# Stronger Automation for Flyspeck by Feature Weighting and Strategy Evolution

Cezary Kaliszyk and Josef Urban

[1] University of Innsbruck, Austria
[2] Radboud University, Nijmegen

**Abstract**

Two complementary AI methods are used to improve the strength of the AI/ATP service for proving conjectures over the HOL Light and Flyspeck corpora. First, several schemes for frequency-based feature weighting are explored in combination with distance-weighted $k$-nearest-neighbor classifier. This results in 16% improvement (39.0% to 45.5% Flyspeck problems solved) of the overall strength of the service when using 14 CPUs and 30 seconds. The best premise-selection/ATP combination is improved from 24.2% to 31.4%, i.e. by 30%. A smaller improvement is obtained by evolving targetted E prover strategies on two particular premise selections, using the Blind Strategymaker (BliStr) system. This raises the performance of the best AI/ATP method from 31.4% to 34.9%, i.e. by 11%, and raises the current 14-CPU power of the service to 46.9%.

## 1 Introduction

Methods for automated reasoning in large theories (ARLT) have started to develop in the recent years [8,15,20,24]. The primary driving force behind this development is the growing use of such methods for assisting ITPs like Isabelle [16] and Mizar [22, 23]. Recently, we have added HOL Light [7] to the pool of systems linked to the large-theory ATP methods [10], and experimented with the strongest and most orthogonal combinations of the premise-selection methods and various ATPs over the Flyspeck corpus [6]. The experimental work, described in [11], has shown that 39% of the 14185 Flyspeck theorems could be proved in a push-button mode (without any high-level advice and user interaction) in 30 seconds of real time on a fourteen-CPU workstation.

The work described in this paper improves two aspects of large-theory reasoning done on the Flyspeck corpus: (i) the premise selection methods, i.e., selecting from a large repository the lemmas, theorems, and definitions that are most relevant for a new conjecture, and (ii) the ATP strategies used to solve the problems after premise selection. The techniques used to achieve these improvements are quite straightforward. In the first case (Section 2), the improvement is obtained by better weighting (scaling) of the large number of features that are used as an input to the machine-learning algorithms that learn premise selection from previous proofs. Such feature weighting seems to be quite important particularly for the $k$-nearest-neighbor algorithm that we initially started to try on Flyspeck in [11]. Feature weighting seems to be a reasonably studied subject in the machine-learning community, in particular in the information-retrieval domain, and in some sense this work is quite a straightforward application of those studies. The result is however quite a surprising improvement over the best AI/ATP method used so far for Flyspeck, and also quite high improvement of the joint power of all AI/ATP methods used. An important practical advantage of using $k$-nearest-neighbor over the more sophisticated kernel methods explored recently on smaller corpora [1,14] is that $k$-nearest-neighbor works quite fast with the number of features and training examples that the Flyspeck corpus provides, while scaling up the kernel methods to the Flyspeck sizes is still work in progress.

The second improvement is obtained by a straightforward running of the newly developed BliStr (Blind Strategymaker) strategy-evolving system [21] on two classes of ATP problems that are created by different premise-selection methods. We have already been using for Flyspeck a custom strategy-scheduling version (Epar) of the E [17] prover, consisting of strategies developed by BliStr for the Mizar@Turing competition [18] on the 1000 Mizar@Turing training problems. While that version turned out to be significantly stronger than standard E on Flyspeck, it was still optimized on problems coming from a different corpus (Mizar). These Mizar problems were additionally quite small in comparison to the sizes of Flyspeck problems produced by the premise-selection methods that are most useful for Flyspeck. Section 3 shows that several hundred runs of the BliStr's strategy-evolving loop (slightly extended in comparison to [21], to also evolve further SInE-based [8] premise-pruning parameters) on these Flyspeck problem classes can again raise the performance of E quite considerably. The additional improvement of the overall power of the system is smaller than for the first method, but it is still quite significant, and more power can likely be added in the future by using strategy evolution also for the remaining important classes of Flyspeck problems.

