ZRes: The old Davis-Putnam procedure meets ZBDD

Philippe Chatalic and Laurent Simon

Laboratoire de Recherche en Informatique,
U.M.R. CNRS 8623,
Université Paris-Sud, 91405 Orsay Cedex, France
{chatalic, simon}@lri.fr

Abstract. ZRes is a propositional prover based on the original procedure of Davis and Putnam, as opposed to its modified version of Davis, Logeman and Loveland, on which most of the current efficient SAT provers are based. On some highly structured SAT instances, such as the well known Pigeon Hole and Urquhart problems, both proved hard for resolution, ZRes performs very well and surpasses all classical SAT provers by an order of magnitude.

1 The DP and DLL algorithms

Stimulated by hardware progress, many more and more efficient SAT solvers have been designed during the last decade. It is striking that most of the complete solvers are based on the procedure of Davis, Logeman and Loveland (DLL for short) presented in 1962 [11]. The DLL procedure may roughly be described as a backtrack procedure that searches for a model. Each step amounts to the extension of a partial interpretation by choosing an assignment for a selected variable. The success of this procedure is mainly due to its space complexity, since making choices only results in simplifications. However, the number of potential extensions remains exponential. Therefore, if the search space cannot be pruned by clever heuristics, this approach becomes intractable in practice.

The picture is very different with DP, the original Davis-Putnam algorithm [3]. DP is able to determine if a propositional formula f, expressed under conjunctive normal form (CNF), is satisfiable or not. Assuming the reader is familiar with propositional logic, DP may be roughly described as follows [9]:

I. Choose a propositional variable x of f.
II. Replace all the clauses which contain the literal x (or $\neg x$) by all binary resolvents (on x) of these clauses (cut elimination of x), and remove all subsumed clauses.
III. a. If the new set of clauses is reduced to the empty clause, then the original set is unsatisfiable.
b. If it is empty, then the original formula is satisfiable.
c. Otherwise, repeat steps I-III for this new set of clauses.

As opposed to DLL, DP avoids making choices by considering the two possible instantiations of a variable simultaneously. It amounts to a sequence of cut eliminations. Since the number of clauses generated at each step may grow exponentially, it is widely acknowledged in the literature as inefficient, although no real experimental study has been conducted to confirm this point. Dechter and Rish [5] have first pointed out some instances for which DLL is not appropriate and where DP obtains better results. A more comprehensive experimentation has been conducted in [2], to evaluate DP on a variety of instances. If this study confirms the superiority of DLL on random instances, it also shows that for strucured instances, DP may outperform some of the best DLL procedures. Substantial progress for DLL are due to better heuristics. For DP, significant improvements are possible thanks to efficient data structures for representing very large sets of clauses.

Several authors have pointed out that resolution-based provers (like DP and DLL) are intrinsically limited, since they have found instances that require an exponential number of resolution steps to be solved (e.g. [9, 10, 14]). This is the case for the Pigeon Hole [10] and for the Urquhart problem [14]. They suggest that more powerful proof systems have to be used practically to solve such problems efficiently. However, all these results are based on the implicit hypothesis that successive resolutions in DP and DLL are performed one by one.

This paper presents the ZRes system, which is an implementation of the DP algorithm that is able to perform several resolutions in a single step. As a result, ZRes is able to solve instances of such hard problems much more efficiently than the best current DLL provers.

2 Reviving DP

The crucial point of DP is step II, which tends to generate a very large number of clauses. Eliminating subsumed clauses at this step induces a significant overhead but still pays off.

Efficient data structures In [2], Trie structures are used to represent sets of clauses. Until now, they seem to remain the state-of-the-art data structures for subsumption checking [15, 4]. Tries allow the factorization of clauses beginning in the same way, according to a given order on literals. In ZRes we further generalize this principle, to allow the factorization of the end as well as the beginning of clauses simultaneously. Sets of clauses are thus represented by means of directed acyclic graphs (DAG) instead of trees (by Tries).

Using DAG to represent a boolean formula has been intensively investigated in many works on binary decision diagrams (BDD) [1, 7]. Many variants of B-DD have been proposed but all attempt to compute the BDD encoding of the formula, expressed in Shannon normal form. From the SAT point of view and since the resulting BDD characterizes the formula validity and satisfiability, this construction is *de facto* more difficult than testing for satisfiability.

The approach followed by ZRes is quite different since, instead of computing the Shannon normal form, we use here BDD-like structures only to represent sets of clauses. Practically, the set of clauses is represented by a ZBDD (a variant of BDD [12]), proved useful for manipulations of large sets. Sets are represented by their characteristic (boolean) functions and basic set operations can thus be performed as boolean operations on ZBDD. Moreover, it has been shown that the size of such a ZBDD is not directly related to the size of the corresponding set. Since the cost of basic set operations only depend on the size of the considered ZBDD, this hints to performing the cut elimination step of the DP algorithm directly at the set level.

