Machine Learning for Automated Inductive Theorem Proving

Machine learning for inductive theorem proving

Yaqing Jiang, Petros Papapanagiotou, and Jacques Fleuriot

School of Informatics, University of Edinburgh,
10 Crichton Street, Edinburgh EH8 9AB, UK
{YQ.Jiang,pe.p,jdf}@ed.ac.uk

Abstract. Over the past few years, machine learning has been successfully combined with automated theorem provers to prove conjectures from proof assistants. However, such approaches do not usually focus on inductive proofs. In this work, we explore a combination of machine learning, a simple Boyer-Moore model and ATPs as a means of improving the automation of inductive proofs in the proof assistant HOL Light. We evaluate the framework using a number of inductive proof corpora. In each case, our approach achieves a higher success rate than running ATPS or the Boyer-Moore tool individually.

Keywords: Induction · Lemma selection · Theorem proving · Machine learning

1 Introduction

Over the past few years, large libraries of formalised theories have been built in interactive theorem provers (ITPs) like Isabelle [26], HOL Light [15] and Coq [1] . Automated, first-order theorem provers (ATPs) like Vampire [23] and E [29], and satisfiability modulo theories (SMT) solvers like Z3 [10] are increasingly being used to facilitate the development of such libraries in large proof corpora.

In order to use such external tools effectively, machine learning (ML) infrastructures have been developed within several proof assistants to automatically select hundreds of potentially relevant lemmas whenever the user tries to prove a goal automatically. Sledgehammer [28] in Isabelle and HOL(y)Hammer [21] in HOL Light are examples of two such ML systems.

Although recursively-defined data types such as lists are widely used in ITPs, the ATPs and SMT solvers do not usually perform well on goals that require inductive theorem proving [9]. However, automated methods for inductive theorem proving do exist. ACL2 [22], for instance, is a system that evolved from the so-called *Boyer-Moore* approach (which we use in our current work) and is successfully being used for the formalization of industrial problems. Inductive theorem proving often requires the manual provision of suitable lemmas to help with the inductive proof (for example as *hints* in ACL2). Identifying such lemmas is a major challenge and the system relies on human expertise and understanding of the problem and its context.

Lemma discovery techniques, which try to automatically speculate relevant lemmas, have been investigated as a solution [8,11]. These include, for example,

generalization, which was incorporated in the original Boyer-Moore prover but has had relatively limited success.

In the current work, we investigate the potential use of machine learning to select lemmas from big corpora in support of automated inductive theorem proving. We aim to select a relatively small number of suitable lemmas that can then be used within a Boyer-Moore based inductive theorem prover to make progress with otherwise blocked proofs.

We incorporate proof strategies that make use of machine learning techniques and ATPs within a Boyer-Moore style model and run these in parallel, in a new environment we call a *multi-waterfall*. Our paper is organised as follows: we introduce the Boyer-Moore model and lemma selection approaches in Section 2. We present the multi-waterfall model in Section 3, together with the application of lemma selection, and other changes to our Boyer-Moore implementation. We evaluate the different strategies on corpora of inductive proofs in Section 4, and discuss the results in Section 5.

2 Background

2.1 Recursively-Defined Data Types and Induction

Recursively-defined data types are usually used in inductive theorem proving. For instance, a natural number is either the constant 0, or obtained by applying the *successor* function s to another natural number. Inductive inference involves the use of particular logical rules to prove properties of recursive datatypes that are not otherwise provable [6]. The induction for natural numbers is:

$$\frac{P(0), \forall n.\ P(n) \implies P(s(n))}{\forall x.\ P(x)} \tag{1}$$

Applying this rule allows us to break a subgoal about a particular property P of natural numbers into two new subgoals: the base case $P(0)$ and the step case $P(s(n))$, assuming $P(n)$ for any n.

2.2 The Boyer-Moore Model

The Boyer-Moore approach [4] covers the key components of an automated theorem prover for inductive proofs. It revolves around the notion of a *waterfall model*, as shown in Fig. 1. In this, conjectures (or proof goals) are poured from the top and through a series of procedures, called *heuristics*. Each heuristic in the waterfall tries to either prove or simplify the goal. It may also determine that the goal is unprovable or, if neither of these is applicable, the heuristic *fails*.

Induction is applied automatically when all heuristics have failed (the goals trickle down to the pool at the bottom of the waterfall). The generated subgoals (base and step cases) are poured over new waterfalls again. This process is repeated recursively, until all subgoals are proven, in which case a proof of the original goal is reconstructed, or a subgoal is determined to be unprovable.