## 2 Better Feature Weights for Nearest Neighbor

*Premise selection* is an essential AI component that has in the past decade allowed the usage of automated theorem provers (ATPs) over large corpora built with ITPs such as Mizar, Isabelle, and HOL Light. The premise-selection methods select from the large repositories the lemmas, theorems, and definitions (i.e., the *premises*) that are most relevant for a new conjecture. This is a hard AI problem, for which various heuristics taking into account the semantics and syntax of mathematical formulas can be considered. Such heuristics can involve (possibly approximative) deductive reasoning components. For example, the MoMM system [19] generalizes hundreds of thousands of existing mathematical lemmas, and using (deductively correct) type-aware ATP indexing methods combined with limited deductive reasoning tries to find suitable lemmas for a new conjecture. The SInE [8] heuristic is based on approximative reasoning about formulas, using only the symbols contained in the formulas. In some sense it carves out the set of formulas that (particularly in Horn-like ontologies) may be transitively related to the conjecture. A much less deductive way of selecting premises is to learn (use machine learning) what is relevant for what from the proofs contained already in the large repositories [1,14,20]. For this, the formulas are characterized by suitably chosen features that can be purely syntactic, such as the symbols and terms occuring in the formulas, or by more semantic/deductive features, such as models and abstractions of the formulas and relations between them. The machine learners then try to learn the association from such features to the premises that were most useful for proving the theorems. The key parts of such methods are therefore the learning (generalization) algorithms and the features used by the algorithms.

In order to get additional premise-selection power for the first large-scale AI/ATP experiments done over Flyspeck, we had quickly added in [11] a custom implementation of the $k$-nearest neighbor ($k$-NN) machine-learning method, which computes for a new example (conjecture) the $k$ nearest (in a given feature distance) previous examples and ranks premises by their frequency in these examples. The motivation was that this fast ("lazy" and trivially incremental) learning method can be easily parametrized and might for some parameters behave quite differently from the naive Bayes learner, which we had been using exclusively until then, because the newly developed kernel-based methods [1, 14] so far do not scale to such large corpora. Our (distance-weighted [4]) implementation weighs the contribution of the $k$ nearest neighbors by their feature-based similarity to the current conjecture, and computes the overall ranking of the

available premises as a sum of the premises' weighted contributions from these $k$ neighbors.

The performance of this (multi-class, distance-weighted) $k$-NN seemed reasonable (the best $k$-NN method solved 21.7% of the Flyspeck problems when combined with E) and quite complementary to premise selection based on naive Bayes. However, $k$-NN was weaker than the best naive-Bayes classifier (24.1% of problems solved when combined with E). The features that we use for characterizing Flyspeck formulas and measuring their similarity with $k$-NN consist of the symbols and normalized shared terms contained in the Flyspeck formulas. To give a concrete example (taken from Section 4.1 of [11]), the set of features characterizing the HOL theorem `DISCRETE_IMP_CLOSED`:[1]

```
∀s:real^N→bool e.
        &0 < e ∧ (∀x y. x IN s ∧ y IN s ∧ norm(y − x) < e ⟹ y = x)
        ⟹ closed s
```

is the following set of strings:

```
"real", "num", "fun", "cart", "bool", "vector_sub", "vector_norm",
"real_of_num", "real_lt", "closed", "_0", "NUMERAL", "IN", "=", "&0",
"&0 < Areal", "0", "Areal", "Areal^A", "Areal^A - Areal^A",
"Areal^A IN Areal^A->bool", " Areal^A->bool", "_0", "closed Areal^A->bool",
"norm (Areal^A - Areal^A)", "norm (Areal^A - Areal^A) < Areal"
```

Here `real` is a type constant, `IN` is a term constructor, `Areal^A->bool` is a normalized type, `Areal^A` its component type, `norm (Areal^A - Areal^A) < Areal` is an atomic formula, and `Areal^A - Areal^A` is its normalized subterm.

The simplest way how to measure the similarity of formulas to the new conjecture is to compute the overlap of their (sparse) feature vectors. One known property [2] of the $k$-NN that was neglected by our first implementation is however the sensitivity of $k$-NN to feature frequencies. For example, without additional weighting, the most common symbol (e.g., equality) has in such similarity function the same weight as the most rare symbol, which is clearly not desirable: overlap on the rarest symbol is much more significant than overlap on a symbol that is present everywhere.

