Search Spaces for Theorem Proving Strategies

David A. Plaisted

UNC Chapel Hill
Chapel Hill, UNC, North Carolina
USA
plaisted@cs.unc.edu

The leading theorem provers are so complicated that it can be difficult to know what is influencing their performance and why an added feature is helping or hurting. Also, the burden of developing a state of the art theorem prover to test ideas is enormous. It would be desirable to have an implementation-independent way to measure the performance of various strategies such as hyper-resolution and many others. In the analysis of algorithms it is possible to derive asymptotic performance bounds independent of any particular implementation, and such an analysis can provide helpful insight into the performance of an algorithm and help in the development of new algorithms. Something like this in theorem proving is desirable.

Suppose Professor R.E.Solution implements Prover A and shows that Factor X improves its performance. Then what do we really know? Could it be that Professor Solution adjusted the coefficients of the prover to make this result possible? Could the prover be tuned to get the desired result? Suppose Professor T.A.Bleaux then comes along and implements Prover B and on this prover Factor X makes the performance worse? Do we really know that this is not possible? And in the long run what is gained by the implementation of a prover? Eventually the writers and maintainers of the prover will pass off the scene and the prover is likely to be abandoned.

In algorithms theory, tremendous advances have been made purely by deriving asymptotic bounds without any implementation. For example, for the graph isomorphism problem, bounds of $2^{O(\sqrt{n}\log^2 n)}$, $2^{O(\sqrt{n\log n})}$, and $2^{O((\log n)^c)}$ for some fixed $c > 0$ were obtained [5]. At least one of these bounds relies on the classification of finite simple groups, a deep mathematical theorem.

There have already been theoretical results concerning the complexity of theorem proving, but they are based on proof size, such as Gentzen's celebrated result [2] about cut elimination and Haken's result [3] that resolution proofs from the pigeonhole formulas are exponential in size. However the total number of clauses generated before finding a proof is a more relevant measure when analyzing the performance of theorem proving strategies.

We propose a measure based on the search space, that is, the number of clauses generated by various strategies before finding a proof. This idea has been presented before by the author, for example [4], but was never made precise. Define a *theorem proving strategy* Σ to be a collection $\Sigma^{\vdash}$ of inference rules, possibly including resolution, factoring, instantiation, paramodulation, demodulation, and applications of DPLL-CDCL for instance-based strategies, together with a computable function Σ^* from sets S of clauses to finite or infinite sequences of clauses $C_1, C_2, \cdots, C_n, \cdots$. To be precise, a computable function Σ^* from S to a finite or infinite sequence $C_1, C_2, C_3, \cdots$ of length k is a computable function Σ^* such that $\Sigma^*(S, i) = C_i$ for all $i \leq k$ such that $i < \infty$. We can denote C_i as $\Sigma^*(S)_i$. Also, each clause C_i must either be in S or derivable from previous clauses in the sequence by a single application of one of

the inference rules of Σ. The intuition is that a prover could generate the sequence of clauses $C_1, C_2, \cdots$ in order. Frequently we write $\Sigma(S)$ instead of $\Sigma^*(S)$.

Let $|\Sigma(S)|$ be the minimum i such that $\Sigma(S)_i$ is the empty clause, if such an i exists, else ∞. We propose that $|\Sigma(S)|$ is a suitable implementation independent measure of the complexity of the strategy Σ. Note that if Σ is sound and complete then for all S, $|\Sigma(S)| < \infty$ iff S is unsatifiable.

In general, let $||U||$ be the length of a set U when it is written out as a character string using a natural encoding. Now, the problem is that we can't give bounds on $|\Sigma(S)|$ that are polynomial or exponential in terms of $||S||$ because the theorems of first-order logic are partially decidable but not decidable. If we could bound $|\Sigma(S)|$ by any recursive (computable) function of $||S||$ then it would give a decision procedure for theoremhood in first-order logic. So how do we give a complexity bound for $|\Sigma(S)|$ that can be used to compare various strategies?