Examples of heuristics that are relevant to this paper are the following:

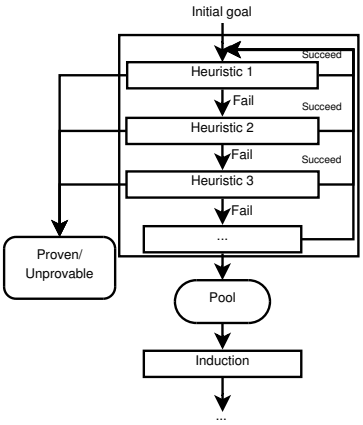


Fig. 1: Diagram of the Waterfall Model

- **The Clausal Form heuristic:** This transforms the goals to Clausal Normal Form (CNF), which other Boyer-Moore heuristics take advantage of.
- **The Simplify heuristic:** This applies rewriting to the goal in order to simplify or prove it using function definitions and rewrite rules. Note that termination is not guaranteed and depends on the selection of rewrite rules.
- **The Generalize heuristic:** A lemma speculation process which tries to generalize a subterm in the goal.
- Automated proof procedures in HOL Light such as the model elimination procedure `MESON` [16] and a simple tautology checker `TAUT` can also be used as heuristics within Boyer-Moore.

Boyer-Moore uses an additional heuristic at the pool of the waterfall to choose the appropriate induction variable based on the definitions of recursive functions [4]. Note that in our implementation, this heuristic only supports primitively recursive definitions [3].

Based on the above, the system configuration can be tailored to deal with different problems. The most common customizations are the following[27]:

- The rewrite rules for the Simplify heuristic can be elaborately chosen by the user to improve its effectiveness towards proving the subgoals.
- The order and combination of heuristics can also be adjusted for different situations. For instance, some heuristics are unsafe and may render the goal more complicated, or result in an infinite loop.

For our implementation we use the Boyer-Moore system implemented in HOL-Light [27]. An important advantage of both this particular system and HOL Light, particularly in comparison with more sophisticated evolutions of the Boyer-Moore approach such as ACL2, is that they are lightweight implementations with simple structures and thus allow easy, direct access to the inner workings. This makes it easier to manipulate and adjust the Boyer-Moore waterfalls and heuristics, and analyze the effects of machine learning more thoroughly.

2.3 Theorem Proving Hammers

As we alluded to in the introduction, ITPs now incorporate so-called *hammers* that act as intermediates between powerful, external ATPs and their built-in proof procedures. With the help of machine learning, these allow users to reconstruct complex formal proofs within ITPs with just one click. Hammers usually consists of four parts [2]:

- A *lemma selection* module to filter relevant lemmas that can be used by ATPs (see Section 2.4).
- A *translation module* that translates ITP problems to a first order syntax acceptable to ATPs.
- Links to external ATPs that search for and output proofs.
- A *proof reconstruction* module that reconstructs the output of ATPs to corresponding ITP proofs.

Sledgehammer is the original tool that started the whole effort: it is integrated into the Isabelle proof assistant that carries out lemma selection using a combination of relevance filtering *MePo* [25] and Bayesian learning [24].

HOL(y)Hammer, the corresponding tool for HOL Light, also uses machine learning for lemma selection. In our work, we incorporate elements from its latest released version[1], such as its *feature extraction* algorithm (see Section 2.4).

2.4 Machine Learning for Interactive Theorem Proving

Lemma selection is an important component of hammers as they provide the external ATPs with the pre-proved results that may lead to a proof. This usually involves training an ML model that can predict the relevance of proven lemmas to new goals and then select those that look more promising. The model is typically trained using existing proofs that have been produced interactively. More specifically, a *dependency tracking* module usually records the definitions and lemmas that have been used during interactive proofs.

In our case, we have developed our own dependency tracking tool that improves upon the one in HOL(y)Hammer by recording additional information such as whether a tracked theorem is a definition and the file that contains it.

Both Sledgehammer and HOL(y)Hammer use ML algorithms, such as naive Bayes, that estimate the relevance of lemmas from the proof library based on features generated from the statements of lemmas and the goal at hand. Such features usually consist of strings generated from the constants, subterms, operators, and other parts of the statement [20].

For example, given the HOL-Light theorem $\forall n.\ EVEN\ n \vee ODD\ n$, the following features are extracted:

$$\begin{aligned} &\text{“}num\text{”}, \text{“}fun\text{”}, \text{“}bool\text{”}, \text{“}ODD\text{”}, \text{“}EVEN\text{”}, \\ &\text{“}Anum\text{”}, \text{“}EVEN\ Anum\text{”}, \text{“}ODD\ Anum\text{”} \end{aligned} \tag{2}$$

[1] http://cl-informatik.uibk.ac.at/software/hh/hh-0.13.tgz

In HOL(y)Hammer this is achieved in several ways. For instance, given the term $EVEN\ n : num$ (where "$n : num$" means the type of n is the natural numbers), the default option normalizes the identifier of variable n to an identifier "A" followed by the type num, i.e. "$Anum$". Moreover, structural information is kept as additional features with entire subterms, e.g. "$EVEN\ Anum$" above, which provides more information for learning [24].