The most common way how to weight (boolean-counted) features in text retrieval with respect to their frequency is the IDF (inverse document frequency) scheme [9]. This scheme weights a term $t$ in a collection of documents $D$ using the logarithm of the inverse of the term's frequency in the document collection:

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

For example, a term (symbol) contained only in one document (formula) will have weight $\log |D|$, a term contained in all of them will have weight $log(1) = 0$, and a term contained in half of the documents will have weight $log(2) = 1$. Apart from using the standard IDF, we have also found useful two other IDF-based weighting schemes, the (smoothed) inverse frequency:

$$\mathrm{IDF}_1(t, D) = \frac{1}{1 + |\{d \in D : t \in d\}|}$$

and quadratically scaled (smoothed) inverse frequency:

$$\mathrm{IDF}_2(t, D) = \frac{1}{(1 + |\{d \in D : t \in d\}|)^2}$$

[1] `http://mws.cs.ru.nl/~mptp/hol-flyspeck/trunk/Multivariate/topology.html#DISCRETE_IMP_CLOSED`

These three feature weighting schemes were combined with different values of $k$ nearest neighbors and (as usual) with ATPs run on different number of top-rated premises (using 30s time limit and the same hardware as in [11]). Table 1 shows the performance of the best 12 methods after these experiments, together with the performance of the best previous method (naive_bayes_0154e). The precision of the logarithmically scaled IDF is clearly the best, and in comparison with the non-logarithmic scaling, only relatively few nearest neighbors are needed. The previously best method (last line in Table 1) is improved by 103 problems, which is a 30% improvement. The proof data (denoted as ATP2 in Table 2) used for training the $k$-NNs were by a negligible margin older (worse, i.e., containing more irrelevant proof dependencies that we eventually prune) than the proof data (ATP3) used for the previously best method. The feature extraction method was the same (the standard one).

Table 1: The top 12 AI/ATP methods, together with the previously best.

| Method | Premises | Prover | Theorem (%) | Σ-SOTAC | Processed |
|---|---|---|---|---|---|
| 40-NN_IDF | 128 | E | 446 (31.43) | 4.24 | 1419 |
| 80-NN_IDF | 512 | E | 445 (31.36) | 4.25 | 1419 |
| 40-NN_IDF | 512 | E | 443 (31.22) | 4.20 | 1419 |
| 80-NN_IDF | 128 | E | 436 (30.73) | 4.22 | 1419 |
| 160-NN_IDF$_1$ | 128 | E | 423 (29.81) | 3.64 | 1419 |
| 300-NN_IDF$_1$ | 128 | E | 422 (29.74) | 3.50 | 1419 |
| 40-NN_IDF | 512 | V | 419 (29.53) | 3.09 | 1419 |
| 760-NN_IDF$_1$ | 128 | E | 417 (29.39) | 3.16 | 1419 |
| 160-NN_IDF$_1$ | 128 | E | 417 (29.39) | 3.49 | 1419 |
| 40-NN_IDF | 128 | V | 416 (29.32) | 3.01 | 1419 |
| 1200-NN_IDF$_1$ | 128 | E | 416 (29.32) | 4.29 | 1419 |
| 1000-NN_IDF$_2$ | 128 | E | 415 (29.25) | 3.21 | 1419 |
| naive_bayes | 154 | E | 343 (24.17) | 1.97 | 1419 |

**Method:** 40-NN_IDF means that 40 nearest neighbors are used with the standard IDF weighting.

**Prover:** V stands for Vampire [12].

**Σ-SOTAC** For each problem P solved by a system, its SOTAC for P is the inverse of the number of systems that solved P. Σ-SOTAC is the sum of a system's SOTAC over all problems.

Table 2 shows the newly computed 14-long *greedy covering sequence*, i.e., the joint performance of the (greedily) best combination of 14 methods, ordered by their inclusion in the greedy algorithm. While the logarithmic IDF scaling is obviously at the top, the linearly-scaled IDF provided many useful complementary predictive methods. The overall 14-method coverage went up from 39.0% (Table 14 in [11]) to 45.45%, i.e., by 16%. The frequency-scaling code in the $k$-NN implementation responsible for this improvement takes about 5 lines of Perl, and the whole sparse distance-weighted multiclass $k$-NN implementation takes about 200 lines of Perl code. Some large improvements in the ARLT domain are still very easy.

