5] Jasmin Christian Blanchette, David Greenaway, Cezary Kaliszyk, Daniel Kühlwein, and Josef Urban. A learning-based fact selector for Isabelle/HOL. *J. Autom. Reasoning*, 57(3):219–244, 2016. URL: https://doi.org/10.1007/s10817-016-9362-8, doi:10.1007/s10817-016-9362-8.

[6] Jasmin Christian Blanchette, Cezary Kaliszyk, Lawrence C. Paulson, and Josef Urban. Hammering towards QED. *J. Formalized Reasoning*, 9(1):101–148, 2016. URL: https://doi.org/10.6092/issn.1972-5787/4593, doi:10.6092/issn.1972-5787/4593.

[7] Karel Chvalovský, Jan Jakubuv, 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. URL: https://doi.org/10.1007/978-3-030-29436-6_12, doi:10.1007/978-3-030-29436-6_12.

[8] Lukasz Czajka and Cezary Kaliszyk. Hammer for Coq: Automation for dependent type theory. *J. Autom. Reasoning*, 61(1-4):423–453, 2018. URL: https://doi.org/10.1007/s10817-018-9458-4, doi:10.1007/s10817-018-9458-4.

[9] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An efficient SMT solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *Tools and Algorithms for the Construction and Analysis of Systems, 14th International Conference, TACAS 2008, Held as Part of the Joint European Conferences on Theory and Practice of Software, ETAPS 2008, Budapest, Hungary, March 29-April 6, 2008. Proceedings*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-78800-3_24, doi:10.1007/978-3-540-78800-3_24.

[10] David Delahaye. A tactic language for the system Coq. In Michel Parigot and Andrei Voronkov, editors, *Logic for Programming and Automated Reasoning, 7th International Conference, LPAR 2000, Reunion Island, France, November 11-12, 2000, Proceedings*, volume 1955 of *Lecture Notes in Computer Science*, pages 85–95. Springer, 2000. URL: https://doi.org/10.1007/3-540-44404-1_7, doi:10.1007/3-540-44404-1_7.

[11] Thibault Gauthier, Cezary Kaliszyk, and Josef Urban. TacticToe: Learning to reason with HOL4 tactics. In Thomas Eiter and David 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. doi:10.29007/ntlb.

[12] Aristides Gionis, Piotr Indyk, and Rajeev Motwani. Similarity search in high dimensions via

hashing. In Malcolm P. Atkinson, Maria E. Orlowska, Patrick Valduriez, Stanley B. Zdonik, and Michael L. Brodie, editors, *VLDB'99, Proceedings of 25th International Conference on Very Large Data Bases, September 7-10, 1999, Edinburgh, Scotland, UK*, pages 518–529. Morgan Kaufmann, 1999. URL: http://www.vldb.org/conf/1999/P49.pdf.

[13] Zarathustra Goertzel, Jan Jakubuv, 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, TABLEAUX 2019, London, UK, September 3-5, 2019, Proceedings*, volume 11714 of *Lecture Notes in Computer Science*, pages 374–388. Springer, 2019. URL: https://doi.org/10.1007/978-3-030-29026-9_21, doi:10.1007/978-3-030-29026-9_21.

[14] Georges Gonthier and Assia Mahboubi. An introduction to small scale reflection in Coq. *J. Formalized Reasoning*, 3(2):95–152, 2010. URL: https://doi.org/10.6092/issn.1972-5787/1979, doi:10.6092/issn.1972-5787/1979.

[15] Thomas Gransden, Neil Walkinshaw, and Rajeev Raman. SEPIA: Search for proofs using inferred automata. In Amy P. Felty and Aart Middeldorp, editors, *Automated Deduction - CADE-25 - 25th International Conference on Automated Deduction, Berlin, Germany, August 1-7, 2015, Proceedings*, volume 9195 of *Lecture Notes in Computer Science*, pages 246–255. Springer, 2015. URL: https://doi.org/10.1007/978-3-319-21401-6_16, doi:10.1007/978-3-319-21401-6_16.

