

Likely Near-term Advances in SAT Solvers¹

Heidi E. Dixon
Matthew L. Ginsberg²
Andrew J. Parkes

CIRL
1269 University of Oregon
Eugene, OR 97403-1269
(541) 346-0471 (phone)
(541) 346-0474 (FAX)
{dixon,ginsberg,parkes}@cir1.uoregon.edu

Abstract

Recent work on microprocessor testing and verification has begun to consider the possible applicability of general-purpose satisfiability tools to this domain. In this paper, we will discuss some of the likely upcoming representational enhancements to these solvers, and the extent to which these representational improvements are likely to impact work in MTV.

¹Presented at the MTV02 workshop, June 2002 (<http://ece.tamu.edu/MTV/>).

© CIRL 2002.

²Contact author.

1 Introduction

Recent years have seen the development of increasingly powerful general-purpose satisfiability engines, and the performance level of these engines is sufficient to provide valuable solutions in the domain of formal verification [6, and others]. The satisfiability solver that currently provides the best solutions on formal verification problems is called CHAFF [18] and comes from the Davis-Putnam-Logemann-Loveland (DPLL) family of algorithms.

Two necessary ingredients for a successful systematic satisfiability solver are fast propagation and effective conflict analysis. Propagation is a technique that takes a variable assignment and derives new consequential assignments. It consumes the bulk of computational resources in DPLL style algorithms and is consequentially one of the most important factors in the efficiency of such solvers. DPLL style algorithms use *unit propagation* which is an incomplete but very fast form of propagation. The recent addition of data structures such as the watched literal technique [18, 23] increases the speed of unit propagation which in turn allows solvers to manage larger constraint databases.

Conflict analysis (also called learning) [22] is a way of generating new valid constraints in response to a contradictory set of assignments. The new constraint eliminates bad sets of assignments and reduces the size of the remaining search space. Learning methods are impractical without methods to bound the memory space used by learning and progress in this area has played an important role in recent performance advances [3, 9, 11]. The CHAFF algorithm builds on previous solvers and makes contributions in both of these areas.

In this paper we discuss a direction that systematic solvers appear likely to take in the near future. The direction is towards adapting DPLL style solvers to use stronger and more concise representations of constraints. Most current solvers (CHAFF included) represent problem constraints as lists of disjunctions. A stronger representation is one that allows us to write a problem (exponentially) more concisely. The main benefit of this comes during conflict analysis when a single learned constraint can potentially be equivalent to an exponential number of disjunctions. This can provide a powerful tool for pruning the search space.

The challenge is to find a representation that provides the benefit of more effective learning without seriously impacting the performance of the propagation methods. We will present two representations that appear to meet this criteria.

2 Pseudo-Boolean representations

The first representational change we will discuss involves the move from Boolean to pseudo-Boolean (PB) representation. A Boolean SAT problem is typically represented as a conjunction of clauses, each of which is a disjunction of literals such as

$$a \vee b \vee \neg c \tag{1}$$

and the goal is to assign values to each of the letters so that every clause in the problem is satisfied. Of course, (1) can be rewritten

$$a + b + \bar{c} \geq 1$$

where a, b, c are now numeric variables required to be 0 or 1 and \bar{c} is shorthand for $1 - c$. A clause, instead of being a particular disjunction, is now represented as

$$\sum_i l_i \geq 1 \tag{2}$$

where each literal l_i is either an atom v_i or its negation \bar{v}_i .

To extend this representation from Boolean to *pseudo*-Boolean, we merely allow clauses to be of the form

$$\sum_i w_i l_i \geq k$$

where the w_i (the weights assigned to the various literals) and k are real numbers, which we can take to be integral without loss of generality. As before, all of the variables in the problem continue to be required to be either 0 or 1.

As an example, we can now say

$$\sum_i v_i \geq n \tag{3}$$

for any specific n , indicating that at least n of the v_i are true. There is no sub-exponential encoding of (3) using pure Boolean rules that does not involve the introduction of new variables.