3 Methodology

As mentioned in Section 2, our work is based on an implementation of the Boyer-Moore model in HOL Light. We followed an experiment-led methodology, using the setup described in Section 4. The results of repeated experiments empirically guided our decision making in order to improve and configure the system and expand it with machine learning techniques inspired by hammers. In this section, we summarize the key changes made to the original Boyer-Moore system.

3.1 Initial Improvements

Initial experiments were done to form a baseline against which to compare the results of changes and additions. During these experiments we noticed and fixed a number of issues, the most important of which are described next.

Removing CNF Heuristic During our initial experiments, some goals became unprovable by Boyer-Moore after the CNF heuristic was applied. For instance, the heuristic splits the goal $\neg EVEN\ x \iff ODD\ x$ into 2 clauses: $EVEN\ x \vee ODD\ x$ and $\neg EVEN\ x \vee \neg ODD\ x$. In the original formalization, the untransformed goal is proven independently and used as a lemma to be able to prove these 2 clauses. This is an indication that the CNF heuristic does not always make progress in the right direction towards a proof.

Moreover, the CNF heuristic breaks goals that contain logical equivalences (iffs) into subgoals containing implication, leading to the generation of a number of subgoals that is exponential to the number of equivalences encountered in the original goal. Therefore, removing it can significantly reduce the total amount of subgoals.

It is worth noting that removing the CNF heuristic directly affects some of the Boyer-Moore heuristics that follow, which rely on CNF. Despite this side-effect, our experiments showed a significant overall improvement in the performance of Boyer-Moore without the CNF heuristic.

Generalising Variables When applying induction to a formula with more than one universally quantified variable, only one is typically selected for induction, and the others are not affected [6]. For example, applying induction on variable n in the formula $\forall n\ m.\ Q(n, m)$ yields the following step case:

$$\forall n'.\ (\forall m.\ Q(n', m)) \implies (\forall m.\ Q(s(n'), m)) \tag{3}$$

However, in Boyer-Moore the input formula is always *quantifier-free*, so the step case generated is the following instead:

$$Q(n, m) \implies Q(s(n), m) \tag{4}$$

This stronger subgoal may be unprovable in certain cases compared to its weaker counterpart (3). Our solution is to generalise all variables other than the one for induction as follows:

$$(\forall m.\ Q(n, m)) \implies Q(s(n), m') \tag{5}$$

Applying induction then yields the same subgoal (3), though we then remove the quantifiers again to fit to the *quantifier-free* environment of Boyer-Moore.

HOL Light's Automated Procedures During early experiments, we identified (sub)goals that could be proven by HOL Light's automated model elimination procedure MESON. Therefore, MESON was added as a heuristic to the waterfall.

Forced Induction As mentioned previously, the induction heuristic in Boyer-Moore can only handle primitively recursive function definitions. This means Boyer-Moore failed to perform induction in terms containing any non-primitively recursive functions as it was unable to choose an appropriate variable. We address this problem by forcing Boyer-Moore to pick the first free variable with a recursive type for induction if no other suitable selection is found by the original heuristic. For the future, we are considering the use of machine learning techniques as a more sophisticated mechanism for the selection of induction variables.

3.2 The Multi-waterfall Model

The original setup of the waterfall works in a serial, monolithic way. Each heuristic is tried sequentially in a static order. However, certain proofs may require different configurations or strategies for different subgoals. Moreover, some of the Boyer-Moore heuristics may naturally get stuck during a proof. For example, certain combinations of rewrite rules may cause the Simplify heuristic to loop endlessly. This is particularly important in the context of automated lemma selection where we have less control over looping rewrite rule sets. Using a different configuration might help unlock and make progress with the proof.

In order to achieve a more flexible implementation that does not rely on a single configuration, we introduce a *Multi-waterfall model*. In this, we run multiple waterfalls with different configurations in parallel and with a preset timeout. We then have the following possible outcomes:

1. One of the waterfalls succeeds and the corresponding (sub)goal is proven. The proof of the (sub)goal is reconstructed and propagated upwards (as in the standard waterfall model), ensuring soundness of the overall proof.

2. One of the waterfalls completes having generated new subgoals that reached their pools. In this case, we apply induction to all unproven goals as in the standard waterfall model (see Section 2.2). We then apply the same set of multiple waterfalls to each of the new sugboals generated by induction.
3. All the waterfalls determine the goal is unprovable, or the timeout is reached. In this case, the whole branch of proof search fails and is discarded.

The timeout applied to each waterfall ensures that any waterfalls that take too long are assumed to have failed and are forcibly stopped and their corresponding branches abandoned. This allows the other waterfalls running in parallel to still potentially make progress towards the proof.

An example search tree with 2 waterfalls is shown in Fig. 2. The waterfalls are run in parallel on the same goal. When a waterfall finishes, we apply induction to any unproven subgoals in its pool, constructing new subgoals indicated by the dashed arrows. We then start new waterfalls for each generated subgoal until all subgoals are proven or deemed unprovable.