[16] Daniel Huang, Prafulla Dhariwal, Dawn Song, and Ilya Sutskever. GamePad: A learning environment for theorem proving. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net, 2019. URL: https://openreview.net/forum?id=r1xwKoR9Y7.

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

[18] Jan Jakubuv 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. URL: https://doi.org/10.1007/978-3-319-96812-4_11, doi:10.1007/978-3-319-96812-4_11.

[19] Karen Spärck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 60(5):493–502, 2004. URL: https://doi.org/10.1108/00220410410560573, doi:10.1108/00220410410560573.

[20] Jan-Oliver Kaiser, Beta Ziliani, Robbert Krebbers, Yann Régis-Gianas, and Derek Dreyer. Mtac2: typed tactics for backward reasoning in Coq. *PACMPL*, 2(ICFP):78:1–78:31, 2018. URL: https://doi.org/10.1145/3236773, doi:10.1145/3236773.

[21] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *J. Autom. Reasoning*, 53(2):173–213, 2014. URL: https://doi.org/10.1007/s10817-014-9303-3, doi:10.1007/s10817-014-9303-3.

[22] 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. URL: https://doi.org/10.1007/978-3-662-48899-7_7, doi:10.1007/978-3-662-48899-7_7.

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

[24] Cezary Kaliszyk, Josef Urban, Henryk Michalewski, and Miroslav Olsák. Reinforcement learning of theorem proving. In Samy Bengio, Hanna M. Wallach, Hugo Larochelle, Kristen Grauman, Nicolò Cesa-Bianchi, and Roman Garnett, editors, *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.

[25] Florent Kirchner and César A. Muñoz. The proof monad. *J. Log. Algebr. Program.*, 79(3-5):264–277, 2010. URL: https://doi.org/10.1016/j.jlap.2010.03.002, doi:10.1016/j.jlap.2010.03.002.

[26] Ekaterina Komendantskaya, Jónathan Heras, and Gudmund Grov. Machine learning in Proof General: Interfacing interfaces. In Cezary Kaliszyk and Christoph Lüth, editors, *Proceedings 10th International Workshop On User Interfaces for Theorem Provers, UITP 2012, Bremen, Germany, July 11th, 2012*, volume 118 of *EPTCS*, pages 15–41, 2012. URL: https://doi.org/10.4204/EPTCS.118.2, doi:10.4204/EPTCS.118.2.

[27] Sarah M. Loos, Geoffrey Irving, Christian Szegedy, and Cezary Kaliszyk. Deep network guided proof search. In Thomas Eiter and David 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 85–105. EasyChair, 2017. doi:10.29007/8mwc.

[28] Gregory Malecha and Jesper Bengtson. Extensible and efficient automation through reflective tactics. In Peter Thiemann, editor, *Programming Languages and Systems - 25th European Symposium on Programming, ESOP 2016, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2016, Eindhoven, The Netherlands, April 2-8, 2016, Proceedings*, volume 9632 of *Lecture Notes in Computer Science*, pages 532–559. Springer, 2016. URL: https://doi.org/10.1007/978-3-662-49498-1_21, doi:10.1007/978-3-662-49498-1_21.

[29] Miroslav Olsák, Cezary Kaliszyk, and Josef Urban. Property invariant embedding for automated reasoning. *CoRR*, abs/1911.12073, 2019. arXiv:1911.12073.

[30] Jens Otten and Wolfgang Bibel. leanCoP: lean connection-based theorem proving. *J. Symb. Comput.*, 36(1-2):139–161, 2003. URL: https://doi.org/10.1016/S0747-7171(03)00037-3, doi:10.1016/S0747-7171(03)00037-3.