PB itself is not a new representation; it is just the specialization of integer programming to the case of Binary Integer Programming (BIP) [19]. However, we expect that MTV problems expressed in PB will have similar characteristics to the SAT formulation, in that the problems can be naturally represented using large numbers of relatively simple constraints. Since IP solvers (based on using linear relaxations and separating cuts) are generally much less effective than DPLL on SAT, we expect that this will also be true for PB formulations.

We have two issues to address. Firstly, we need to ensure that PB will not lose the computational advantages of DPLL; much effort has gone into developing efficient data structures and heuristics for DPLL and these should not be discarded lightly. In fact, we have found that there is only a modest computational cost to changing from SAT to pseudo-Boolean – the mechanisms used in a DPLL solver such as CHAFF can be translated and used in pseudo-Boolean with little computational overhead.

Secondly, we will argue that use of PB does not merely preserve the advantages of solvers such as CHAFF, but can potentially augment their constraint learning mechanisms in significant ways. A single PB clause can represent an exponential number of SAT disjunctions; as a result, learning PB clauses rather than just disjunctions can potentially lead to exponential savings in terms search nodes expanded.

2.1 Fast Unit Propagation in PB

One of the most important factors in determining the raw speed of a SAT solver is unit propagation, as this is the step that consumes the bulk of the computational resources in virtually all DPLL-based algorithms. In unit propagation, one takes a clause

$$\bigvee_i l_i \tag{4}$$

for which every literal has already been assigned the value false except one. This final literal must now be assigned the value true if the clause itself is to be satisfied.

In the PB case, suppose that we have a clause

$$\sum_i w_i l_i \geq k$$

and that the sum of the w_i for the literals that have been valued true is v while the sum of the w_i for the unvalued literals is u . Now any unvalued l_i for which $v + u - w_i < k$ can safely be required to be true at this point. Provided that the literals in the clause are sorted by weight, the cost of finding these l_i involves at most a partial walk along the length of the clause, increasing the cost of the unit propagation only modestly. It is also possible to transfer vital implementational “engineering” issues such as watched literals over to PB. Finally, for many domains the majority of PB constraints will likely be cardinality constraints in which all weights are one; $\sum_i l_i \geq k$. This type of PB constraint can be managed with no additional cost at all. Overall, the raw speed of unit propagation should be decreased only modestly by the use of PB clauses.

2.2 Advantages of PB for Constraint Learning

One of the critical factors in an effective DPLL algorithm is conflict-driven learning. Whenever a clause becomes unsatisfied we backtrack but also learn a new clause that is designed to prevent the search making closely related mistakes in the future. In most recent implementations the clause is generated by means of analysis of an implication graph, with a cut through the graph selecting the literals of the learned disjunction. Such a graph could also be produced from PB constraints, and a disjunction could be learned. However, our goal is that the conflict analysis should strive to produce not mere disjunctions but rather PB clauses. The intuition behind this is that a single PB clause can correspond to an exponential number of different disjunctions. Learning such a PB clause from a conflict and inserting it into the constraint database increases the chance that at a subsequent point in the analysis a propagation will occur and reduce the amount of search remaining.

Hence, conflict analysis or other forms of learning within DPLL should be based on a method that can produce PB clauses. Now note that the conflict analysis methods within CHAFF and all other DPLL-based SAT algorithms are all equivalent to resolution: Variables involved in the conflict are resolved away from the set of conflicting clauses. The key to learning PB clauses is thus to replace resolution with an inference scheme better suited to the pseudo-Boolean environment. The simplest choice for such an inference scheme is cutting planes (CP) [19]. CP inference is based on taking linear combinations of constraints together with rounding steps that exploit the fact that the variables are constrained to be integer valued.

Rather than give a formal description, we give a representative example. Consider the system

$$\begin{aligned} x_1 + x_2 &\geq 1 \\ x_1 + x_3 &\geq 1 \\ x_2 + x_3 &\geq 1 \end{aligned} \tag{5}$$

Adding together these constraints gives

$$2x_1 + 2x_2 + 2x_3 \geq 3$$

or

$$x_1 + x_2 + x_3 \geq 3/2$$

However, the left-hand side is a sum of Boolean variables and must therefore take an integral value. It follows that the right-hand side can be rounded up to the nearest integer giving the inferred clause

$$x_1 + x_2 + x_3 \geq 2 \tag{6}$$

Resolution also has a straightforward PB analog. In the Boolean case, we combine

$$\sum_i l_i + m \geq 1$$

with

$$\sum_j l'_j + \bar{m} \geq 1$$

to eliminate the complementary literals m and \bar{m} , giving

$$\sum_i l_i + \sum_j l'_j \geq 1$$

since $m + \bar{m} = 1$.