Another way to represent f in the cut elimination step of DP is to factorize x and $\neg x$ among its clauses. The formula f can then be rewritten as $(x \vee f_x^+) \wedge (\neg x \vee f_x^-) \wedge f_{x'}$, where f_x^+ (resp. f_x^-) is the CNF obtained from the set of clauses containing x (resp. $\neg x$), after factorization, and where $f_{x'}$ denotes the set of clauses containing neither x nor $\neg x$. The second step of the algorithm then amounts to put the formula $(f_x^+ \vee f_x^-) \wedge f_{x'}$ into CNF. This can be done in 3 stages. First, distribute the set of clauses f_x^+ over f_x^-. Second, eliminate tautologies and subsumed clauses from the resulting clauses. Third, compute the union of the remaining clauses with those of $f_{x'}$, while deleting subsumed clauses. The two first stages could be performed successively, using standard operations on ZBDD. However, the ZBDD used in ZRes have a special semantics and thus, a more efficient algorithm, called *clause-distribution*, can be designed. This operation guarantees that, during the bottom-up construction of the result, each intermediate ZBDD is free of tautologies and subsumed clauses. Tautologies are eliminated on the fly and subsumed clauses are deleted by a set difference operation, at each level. Similarly, in the third stage of the cut elimination, subsumed clauses may be deleted while computing the union of the two sets of clauses. This new algorithm takes full advantage of the data structure used to represent sets of clauses.

3 Experimental Results

ZRes[1] is written in C, using the Cudd package [13] which provides us with basic ZBDD operations as well as useful dynamic reordering functions. We have tested ZRes on two classes of hard problems for resolution: Hole and Urquhart. Those tests have been performed on a Linux Pentium-II 400MHz[2] with 256MB. Our results are compared, when possible, with those of two DLL implementations: Asat [8], which is a good-but-simple DLL implementation, and Sato 3.0 [15], which includes many optimizations and recent improvements, such as backjumping and conflict memorization. Cpu times are given in seconds. We assume that an instance that cannot be solved in less than 10000 seconds counts for 10000.

The Hole problem The following table describes the obtained results on different instances of the Hole problem by Asat, Sato and ZRes. For ZRes, the

[1] ZRes is available at http://www.lri.fr/~simon/research/zres.

[2] On the DIMACS [6] machine scale benchmark, our tests have granted this machine a user time saving of 305%, in comparison with the Sparc10.41, given as a reference.

std and *cd* columns describe respectively the time obtained without (resp. with) the *clause-distribution* operation.

Instances	Var. Nb.	Cl. Nb.	ASAT	SATO	ZRES *std*	ZRES *cd*
Hole-09	90	415	11.94	8.90	1.87	1.01
Hole-10	110	561	141.96	80.94	3.26	1.61
Hole-11	132	738	1960.77	7373.65	5.76	2.65
Hole-12	156	949	10000	10000	10.18	4.06
Hole-20	420	4221	–	–	654.8	69
Hole-30	930	13981	–	–	10000	1102
Hole-40	1640	32841	–	–	–	9421

ZRES clearly surpasses both ASAT and SATO. While DLL algorithms can't solve instances of Hole-n for $n > 11$, ZRES manages to solve much larger instances[3]. As we can see, the speedup induced by the *clause-distribution* operation is significant on such instances.

Our experiments have shown that this problem is very sensitive to the heuristic function used to choose the cut variable. Surprisingly, the best results were obtained using a heuristic function that tends to maximize the number of clauses produced by the cut. Other heuristics, such as in [2], did not allow to solve those Hole instances.

The Urquhart problem Urquhart has described a class of problems based on properties of expander graphs [14]. Actually, each Urq-n is a class of problems where the number of clauses and variables is not fixed but only bounded to a specific interval. In the last table, MnV (resp. MnC) denotes the mean number of variables (resp. clauses) for a set of instances of a given class. Contrary to Hole, Urquhart problem does not seem sensitive to the heuristics used. The results of ASAT and SATO on 100 Urq-3 instances attest the hardness of these problems:

System	Total cpu time	#resolved	Mean cpu time (resolved)
ASAT	404 287	69	1366
SATO	776 364	26	1398
ZRES *cd*	69.2	100	0.69

Solving instances for greater values of n seems out of the scope of these systems. On the other hand, ZRES performs quite well on such instances. The following table gives the mean time on 1000 instances for greater values of n. Note that we do not give the *std* time because the speedup due to the *clause-distribution* operation is not relevant for this problem.

[3] We even solved the Hole-55 instances, with 3080 variables and 84756 clauses, in less than 2 days.