[31] Christine Paulin-Mohring. Inductive definitions in the system Coq rules and properties. In Marc Bezem and Jan Friso Groote, editors, *Typed Lambda Calculi and Applications, International Conference on Typed Lambda Calculi and Applications, TLCA '93, Utrecht, The Netherlands, March 16-18, 1993, Proceedings*, volume 664 of *Lecture Notes in Computer Science*, pages 328–345. Springer, 1993. URL: https://doi.org/10.1007/BFb0037116, doi:10.1007/BFb0037116.

[32] Pierre-Marie Pédrot. Ltac2: Tactical warfare. In *The Fifth International Workshop on Coq for Programming Languages, CoqPL 2019, Held as Part of the 46th ACM SIGPLAN Symposium on Principles of Programming, POPL 2019, Cascais, Portugal, January 13-19, 2019*, 2019. URL: https://popl19.sigplan.org/details/CoqPL-2019/8/Ltac2-Tactical-Warfare.

[33] Alexandre Riazanov and Andrei Voronkov. Vampire. In Harald Ganzinger, editor, *Automated Deduction - CADE-16, 16th International Conference on Automated Deduction, Trento, Italy, July 7-10, 1999, Proceedings*, volume 1632 of *Lecture Notes in Computer Science*, pages 292–296. Springer, 1999. URL: https://doi.org/10.1007/3-540-48660-7_26, doi:10.1007/3-540-48660-7_26.

[34] Stephan Schulz. System description: E 1.8. In Kenneth L. McMillan, Aart Middeldorp, and Andrei Voronkov, editors, *Logic for Programming, Artificial Intelligence, and Reasoning - 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14-19, 2013. Proceedings*, volume 8312 of *Lecture Notes in Computer Science*, pages 735–743. Springer, 2013. URL: https://doi.org/10.1007/978-3-642-45221-5_49, doi:10.1007/978-3-642-45221-5_49.

[35] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, Yutian Chen, Timothy Lillicrap, Fan Hui, Laurent Sifre, George van den Driessche, Thore Graepel, and Demis Hassabis. Mastering the game of Go without human knowledge. *Nature*, 550:354–359, October 2017. doi:10.1038/

nature24270.

[36] Konrad Slind and Michael Norrish. A brief overview of HOL4. In Otmane Aït Mohamed, César A. Muñoz, and Sofiène Tahar, editors, *Theorem Proving in Higher Order Logics, 21st International Conference, TPHOLs 2008, Montreal, Canada, August 18-21, 2008. Proceedings*, volume 5170 of *Lecture Notes in Computer Science*, pages 28–32. Springer, 2008. URL: https://doi.org/10.1007/978-3-540-71067-7_6, doi:10.1007/978-3-540-71067-7_6.

[37] The Coq Development Team. The Coq proof assistant, version 8.10.0, Oct 2019. doi:10.5281/zenodo.3476303.

[38] Josef Urban, Jirí Vyskocil, and Petr Stepánek. MaLeCoP machine learning connection prover. In Kai Brünnler and George Metcalfe, editors, *Automated Reasoning with Analytic Tableaux and Related Methods - 20th International Conference, TABLEAUX 2011, Bern, Switzerland, July 4-8, 2011. Proceedings*, volume 6793 of *Lecture Notes in Computer Science*, pages 263–277. Springer, 2011. URL: https://doi.org/10.1007/978-3-642-22119-4_21, doi:10.1007/978-3-642-22119-4_21.

[39] Minchao Wu, Michael Norrish, Christian Walder, and Amir Dezfouli. Reinforcement learning for interactive theorem proving in HOL4. Accepted to AITP'20, 2020.

[40] Kaiyu Yang and Jia Deng. Learning to prove theorems via interacting with proof assistants. In Kamalika Chaudhuri and Ruslan Salakhutdinov, editors, *Proceedings of the 36th International Conference on Machine Learning, ICML 2019, 9-15 June 2019, Long Beach, California, USA*, volume 97 of *Proceedings of Machine Learning Research*, pages 6984–6994. PMLR, 2019. URL: http://proceedings.mlr.press/v97/yang19a.html.