A full proof can be reconstructed by tracking all successful waterfalls in a branch. This means a proof may be found by a chain of different waterfalls. In Fig. 2, for example, the proof is reconstructed by the waterfalls enclosed in the marked area. Notice that both types of waterfalls were used to make progress on or prove different subgoals.

In our implementation, we spawn the waterfalls for a particular goal using threaded concurrency. If a waterfall fully proves a goal (such as Waterfall 1" in Fig. 2), the other waterfalls working on the same goal (such as Waterfall 2") and their children are forcibly stopped in order to release system resources. Waterfalls could be tried sequentially instead, but this would dramatically increase the time taken for a proof to complete, e.g. because the user would need to wait for different waterfalls to timeout for each and every subgoal.

3.3 Lemma Selection for Boyer-Moore

A straightforward way to apply lemma selection in the Boyer-Moore model is to pick rewrite rules for the Simplify heuristic or, more generally, any heuristic that requires relevant lemmas. For this purpose, we train a classifier on the proofs that are encountered up to the current goal (see Section 2.4). We then use that to select relevant lemmas for each subgoal encountered in the waterfall.

The main issue with lemma selection in this context is that the number of selected lemmas must be bounded. The larger the rewrite rule set, the more likely it is that the Simplify heuristic will loop. Selecting fewer lemmas means that key lemmas may be classified as 'not relevant enough' and not be selected.

Replacing the conditional simplifying function `SIMP_CONV` in the original Boyer-Moore implementation with the simpler rewrite function `REWRITE_CONV` helped improve our results, but only slightly. The same problem was observed with MESON as it could not handle large sets of lemmas, and timed out. For that reason MESON is currently used on its own without lemmas.

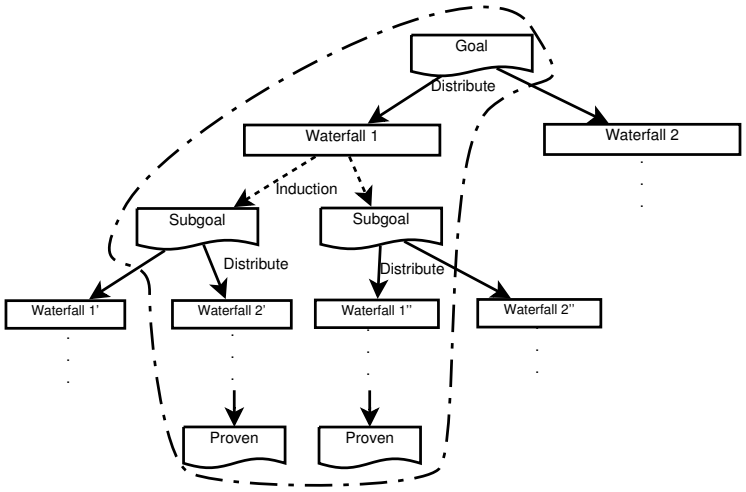


Fig. 2: Proof search with multi-waterfall

In contrast, ATPs are good at handling large numbers of lemmas in more ways than just simplification (see Section 2.3). We take advantage of this by adding a modified version of HOL(y)Hammer (see Section 4.4) as a heuristic that can directly prove a (sub)goal. We call this heuristic the *ATP heuristic*.

3.4 Direct Induction

It is quite common in manual inductive proofs for the reasoning to begin with induction before any simplification or other proof steps. In Boyer-Moore such proofs may get stuck waiting for the ATP or Simplify heuristics and eventually timing out and failing, whereas applying induction directly could help unlock the proof. Moreover, some goals in our initial experiments were being rewritten to a form that caused Boyer-Moore to either choose a wrong variable for induction or have more complicated subgoals after induction (for example because complex definitions were expanded unnecessarily) and fail.

For these reasons, we constructed a new configuration of the waterfall with no heuristics, but instead induction is applied directly. Including this in our multi-waterfall model (see Section 3.2) enables proofs where this waterfall is used first, so that induction is applied directly, then another waterfall uses heuristics to prove the subgoals, thus mimicking the manual proofs mentioned above.

4 Experiment Set-Up

4.1 Datasets

In order to evaluate our work, we use proven theorems about recursively-defined data types. We note here that the *IsaPlanner benchmark* [19], which has been

used by some to test automated inductive theorem provers [8,9], is unsuitable in our case for the following reasons:

1. Many of the definitions use case-expressions, which are not currently supported by HOL Light.
2. The available version[2] contains many theorems that are part of the recursive definitions of the corresponding functions, and so can be proven trivially.
3. In the evaluation of HipSpec, 67 theorems were proven without using any auxiliary lemmas, and more than 10 were proven using only rewriting. Therefore, lemma selection would not have any impact in these examples.

Instead, we chose the following corpora for testing [3]:

1. *The core list library in HOL Light*, which we refer to as *List*(*core*).
2. *An additional list library* used in the formalization of Hilbert's Foundations of Geometry [30]. We refer to this as *List*(*hilbert*).
3. *A polynomials library in HOL Light* with properties about real polynomials represented as lists of coefficients. We refer to this as *Poly*.