Instances	MnV	MnC	Mean cpu-time
Urq-3	42	364	0.57
Urq-4	77	705	1.72
Urq-5	123	1143	4.25
Urq-6	178	1665	8.88
Urq-7	242	2299	16.5
Urq-8	317	3004	29.6
Urq-9	403	3837	48.8

About the compression power of ZBBD The previous examples illustrate quite well that DP, associated with the ZBDD encoding of sets of clauses, may in some cases be more effective than DLL. The experiment on the Hole-40 shows that for some of the cut eliminations, the number of clauses corresponding to f_x^+ and f_x^- may exceed 10^{60}. Clearly, the result of such a cut could not be computed efficiently without using such an extremely compact data structure. The ability of ZBDD to capture redundancies in sets of clauses suits the DP algorithm particularly well. Indeed, additional redundancies are produced during each cut elimination, when each clause of f_x^+ is merged with each clause of f_x^-. Moreover, unlike in random instances, we think that such redundancies may also be found in structured instances, corresponding to real-world problems.

In order to appreciate the compression power of ZBDD structures, it is interesting to consider *level of compression* which may be characterized by the ratio $nb\ of\ literals/nb\ of\ nodes$. We have recorded its successive values on 1000 random 3-SAT instances of 42 variables and 180 clauses, on wich DP is known to be a poor candidate [2]. For such instances, the initial value of the ratio is about 2, then it increases up to 6.15 and eventually decreases together with the number of clauses. In contrast, on Hole-40, this ratio varies from 10 to more that 10^{60}. Similarly, on Urq-10 it may exceed 10^{64}. Pigeon and Urquart classes however correspond to extreme cases. We have also tested ZRes on some other instances of the SAT DIMACS base [6]. Results are particularly interesting. For some instances the compression level is much more important than for random instances (more than 10^8), while on others, like *ssa* or $flat-50$ it is very close to that of random instances. It is however striking that the latter instances, even if they correspond to concrete problems, have been generated in a random manner. This seems to confirm that our hypothesis (structured problems generally have regularities) is well founded.

4 Conclusion

Dechter and Rish [5] were the first to revive the interest in the original Davis-Putnam procedure for SAT. But DP also proves useful for knowledge compilation or validation techniques, which was our initial motivation [2]. The introduction of ZBDD brings significant improvements allowing ZRes to deal with huge sets of clauses. It leads us to completely reconsider the performances of the cut elimination, which can be performed independently of the number of handled clauses.

It is thus able to solve two hard problems out of the scope of other resolution-based provers. Although such examples might be considered somewhat artificial, their importance in the study of the complexity of resolution procedures must not be forgotten. On other examples, such like DIMACS ones, results are not so good, but important compression level, due to ZBDD, can be observed on real-world instances. The strength of ZRes definitely comes from its ability to capture regularities in sets of clauses. Although it has a no chance to compete on random instances, which lack such regularities, it might be a better candidate for solving real-world problems.

Further improvements are possible. The Hole example pointed out that standard heuristics for DP are not always appropriate for ZRes. We are studying new heuristics, based on the structure of ZBDD rather than on what they represent. One may also investigate more adapted reordering algorithms, which take advantage of the particular semantics of the ZBDD used in ZRes. Eventually, DP and DLL may be considered as complementary approaches. An interesting idea is to design an hybrid algorithm integrating both DP and DLL in ZRes.

References

1. R.E. Bryant. Graph - based algorithms for boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.
2. Ph. Chatalic and L. Simon. Davis and putnam 40 years later: a first experimentation. Technical Report 1237, LRI, Orsay, France, 2000. *Submitted to the Journal of Automated Reasoning.*
3. M. Davis and H. Putnam. A computing procedure for quantification theory. *Journal of the ACM*, pages 201–215, 1960.
4. Johan de Kleer. An improved incremental algorithm for generating prime implicates. In *AAAI'92*, pages 780–785, 1992.
5. R. Dechter and I. Rish. Directional resolution: The davis-putnam procedure, revisited. In *Proceedings of KR-94*, pages 134–145, 1994.
6. The DIMACS challenge benchmarks. ftp://ftp.rutgers.dimacs.edu/challenges/sat.
7. R. Drechsler and B. Becker. *Binary Decision Diagram: Theory and Implementation.* Kluwer Academic Publisher, 1998.
8. Olivier Dubois. Can a very simple algorithm be efficient for SAT? ftp://ftp.dimacs.rutgers.edu/pub/challenges/sat/contributed/dubois.
9. Zvi Galil. On the complexity of regular resolution and the davis-putnam procedure. *Theorical Computer Science*, 4:23–46, 1977.
10. A. Haken. The intractability of resolution. *Theorical Computer Science*, 39:297–308, 1985.
11. G. Logeman M. Davis and D. Loveland. A machine program for theorem-proving. *Communications of the ACM*, pages 394–397, 1962.
12. S. Minato. Zero-suppressed bdds for set manipulation in combinatorial problems. In *30th ACM/IEEE Design Automation Conference*, 1993.
13. F. Somenzy. Cudd release 2.3.0. http://bessie.colorado.edu/~fabio.
14. A. Urquhart. Hard examples for resolution. *Journal of the ACM*, 34:209–219, 1987.
15. Hantao Zhang. SATO: An efficient propositional prover. In *CADE-14*, LNCS 1249, pages 272–275, 1997.