# 3 Better Strategies for Different Premise Selections

BliStr (Blind Strategymaker) [21] is a recently developed system that automatically develops strategies for E prover on a large set of related problems. Its main idea is to interleave (i)

Table 2: The greedy sequence including the new $k$-NNs.

| Method | Premises | Prover | Training data | Sum % | Sum |
|---|---|---|---|---|---|
| 40-NN_IDF | 128 | E | ATP2 | 31.43 | 446 |
| 760-NN_IDF$_1$ | 128 | V | ATP2 | 35.09 | 498 |
| naive_bayes | 128 | Z3 | ATP4 | 37.27 | 529 |
| 760-NN_IDF$_1$ | 32 | Z3 | ATP2 | 38.97 | 553 |
| naive_bayes | 184 | E | ATP3 | 40.31 | 572 |
| naive_bayes | 12 | E | ATP3 | 41.22 | 585 |
| 160-NN_IDF$_1$ | 128 | Z3 | ATP2 | 42.07 | 597 |
| naive_bayes | 512 | E | ATP0+HOL0 | 42.91 | 609 |
| 80-NN_IDF | 512 | V | ATP2 | 43.48 | 617 |
| 760-NN_IDF$_1$ | 512 | E | ATP2 | 43.97 | 624 |
| 40-NN_IDF$_1$ | 32 | V | ATP2 | 44.39 | 630 |
| 1000-NN_IDF$_2$ | 740 | V | ATP2 | 44.74 | 635 |
| 40-NN_IDF | 32 | E | ATP2 | 45.10 | 640 |
| 40-NN | 32 | Z | ATP2 | 45.45 | 645 |

iterated low-timelimit local search for new strategies on small sets of similar easy problems with (ii) higher-timelimit evaluation of the new strategies on all problems. The accumulated results of the global higher-timelimit runs are used to define and evolve the notion of "similar easy problems", and to control the selection of the next strategy to be improved.

BliStr was used to grow a set of E strategies for the Mizar@Turing competition.[2] The final improvement of the resulting strategy-scheduler (Epar) over the E's auto-mode was 25% on the Mizar@Turing competition problems. Epar has already been used for practically all AI/ATP experiments over Flyspeck, i.e., whenever we refer to E above, it was run using the Epar strategy-scheduler.

Since the Mizar@Turing pre-competition training problems were quite small (ca. 25 premises per problem), while the best methods in Table 2 use 128 premises, it seemed potentially rewarding to automatically evolve E strategies on such Flyspeck problem classes instead. This has been so far tried for the two top problem classes (i.e., classes of problems generated using the particular premise selection method described in the table) from Table 2 that use E: 40-NN_IDF_128_ATP2 and naive_bayes_184_ATP3. For each of them, the set of (randomly chosen) 1419 Flyspeck problems was further randomly divided into a training part (800 problems) and a testing part (619 problems). The 800 training problems were then used for ca. 30 hours of parallelized strategy evolution with Blistr. Since a major factor in ATP efficiency over problems with many premises is also a good SInE pre-selection,[3] the E parameters tunable by BliStr were extended in comparison to [21] to also evolve the SInE parameters. The starting set of strategies is the same for both problem sets. These were 15 strategies previously developed by BliStr that were giving good performance on Mizar/MPTP problems. These strategies were however slightly weaker than the old Epar, because SInE parameters have been heuristically added to Epar, while the 15 strategies do not contain any SInE parameters. We left it to the extended BliStr to develop good SInE parameters.

For 40-NN_IDF_128_ATP2, the 15 initial strategies cover (in 10s) 257 of the 800 training

[2] http://www.cs.miami.edu/~tptp/CASC/J6/Design.html#CompetitionDivisions

[3] SInE-based selection is obviously interacting in various ways with the premise selection done by naive Bayes and $k$-NN. This typically turns out to be a fruitful interaction of two different ranking methods, see [14].