The PB version combines

$$\sum_i w_i l_i + xm \geq k$$

with

$$\sum_j w'_j l'_j + y\bar{m} \geq k'$$

to get

$$\sum_i yw_i l_i + \sum_j xw'_j l'_j \geq yk + xk' - xy$$

since, once again, $m + \bar{m} = 1$. Using such reasoning during the conflict analysis allows the learning of PB clauses as opposed to simple disjunctions.

Furthermore, unlike SAT, pseudo-Boolean is not limited to resolving away only complementary literals. For example, the inference of (6) from (6) involve no negative literals and has no counterpart in resolution. Critically, such inferences can also often be found automatically.

One such inference technique is known in the OR community as *strengthening* [13, 21], a form of learning that is based upon clauses becoming over-satisfied.³ As an example, consider (6) once again. Suppose that we set $x_1 = 0$, so that by propagation, we must have $x_2 = x_3 = 1$. Now

³One can think of the usual conflict-based analysis as learning from cases where a clause is *under*-satisfied

the constraint $x_2 + x_3 \geq 1$ is *over-satisfied*. Strengthening exploits this by adding extra terms that absorb the slack in the over-satisfied constraints. In this case, we can add the term x_1 to the over-satisfied third constraint directly yielding

$$x_1 + x_2 + x_3 \geq 2$$

Note that the strengthening can be applied even though the starting clauses are pure CNF. Furthermore, in this case, the strengthened clause subsumes the originals, and they can be removed from the constraint set.

The clearest example supporting the potential advantage of using cutting planes inference arises from considering a Boolean encoding of the pigeonhole problem (PHP): the principle that n pigeons cannot be placed into $n - 1$ holes without sharing. Using p_{ij} to indicate that pigeon i is in hole j , then the constraints that every pigeon is in some hole, but that no two pigeons can share a hole become

$$\begin{aligned} p_{i1} \vee \dots \vee p_{i,n-1} & \quad \forall i. \\ \neg p_{ik} \vee \neg p_{jk} & \quad \forall i \neq j, k. \end{aligned}$$

A DPLL algorithm that is restricted to SAT clauses cannot efficiently prove these collected axioms unsatisfiable, since every resolution proof is of size exponential in n [14]. Using PB methods, however, it is straightforward to produce polynomial size proofs [5]. For example, using strengthening the binary constraints can be automatically combined, and the system then becomes

$$\begin{aligned} \sum_k p_{ik} & \geq 1 & \quad \forall i. \\ \sum_i \bar{p}_{ik} & \geq (n - 1) & \quad \forall k. \end{aligned} \tag{7}$$

Summing these together and using $p_{ik} + \bar{p}_{ik} = 1$ gives $0 \geq 1$, the desired contradiction. Furthermore, such a proof (or close equivalent) can be automatically produced from conflict analysis within a DPLL algorithm [10]. Note that the mere use of PB is in itself not a guarantee of successfully solving the PHP. For example, a non-learning DPLL-based PB solver, such as that of Barth [2], will take the PB clauses (7) directly as input but will still take exponential time, and perform no better than if given the SAT formulation.

In summary, effective exploitation of the PB representation requires that the reasoning methods underlying constraint learning be true PB inference methods. In cases where problems such as the PHP are embedded within larger problems, if the solver only uses SAT methods, such embedded problems will lead to an exponential amount of unnecessary work.