The size of the test data is shown in Table 1. Note that conjunctions have been split, meaning that a theorem (or definition) $P \wedge Q$ is automatically split into P and Q as separate goals (or definitions).

	Definitions	Theorems	Inductive
List(*core*)	44	97	73 (75.26%)
List(*hilbert*)	22	115	80 (69.57%)
Poly	20	123	67 (54.47%)

Table 1: Size of the testing data

Note that the number of inductive proofs is a lower bound, obtained by tallying the proofs containing the string "INDUCT". In our current datasets we did not observe any inductive proofs that were not captured in this way, but this is not necessarily true for other libraries. Since induction can be applied in various ways in HOL Light (e.g. by matching different induction rules), it is somewhat difficult to automatically determine the exact number of inductive proofs.

4.2 Experiments

In order to show that the Boyer-Moore model is a good starting point for inductive theorem proving, a comparison between Boyer-Moore and a simple "induction then rewriting" proof strategy was made. Such a strategy is commonly used in manual proofs for a large number of (relatively simple) inductive theorems. We will refer to it as *Ind simp*.

[2] https://github.com/tip-org/benchmarks/tree/master/original/isaplanner
[3] https://github.com/zidongtuili/BM_test

We then performed the following experiments using the methods described in Section 3:

1. *Original*: Running the original Boyer-Moore implementation as a baseline.
2. *Initial*: Running Boyer-Moore with the changes from Section 3.1.
3. *Multi-waterfall*: Running the multi-waterfall model described in Section 3.2, using the three waterfalls shown in Table 2. More specifically, we used a waterfall with the ATP heuristic, a standard waterfall with the Simplify and MESON heuristics, and a waterfall with direct induction (see Section 3.4).
4. *ATP*: The combination of lemma selection with the ATP heuristic *outside* Boyer-Moore, i.e. without induction, so that we evaluate and compare the performance of ATPs on inductive proofs independently.

Heuristic	Waterfall 1	Waterfall 2	Waterfall 3
Simplify		×	
MESON		×	
Other Heuristics	×	×	
HOL(y)Hammer	×		
Induction	×	×	×

Table 2: Heuristic settings for three waterfalls

Note that in the experiments without lemma selection (*Original* and *Initial*), the built-in rewrite rules and definitions in Boyer-Moore are used.

4.3 Metrics

For each experiment we evaluate the *total* success rate as n/m where n is the total number of theorems proven and m is the total number of tested theorems. We also consider the *inductive* success rate in the same way for the subset of inductive theorems tested.

4.4 Environment

Two ATPs were used in our experiments: Vampire 4.1[4] and Epar (a wrapper of E included in HOL(y)Hammer) [31]. *Sparse Naïve Bayes*, as the only ML algorithm included in the source code of HOL(y)Hammer, was used as the learning algorithm. We ported their optimised implementation from *Mash*[5] [24].

We set the timeout for each waterfall to 30 seconds, which is a reasonable time that a user would wait for the system as well as the default timeout of Sledgehammer and HOL(y)Hammer [24,20]. For lemma selection we select the top 256 most relevant lemmas, which is the value at which the success rate of

[4] http://www.cs.miami.edu/ tptp/CASC/J8/

[5] https://github.com/seL4/isabelle/blob/master/src/HOL/Tools/Sledgehammer/sledgehammer_mash.ML

Vampire and Epar is known to drop significantly [20]. Such parameters cannot be optimized globally as each goal may require different values (the user could tinker with the values in an interactive setting). We believe that the current settings are reasonable for the automated evaluation of our implementation, and further optimisations can be tested in future experiments.

In order to run multiple waterfalls in parallel, a multi-core machine was used with 2 *Intel(R) Xeon(R) CPU E5-2690 v2 @ 3.00GHz* (40 threads in total) with 64GB RAM. Note that the actual CPU load varies for different problems and is relatively low in most cases.

5 Evaluation

5.1 Results

The comparison between the original implementation of Boyer-Moore and *Ind simp* is shown in Table 3. *Ind simp* is weaker overall than Boyer-Moore. Boyer-Moore only failed on 2 theorems proven by *Ind simp* mainly due to the issue with the CNF heuristic mentioned in Section 3.1.

	List(*core*)	*List*(*hilbert*)	*Poly*
Ind simp	24.74%	13.04%	8.94%
Original	41.24%	14.78%	13.01%

Table 3: Success rate of *Ind simp* and the original Boyer-Moore.

The results of the rest of the experiments are shown below in Table 4.

	Total			Induction		
	List(*core*)	*List*(*hilbert*)	*Poly*	*List*(*core*)	*List*(*hilbert*)	*Poly*
Original	41.24%	14.78%	13.01%	36.99%	8.75%	11.94%
Initial	52.58%	20.00%	14.63%	45.21%	17.50%	13.43%
Multi-waterfall	57.73%	63.48%	40.65%	46.58%	62.50%	37.31%
ATP	25.77%	36.52%	24.39%	5.48%	30.00%	10.45%

Table 4: Success rates of the different configurations

Initial generally outperformed *Original*, which was still able to prove some theorems that *Initial* failed on though, due to the failure of some heuristics that rely on CNF.