problems. 12 BliStr instance were run in total (on a 32-core Intel machine), producing 353 strategies eventually covering 320 of the 800 training problems when run with a 10s time limit. For `naive_bayes_184_ATP3`, the 15 initial strategies cover (in 10s) 215 of the 800 training problems. 16 BliStr instances were run in total (on a 64-core AMD machine), producing 637 strategies, covering 253 of the 800 training problems with 10s time limit. To construct a new set of Epar strategies, we used again greedy algorithm that chooses the 14 strategies with the (greedily) best joint coverage. Note that using for example Minisat++ [5] for solving the set-cover problem optimally is not easy: it takes 400s for the 353 strategies (finding an 18-cover), and Minisat++ did not finish within one hour for the 637 strategies. The 14 new greedy strategies for `40-NN_IDF_128_ATP2` cover 313 training problems, and the 14 `naive_bayes_-184_ATP3` strategies cover 245 training problems. The new version of Epar (marked as E2 below) was then for each problem class evaluated with 30s overall time limit on the 64-CPU AMD machine both on the training and testing problems. The results are shown in Table 3 and Table 4. Note that in both cases the performance of the new Epar is obviously lower than the combined 10s performance of its underlying 14 strategies, because within its 30s overall time limit Epar gives only 2s to each of its strategies (this could obviously be made smarter, as is done in Vampire [12]).

Table 3: The strategy evaluation for `40-NN_IDF_128_ATP2`.

| `40-NN_IDF_128_ATP2` | OldEpar | NewEpar | Improvement (%) |
|---|---|---|---|
| Training | 254 | 290 | 36 (14.2) |
| Testing | 192 | 209 | 17 (8.9) |
| Total | 446 | 499 | 53 (11.9) |

Table 4: The strategy evaluation for `naive_bayes_184_ATP3`.

| `naive_bayes_184_ATP3` | OldEpar | NewEpar | Improvement (%) |
|---|---|---|---|
| Training | 173 | 208 | 35 (20.2) |
| Testing | 132 | 162 | 30 (22.7) |
| Total | 305 | 370 | 65 (21.3) |

The tables seem to suggest that some overfitting on the training problems took place for `40-NN_IDF_128_ATP2`, while the new Epar testing performance on `naive_bayes_184_ATP3` is even better than its performance on the training problems. It is interesting to note that SInE is used in 11 of the 14 new `40-NN_IDF_128_ATP2` strategies, and 12 of the 14 new `naive_bayes_-184_ATP3` strategies. Quite often the SInE parameters in the later complementary methods are quite severe, limiting the SInE recursion to 1 or 2. This could mean that the BliStr's loop run separately for the current premise-selection slice without interaction with other premise slices causes quite a lot of incompleteness. This might however be destroying some complementarity with the other premise-selection methods specialized in low premise numbers. An obvious remedy would be to evolve the strategies together on merged problem classes, and only later select the best strategies for the particular classes in a way that provides minimal overlap between them.

Finally, the new 14-long greedy covering sequence using the new Epars for `40-NN_IDF_-128_ATP2` and `naive_bayes_184_ATP3` was computed on our main evaluation machine. The

result is shown in Table 5. While originally the new Epar developed for `naive_bayes_184_ATP3` was indeed second in the greedy sequence, in the final version it became completely redundant, because we have also run the new strategies grown for `40-NN_IDF_128_ATP2` on the `440-NN_IDF` slices. This strengthened the `440-NN_IDF_128_ATP2` slice by 46 problems (12% improvement), and the `440-NN_IDF_512_ATP2` slice by 102 problems (33% improvement). This improvement was again quite unexpected, and it suggests that the overfitting suspected in Table 3 is not really a problem. The 14 methods of the final combination solve without any user interaction 46.86% of the Flyspeck problems.

Table 5: The new greedy sequence including the new Epars.