Lastly, to understand the question of whether pseudo-Boolean encodings are likely to be of use in an MTV setting, we examined the set of MTV benchmarks made available by Velev.⁴ We examined the Velev problems first with a preprocessor that was designed to remove unit clauses and make other Boolean simplifications, and then with a second preprocessor that was designed to use the strengthening idea to find collections of axioms that could be more efficiently represented in a pseudo-Boolean setting. The second preprocessor typically reduced the size of the problem in

⁴<http://www.ece.cmu.edu/~mvelev>

question by some 10-20%. It seems reasonable to expect that this relatively modest reduction in problem size could lead to a substantial saving in inference, since if the axioms being eliminated correspond to embedded pigeonhole problems, a Boolean encoding will require exponential time for analysis by CHAFF or any similar algorithm.

3 Partial quantification

The second representational extension that we would like to discuss involves lifting a ground axiomatization to a partially first-order one. Since first-order theories are in general only semidecidable, no complete generalization is possible without substantially changing the solution techniques that can be applied.

Partial generalizations are possible, however. Somewhat more specifically, suppose that we have a universally quantified axiom of the form

$$\forall v_1, \dots, v_k . c(v_1, \dots, v_k) \tag{8}$$

where c is a nonquantified clause depending on the variables v_i . Working with this clause in existing SAT solvers involves constructing all of its ground instances and then manipulating each separately. We are able to do this because the domain of quantification is finite and has been specified explicitly for each variable v_i ; this guarantees that a “first-order” theory consisting of clauses like (8) is in fact equivalent to some larger ground theory, and is therefore decidable. This kind of partial lifting has been called “quantified propositional logic”, or QPROP [12] and is in sharp contrast to any attempt to lift satisfiability techniques to a full first-order setting using full-blown theorem provers. (Among other things, a failed proof in QPROP will still be accompanied by a counterexample, which is critical to MTV applications.)

As with the pseudo-Boolean extension, there are important computational ramifications to changing from a ground axiomatization to one that allows QPROP clauses such as (8). To understand why, it is important to realize that most ground SAT solvers spend most of their time searching for clauses with specific properties; they may, for example, be looking for unsatisfied clauses with a single unvalued literal, as in unit propagation and (4). To take a specific example, suppose that our theory contains all of the ground instances of

$$f(x, y) \wedge g(y, z) \rightarrow h(x, z)$$

or, in conjunctive normal form,

$$\neg f(x, y) \vee \neg g(y, z) \vee h(x, z) \tag{9}$$

Suppose also (as will often be the case) that both $f(x, y)$ and $g(y, z)$ are almost universally either false or unvalued. The only instances of (9) for which unit propagation is a possibility will be instances for which at least one of $f(x, y)$ and $g(y, z)$ is true. Even if every instance of f and g needs to be examined separately, the time needed to identify all of the relevant clauses has fallen from $o(n^3)$ to $o(n^2)$ if n is the size of the variable domain from which x, y and z are instantiated.

It is possible to understand this phenomenon generally. The problem of identifying all of the instances of an axiom that have a certain data property (e.g., unsatisfied with a single unvalued literal) is itself a search problem. The conventional approach of constructing all groundings of the original axiom and then examining these ground clauses singly solves this “subsearch” problem by generate-and-test, but much more efficient solutions are possible as well [12].

As with pseudo-Boolean encodings, there are more direct advantages to the QPROP analysis also. To understand the most obvious, suppose that we are resolving

$$a \wedge b \rightarrow c$$

with

$$c \rightarrow d$$

to obtain

$$a \wedge b \rightarrow d \tag{10}$$

(In more conventional terms, we resolve $\neg a \vee \neg b \vee c$ with $\neg c \vee d$ to get $\neg a \vee \neg b \vee d$.)

If the resolvents are in fact ground instances of the two axioms

$$a(x) \wedge b(y) \rightarrow c(x, y) \tag{11}$$

and

$$c(x, A) \rightarrow d(x) \tag{12}$$

we can resolve these two axioms (in the traditional first-order sense) to conclude

$$a(x) \wedge b(A) \rightarrow d(x)$$

which is substantially more general than the specific instance (10). The ability to learn more general clauses can be expected to improve the performance of the solver on the overall problem in question, although experimental validation of this claim has yet to be obtained. As before, the computational expense of working with these more general clauses can be expected to be extremely modest, since the new unification step (unifying $c(x, y)$ from (11) and $c(x, A)$ from (12) in the example) can be performed in time linear in the terms being unified. For QPROP specifically, the subsearch argument suggests that manipulating the lifted axioms can actually be *faster* than working with their sets of ground instances.