This can be done using a non-inference based measure of the complexity of a theorem. For unsatisfiable sets of S of clauses, define a *Herbrand set* to be an unsatisfiable set of ground instances of S. Let U be a Herbrand set such that $||H(S)||$ is minimal. Define $||S||_H$ to be $||U||$. Then for unsatisfiable clause sets, one can take $||S||_H$ as a measure of the difficulty of showing that S is unsatisfiable. Now it is possible to give recursive bounds of $|\Sigma(S)|$ in terms of $||S||_H$ for unsatisfiable clause sets S and these bounds can be used as an implementation independent measure of the complexity of the strategy Σ.

Some results of this nature have been given [4] by the author in a previous work. One of the bounds derived there for resolution and factoring is $O(H^{2^{H^2}})$, writing H for $||S||_H$. Perhaps this should be $(2H|S|)^{2^{2(H^2)}}$. A bound of $O((c+H)^H)$ is also given there for a version of DPLL appllied to first-order logic where c is the number of predicate, function, and constant symbols. The conclusion given there is that DPLL-type or instance-based methods are better than resolution for very hard problems. Because one never needs a clause C with $||C|| > H$ after factoring, and one never needs clauses C with $||C|| > 2H$ before factoring, the number of clauses in a resolution refutation can be bounded by $(c+2H)^{2H}$ (allowing for up to $2H$ variables) giving a trivial single exponential bound on $|\Sigma(S)|$ in terms of H for resolution and factoring. An advantage of resolution is that it incorporates the paramodulation and demodulation rules for equality. There needs to be more work to find ways to incorporate equality into instance-based methods.

There are also other ways one can bound the work. Other bounds could be based on the sum of the sizes of the clauses in a sequence $C_1, C_2, \cdots$ or some modified measure that reduces the bound because common subterms may only be stored once. If one allows a $delete(i)$ operation that deletes clause C_i (perhaps it is subsumed) then one can have a bound based on the number of clauses stored at one time. Bounds in terms of $||S||_H$ can also be defined based on the time and space used by a prover.

It would be desirable to extend such work to other inference systems such as sequent style, tableau style, and Hilbert style systems. For instance-based strategies, there may be occasional calls to DPLL; if desired, these can be considered as a single inference rule producing either the empty clause or True if DPLL returns a result of satisfiable.

Theoretical results about bounds do have limitations. For example, the satisfiability problem is NP complete, but implementations of satisfiability testers frequently run quickly for large problems, and problems in other domains are often translated into satisfiability so that satisfiability techniques can be applied to them. Implementations of resolution for propositional problems generally are inefficient compared to techniques based on DPLL [1]. Even this does not seem to be easy to prove theoretically; the worst case bound for DPLL for n variables is $O(2^n)$ but there can be as many as 3^n clauses over n atoms, and attempting all pairwise resolu-

tions could raise the bound to 9^n. However, if subsumed clauses are deleted, then the number of clauses that can exist at one time without one subsuming another is at least $2^{n/2}\binom{n}{n/2}$ (the number of clauses having exactly $n/2$ literals), but the exact bound is not known to this author.

References

[1] Martin Davis, George Logemann, and Donald W. Loveland. A machine program for theorem-proving. *Commun. ACM*, 5(7):394–397, 1962.

[2] Gerhard Gentzen. Untersuchungen über das logische schließen. ii. *Mathematische Zeitschrift*, 39(1), 1935.

[3] Armin Haken. The intractability of resolution. *Theor. Comput. Sci.*, 39:297–308, 1985.

[4] David A. Plaisted. History and prospects for first-order automated deduction. In *25th International Conference on Automated Deduction*, pages 3–28, Berlin, Germany, August 1-7, 2015.

[5] Wikipedia contributors. Graph isomorphism problem — Wikipedia, the free encyclopedia, 2021. [Online; accessed 2-July-2021].