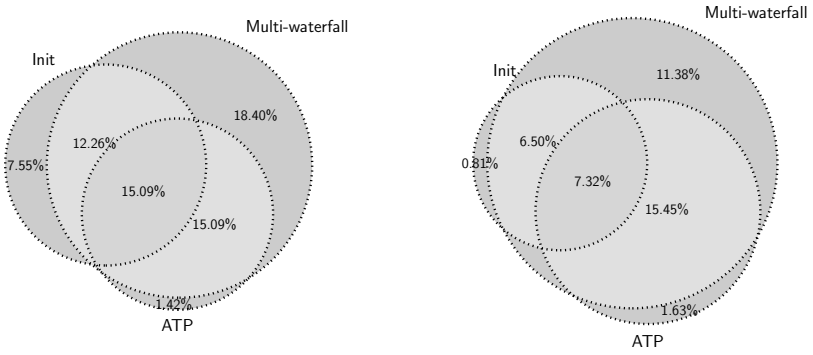
Performance was increased in *Multi-waterfall* compared to *Initial* at a different scale for each of the 3 sets, as shown in Table 4. This indicates that the original Boyer-Moore's built-in lemmas are enough to prove theorems in *List*(*core*), while lemma selection is more effective for corpora that contain more difficult theorems and a larger variety of useful lemmas.

ATPs performed relatively poorly on inductive theorems (which significantly affected their total success rate as well). However, ATPs had a high success rate

in *List*(*hilbert*). This shows that with appropriate lemma selection, ATPs can indeed be powerful enough to prove inductive problems.

Fig. 3 shows a Venn diagram representation of the theorems proven by *Initial*, *Multi-waterfall*, and *ATP*, demonstrating the percentage of theorems that could only be proven by some of the methods, but not the others. *Multi-waterfall* could prove many theorems that none of the other methods could. This reveals the enhanced potential of combining lemma selection and Boyer-Moore.

In *List*(*core*)and *List*(*hilbert*), *Multi-waterfall* failed to prove some theorems that were proven by *Initial*. This is mainly due to the lack of conditional rewriting (see Section 3.3). Moreover, some theorems were proven by *ATP* but not *Multi-waterfall*, because *Multi-waterfall* requires *quantifier-free* goals as input. This affects how the goals are translated to the ATP format, particularly for higher order (i.e. function) variables, and thus impacts the performance of ATPs.



(a) *List*(*core*) & *List*(*hilbert*) (b) *Poly*

Fig. 3: Coverage of proven theorems by different methods in Table 4

Examining failed proofs in *Multi-waterfall*, we discovered that in many cases the wrong variable was chosen for induction, particularly when 2 or more induction steps are used in a proof (at least 25% of the time in each data set). Other failed proofs can be attributed to missing key lemmas during lemma selection.

5.2 Examples

An example of an inductive theorem is `DROP_DROP` from *List*(*hilbert*) shown in Fig. 4. It is worth comparing the manual proof to the one generated by Boyer-Moore. With the push of a button, a theorem with a complex manual proof containing 3 induction steps can be proven by *Multi-waterfall* automatically in only 2 induction steps. The corresponding proof script for the new proof is automatically generated and verified in HOL Light. Also note that HOL(y)Hammer was unable to find a proof on its own, neither when supplied with the same lemmas used in *Multi-waterfall* nor with its own selection of 256 lemmas.

DROP_DROP: $\forall n, m, xs$: DROP $(n + m)$=DROP n (DROP m xs)
Manual proof:

```
INDUCT_TAC THEN REWRITE_TAC [ADD_CLAUSES;DROP]
   THEN INDUCT_TAC THEN ASM_REWRITE_TAC [LENGTH;ADD_CLAUSES;DROP]
   THEN LIST_INDUCT_TAC THEN ASM_REWRITE_TAC [LENGTH;ADD_CLAUSES;DROP]
   THEN REWRITE_TAC [GSYM ADD] THEN ASM_REWRITE_TAC [DROP;ADD_CLAUSES]
```

Proof generated by Boyer-Moore *Multi-waterfall*:

```
REPEAT GEN_TAC THEN REWRITE_TAC[conj 0 ADD_AC] THEN
IND_MP_TAC [‘xs:(a)list‘] list_INDUCT THEN CONJ_TAC THEN
CONV_TAC (REPEATC (DEPTH_FORALL_CONV RIGHT_IMP_FORALL_CONV)) THEN
(REPEAT GEN_TAC) THENL [REWRITE_TAC[conj 1 DROP];
IND_MP_TAC [‘m:num‘] num_INDUCTION THEN CONJ_TAC THEN
CONV_TAC (REPEATC (DEPTH_FORALL_CONV RIGHT_IMP_FORALL_CONV)) THEN
(REPEAT GEN_TAC) THENL [REWRITE_TAC [conj 0 DROP;conj 0 ADD];
SIMP_TAC[conj 1 ADD;conj 2 DROP];];]
```