| Method | Premises | Prover | Training data | Sum % | Sum |
|---|---|---|---|---|---|
| 40-NN_IDF | 128 | E2 | ATP2 | 34.88 | 495 |
| 440-NN_IDF | 128 | E2 | ATP2 | 38.33 | 544 |
| 440-NN_IDF | 512 | E2 | ATP2 | 40.02 | 568 |
| $760\text{-NN\_IDF}_1$ | 32 | Z3 | ATP2 | 41.64 | 591 |
| naive_bayes | 128 | V | ATP2 | 42.84 | 608 |
| 40-NN | 512 | Z | ATP2 | 43.55 | 618 |
| $1000\text{-NN\_IDF}_2$ | 740 | V | ATP2 | 44.11 | 626 |
| naive_bayes | 64 | E | ATP3 | 44.67 | 634 |
| $160\text{-NN\_IDF}_1$ | 512 | Z3 | ATP2 | 45.10 | 640 |
| naive_bayes | 32 | Z3 | ATP0+HOL0 | 45.52 | 646 |
| naive_bayes | 512 | E | ATP0+HOL0 | 45.94 | 652 |
| 40-NN | 32 | E | ATP2 | 46.30 | 657 |
| 80-NN_IDF | 512 | V | ATP2 | 46.58 | 661 |
| $160\text{-NN\_IDF}_1$ | 128 | V | ATP2 | 46.86 | 665 |

**Prover:** E2 is the new Epar, while is the old Epar. Z3 is the Z3 [3] SMT solver.

**Training data:** $\text{ATP}_i$ is worse (older, less pruned) than $\text{ATP}_{i+1}$ . ATP0+HOL0 is a combination of proof data obtained from ATPs with the proof data extracted directly by tracking proof dependencies inside HOL.

# 4 Conclusion and Future Work

It is interesting that a similar treatment of features as in processing of natural language texts helps so significantly also for formal mathematical libraries. While mathematics in its extreme is undecidable, and has (theoretically constructed) parts that provably behave as random, it is clear that human-organized mathematics is very far from such randomness. While the automation that we currently have is still very far from the power of human mathematicians, the observance of such simple statistical laws may be providing evidence that the Penrose-style arguments about the "necessarily super-Turing" power of the human mathematical mind might more and more look like just another case of the *AI effect*.[4] The obvious future work concerning feature weighting is to re-run the machine learning methods tested in [14] on the MPTP2078 benchmark (in particular, the kernel-based MOR, and van Laarhoven's BiLi, whose

[4] `http://en.wikipedia.org/wiki/AI_effect`

random-projection mechanism should be quite sensitive to feature distribution) with such feature weightings.

Since ATPs are in some sense universal problems solvers, it should not be very surprising that their parameterization matters a lot. However, some of the results, like the 33% improvement of the `440-NN_IDF_512_ATP2` slice (which was not subject to any direct tuning), were quite unexpected. While we have added the SInE parameters to BliStr, there are still many more E parameters that could be further tuned. A particularly interesting challenge is to grow and guess suitable term orderings for similar classes of problems.

In general the experimental results show that large improvements can be still achieved in the ARLT domain by quite simple methods and simple technology transfer. It seems that this AI domain is still wide open to a number of techniques that could further considerably improve the strenth of automation for large-theory mathematics.

# 5 Acknowledgments

While we have been for long time without much action discussing experiments with various state-of-the-art feature preprocessing methods, it was Jasmin Blanchette and his immediate interest in improving the machine learning for Sledgehammer [13] that led the second author to propose and quickly implement feature weighting in $k$-NN as a reasonable remedy to some of the problems discussed by Jasmin. While the TF-IDF scaling is reasonably known in the text retrieval domain, the idea of using the inverse document frequency as a formula classifier is in some sense also at the heart of the SInE axiom-selection heuristic [8] developed independently by Kryštof Hoder.

# References

[1] Jesse Alama, Tom Heskes, 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.

[2] Fabrice Colas and Pavel Brazdil. Comparison of svm and some older classification algorithms in text classification tasks. In Max Bramer, editor, *Artificial Intelligence in Theory and Practice*, volume 217 of *IFIP International Federation for Information Processing*, pages 169–178. Springer US, 2006.

[3] Leonardo Mendonça de Moura and Nikolaj Bjørner. Z3: An Efficient SMT Solver. In C. R. Ramakrishnan and Jakob Rehof, editors, *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[4] Sahibsingh A. Dudani. The distance-weighted k-nearest-neighbor rule. *Systems, Man and Cybernetics, IEEE Transactions on*, SMC-6(4):325–327, 1976.

[5] Niklas Eén and Niklas Sörensson. An extensible SAT-solver. In Enrico Giunchiglia and Armando Tacchella, editors, *SAT*, volume 2919 of *LNCS*, pages 502–518. Springer, 2003.