Fig. 4: User and Boyer-Moore proofs for DROP_DROP

An example of a failed proof is the `LENGTH_REVERSE` theorem shown in Figure 5. It has a short manual proof with only one induction step and was proven by *Initial*, but not by *Multi-waterfall*. Further investigation showed that when trying to prove a particular subgoal, although lemma selection included the 6 lemmas that were sufficient for the proof, ATPs still failed to find it (even after being allowed to run for 60 seconds, i.e. double the time). In our later experiments, a list of 13 theorems (including definitions and common rewrite rules for lists) can easily prove many subgoals when used on their own, but not as part of a large selection. This shows that a small group of carefully picked lemmas can be more effective than a large number of automatically selected lemmas. This explains why *Multi-waterfall* failed to prove some theorems that *Initial* proved.

LENGTH_REVERSE: $\forall xs$. LENGTH (REVERSE xs) = LENGTH xs
Manual proof:

```
LIST_INDUCT_TAC THEN ASM_REWRITE_TAC
   [LENGTH;REVERSE;LENGTH_APPEND] THEN ARITH_TAC
```

Fig. 5: User proof for LENGTH_REVERSE

6 Related Work

There is a number of other systems for the automation of inductive proofs. *Isaplanner* [11] is a generic framework for proof planning in Isabelle with lemma speculation techniques [12] that try to derive and prove useful lemmas from a goal. *HipSpec* [7] uses a bottom-up approach to generate lemmas that can be used to prove inductive properties of Haskell programs. *Cruanes* [9] is another system which supports structural induction with an extension to superposition-based provers. *TacticToe* [13,14] is a very recent effort that attempts to learn from human (manual) proofs and uses a Monte Carlo Tree Search [5] as it attempts to construct a proof. Based on a timeout of 60s, a (very high) success rate of 79.5% is reported when it comes to reproving the theorems in the HOL4 list library. It should be instructive to compare the performance of our approach on the same corpus.

We should also note that there has been some work on combining machine learning techniques with inductive theorem proving in ACL2 [18,17]. The approaches are different from ours in the following ways :

- We apply lemma selection at each subgoal independently, while ACL2(ml) generally applies its search only at the beginning on the whole goal. Our fine-grained approach is possible thanks to the simplicity and accessibility our HOL Light test bed (in contrast to the complicated structure of ACL2).
- Unsupervised learning (clustering), which focuses on the similarity between goals and theorems, was used in ACL2(ml).
- The features used in ACL2(ml) are based on the structure of the formulae, which makes them suitable for selecting lemmas with a desired structure and then mutating them into a simple form of analogical reasoning.

7 Conclusion

Experiments with three corpora containing a large number of inductive proofs have demonstrated that the integration of machine learning in a Boyer-Moore model can greatly improve its ability to prove complex inductive theorems. The combination of powerful ATPs with lemma selection techniques and the Boyer-Moore strategies and heuristics for inductive proofs have allowed us to automatically prove a large number of theorems that neither system could prove independently.

This effective combination was enabled by a new multi-waterfall model that allows multiple proof strategies to be used in parallel to prove different subgoals. This model is configurable with respect to the time out and number of selected lemmas, which can be changed to improve its effectiveness, particularly in an interactive setting. However, improvements in the user interaction and feedback provided by the Boyer-Moore tool (perhaps with ideas from ACL2) seem paramount in order to achieve even higher proof success.

The model can also be extended with more than the currently suggested three waterfalls, so as to incorporate additional strategies and techniques in the

future. For example, we could add more types or combinations of heuristics, incorporate case splitting, and include better support for non-recursive types.

Our future work will also focus on further uses of machine learning in this setting, for example as a mechanism to select an appropriate induction variable.

We believe our approach is a generic solution for the use of machine learning within proof strategies for automated inductive theorem proving. Using our *Multi-waterfall* model as a skeleton to develop such inductive proof strategies has the potential to greatly enhance the current capabilities of existing systems without sacrificing their individual effectiveness.

Acknowledgements This research was supported by EPSRC grants: ProofPeer: Collaborative Theorem Proving EP/L011794/1 and The Integration and Interaction of Multiple Mathematical Reasoning Processes EP/N014758/1. It was also supported by the China Scholarship Council (CSC).

References

1. Bertot, Y., Castéran, P.: Interactive theorem proving and program development: CoqArt: the calculus of inductive constructions. Springer Science & Business Media (2013)
2. Blanchette, J.C., Kaliszyk, C., Paulson, L.C., Urban, J.: Hammering towards QED. Journal of Formalized Reasoning **9**(1), 101–148 (2016)
3. Boulton, R.: Boyer-Moore automation for the HOL system. In: Higher Order Logic Theorem Proving and its Applications, pp. 133–142. Elsevier (1993)
4. Boyer, R., Moore, J.: A Computational Logic. ACM monograph series, Academic Press (1979)
5. Browne, C., Powley, E.J., Whitehouse, D., Lucas, S.M., Cowling, P.I., Rohlfshagen, P., Tavener, S., Liebana, D.P., Samothrakis, S., Colton, S.: A survey of Monte Carlo Tree Search methods. IEEE Trans. Comput. Intellig. and AI in Games **4**(1), 1–43 (2012)
6. Bundy, A.: The automation of proof by mathematical induction. Tech. rep. (1999)
7. Claessen, K., Johansson, M., Rosén, D., Smallbone, N.: Hipspec: Automating inductive proofs of program properties. In: ATx/WInG@ IJCAR. pp. 16–25 (2012)
8. Claessen, K., Johansson, M., Rosén, D., Smallbone, N.: Automating inductive proofs using theory exploration. In: International Conference on Automated Deduction. pp. 392–406. Springer (2013)
9. Cruanes, S.: Superposition with structural induction. In: International Symposium on Frontiers of Combining Systems. pp. 172–188. Springer (2017)
10. De Moura, L., Bjørner, N.: Z3: An efficient smt solver. In: Tools and Algorithms for the Construction and Analysis of Systems, pp. 337–340. Springer (2008)
11. Dixon, L., Fleuriot, J.: Isaplanner: A prototype proof planner in Isabelle. In: International Conference on Automated Deduction. pp. 279–283. Springer (2003)
12. Dixon, L., Johansson, M.: Isaplanner 2: A proof planner in isabelle. DReaM Technical Report (System description) (2007)
13. Gauthier, T., Kaliszyk, C., Urban, J.: Tactictoe: Learning to reason with HOL4 tactics. In: LPAR-21. 21st International Conference on Logic for Programming, Artificial Intelligence and Reasoning. vol. 46, pp. 125–143 (2017)

14. Gauthier, T., Kaliszyk, C., Urban, J., Kumar, R., Norrish, M.: Learning to prove with tactics. CoRR **abs/1804.00596** (2018), http://arxiv.org/abs/1804.00596
15. Harrison, J.: HOL Light: A tutorial introduction. In: Formal Methods in Computer-Aided Design. pp. 265–269. Springer (1996)
16. Harrison, J.: Optimizing proof search in model elimination. In: International Conference on Automated Deduction. pp. 313–327. Springer (1996)
17. Heras, J., Komendantskaya, E.: ACL2(ml): Machine-learning for ACL2. arXiv preprint arXiv:1404.3034 (2014)
18. Heras, J., Komendantskaya, E., Johansson, M., Maclean, E.: Proof-pattern recognition and lemma discovery in ACL2. In: Logic for Programming, Artificial Intelligence, and Reasoning. pp. 389–406. Springer (2013)
19. Johansson, M., Dixon, L., Bundy, A.: Conjecture synthesis for inductive theories. Journal of Automated Reasoning **47**(3), 251–289 (2011)
20. Kaliszyk, C., Urban, J.: Learning-assisted automated reasoning with flyspeck. Journal of Automated Reasoning pp. 1–41 (2014)
21. Kaliszyk, C., Urban, J.: Hol (y) hammer: Online atp service for HOL Light. Mathematics in Computer Science **9**(1), 5–22 (2015)
22. Kaufmann, M., Manolios, P., Moore, J.: Computer-Aided Reasoning: An Approach. Advances in Formal Methods, Springer US (2000)
23. Kovács, L., Voronkov, A.: First-order theorem proving and vampire. In: Computer Aided Verification. pp. 1–35. Springer (2013)
24. Kühlwein, D., Blanchette, J.C., Kaliszyk, C., Urban, J.: Mash: Machine learning for Sledgehammer. In: Interactive Theorem Proving, pp. 35–50. Springer (2013)
25. Meng, J., Paulson, L.C.: Lightweight relevance filtering for machine-generated resolution problems. Journal of Applied Logic **7**(1), 41–57 (2009)
26. Nipkow, T., Paulson, L.C., Wenzel, M.: Isabelle/HOL: a proof assistant for higher-order logic, vol. 2283. Springer Science & Business Media (2002)
27. Papapanagiotou, P., Fleuriot, J.: The Boyer-Moore waterfall model revisited
28. Paulson, L.C., Blanchette, J.C.: Three years of experience with Sledgehammer, a practical link between automatic and interactive theorem provers. IWIL-2010 **1** (2010)
29. Schulz, S.: E - a brainiac theorem prover. AI Communications **15**(2, 3), 111–126 (2002)
30. Scott, P.: Ordered geometry in Hilbert's Grundlagen der Geometrie. Ph.D. thesis, The University of Edinburgh (2015)
31. Urban, J.: Blistr: The blind strategymaker. arXiv preprint arXiv:1301.2683 (2013)