[6] Thomas C. Hales. Introduction to the Flyspeck project. In Thierry Coquand, Henri Lombardi, and Marie-Françoise 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.

[7] 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.

[8] Krystof Hoder and Andrei Voronkov. Sine qua non for large theory reasoning. In Nikolaj Bjørner and Viorica Sofronie-Stokkermans, editors, *CADE*, volume 6803 of *LNCS*, pages 299–314. Springer, 2011.

[9] Karen Sprck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of Documentation*, 28:11–21, 1972.

[10] Cezary Kaliszyk and Josef Urban. Automated reasoning service for HOL Light. In Jacques Carette, David Aspinall, Christoph Lange, Petr Sojka, and Wolfgang Windsteiger, editors, *MKM/Calculemus/DML*, volume 7961 of *Lecture Notes in Computer Science*, pages 120–135. Springer, 2013.

[11] Cezary Kaliszyk and Josef Urban. Learning-assisted automated reasoning with Flyspeck. *Journal of Automated Reasoning*, 2014. `http://dx.doi.org/10.1007/s10817-014-9303-3`.

[12] Laura Kovács and Andrei Voronkov. First-order theorem proving and Vampire. In Natasha Sharygina and Helmut Veith, editors, *CAV*, volume 8044 of *Lecture Notes in Computer Science*, pages 1–35. Springer, 2013.

[13] Daniel Kühlwein, Jasmin Christian Blanchette, Cezary Kaliszyk, and Josef Urban. MaSh: Machine learning for Sledgehammer. In Sandrine Blazy, Christine Paulin-Mohring, and David Pichardie, editors, *Proc. of the 4th International Conference on Interactive Theorem Proving (ITP'13)*, volume 7998 of *LNCS*, pages 35–50. Springer, 2013.

[14] 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.

[15] Lawrence C. Paulson and Jasmin Blanchette. Three years of experience with Sledgehammer, a practical link between automated and interactive theorem provers. In *8th IWIL*, 2010. Invited talk.

[16] Lawrence C. Paulson and Kong Woei Susanto. Source-level proof reconstruction for interactive theorem proving. In Klaus Schneider and Jens Brandt, editors, *TPHOLs*, volume 4732 of *LNCS*, pages 232–245. Springer, 2007.

[17] Stephan Schulz. E - A Brainiac Theorem Prover. *AI Commun.*, 15(2-3):111–126, 2002.

[18] Geoff Sutcliffe. The 6th IJCAR automated theorem proving system competition - CASC-J6. *AI Commun.*, 26(2):211–223, 2013.

[19] Josef Urban. MoMM - fast interreduction and retrieval in large libraries of formalized mathematics. *Int. J. on Artificial Intelligence Tools*, 15(1):109–130, 2006.

[20] Josef Urban. An Overview of Methods for Large-Theory Automated Theorem Proving (Invited Paper). In Peter Höfner, Annabelle McIver, and Georg Struth, editors, *ATE Workshop*, volume 760 of *CEUR Workshop Proceedings*, pages 3–8. CEUR-WS.org, 2011.

[21] Josef Urban. BliStr: The Blind Strategymaker. *CoRR*, abs/1301.2683, 2013.

[22] Josef Urban, Piotr Rudnicki, and Geoff Sutcliffe. ATP and presentation service for Mizar formalizations. *J. Autom. Reasoning*, 50:229–241, 2013.

[23] Josef Urban and Geoff Sutcliffe. Automated reasoning and presentation support for formalizing mathematics in Mizar. In Serge Autexier, Jacques Calmet, David Delahaye, Patrick D. F. Ion, Laurence Rideau, Renaud Rioboo, and Alan P. Sexton, editors, *AISC/MKM/Calculemus*, volume 6167 of *LNCS*, pages 132–146. Springer, 2010.

[24] Josef Urban, Geoff Sutcliffe, Petr Pudlák, and Jiří Vyskočil. MaLARea SG1 - Machine Learner for Automated Reasoning with Semantic Guidance. In Alessandro Armando, Peter Baumgartner, and Gilles Dowek, editors, *IJCAR*, volume 5195 of *LNCS*, pages 441–456. Springer, 2008.