In order for QPROP encodings to be of interest to the MTV community, of course, we will need the problems being solved to be ground instances of quantified axiomatizations. In many encodings of AI planning problems, for example, an axiomatization is used that is extremely unnatural from an intuitive perspective but has the property that it has a manageably sized set of groundings [16].

In MTV, this appears not to be the case. Here is an axiomatization of an n -bit counter represented in terms of a vector $v(1), \dots, v(n)$ of Boolean variables [4]:

$$\forall t \in [1, T] . \quad v(1, t + 1) = \neg v(1, t) \tag{13}$$

$$\forall t \in [1, T], i \in [2, n] . \quad v(i, t + 1) = v(i, t) \oplus c(i - 1, t) \tag{14}$$

where T is the number of time points being analyzed, n is the number of bits in the counter, \oplus is exclusive or and the carry bit c is defined by

$$\forall i, t . c(i, t) = \bigwedge_{j \leq i} v(j, t) \quad (15)$$

We convert this to QPROP by introducing a fixed predicate

$$inc_t(t_1, t_2) \leftrightarrow (t_1 = t_2 + 1) \quad (16)$$

$$inc_i(i_1, i_2) \leftrightarrow (i_1 = i_2 + 1) \quad (17)$$

in order to remove the functional dependencies between the domain variables $t, t + 1$ and similarly for $i, i - 1$. The axiomatization of the full adder now becomes:

$$\forall t_1, t_2 . inc_t(t_1, t_2) \rightarrow v(1, t_1) = \neg v(1, t_2) \quad (18)$$

$$\forall t_1, t_2, i_1, i_2 . inc_t(t_1, t_2) \wedge inc_i(i_1, i_2) \rightarrow v(i_2, t_2) = v(i_2, t_1) \oplus c(i_1, t_1) \quad (19)$$

The $(n + 1)T$ axioms of the form (13) and (14) have been replaced by $2(n + T)$ axioms of the form (16) and (17) and the two axioms (18) and (19). To deal with (15), we note that it is equivalent to

$$\forall i, t . c(i, t) = v(i, t) \wedge c(i - 1, t)$$

and (15) is thus equivalent to

$$\forall t, i_1, i_2 . inc_i(i_1, i_2) \rightarrow c(i_2, t) = v(i_2, t) \wedge c(i_1, t)$$

In general, recovering the set of lifted axioms from which a set of ground axioms has been constructed appears to a subgraph isomorphism problem and potentially hard. We are thus unable to repeat the experiment of the previous section, analyzing Velev's benchmark suite directly. Instead, it seems important that the original lifted axiomatization be preserved if these techniques are to be applied in practice.

4 Symmetry Exploitation

There is one further technique that has recently come to the attention of the MTV community, specifically the possibility of exploiting symmetry to reduce search in some way. As an example, suppose the constraints are symmetric between two variables p and q . If search shows that the partial assignment $p \wedge \neg q$ is not part of any solution, then we immediately know that $q \wedge \neg p$ also cannot lead to a solution. This can be exploited in two fashions: symmetric reasoning [17] or symmetry breaking [7, 20]. In symmetric reasoning, when we deduce a new constraint we also deduce the symmetric variants of that constraint. In this example, the initial search corresponds to learning the clause $\neg p \vee q$, and symmetric reasoning would then allow us to conclude $\neg q \vee p$ as well.

In symmetry breaking, we add constraints that do not affect satisfiability, but remove some of the solution redundancy caused by the symmetry. In this example, we might add $q \rightarrow p$ without affecting satisfiability. This method has often been exploited for SAT problems generally and verification problems specifically [1].

Both symmetry reasoning and symmetry breaking are not specific to SAT, but can easily be extended pseudo-Boolean and QPROP. In fact, symmetry has already been exploited in lifted constraints [15]. Furthermore, symmetry detection is well known to be equivalent to finding an automorphism of a graph derived from the constraints themselves [8, 7]. This means that the smaller graphs appearing in the pseudo-Boolean and QPROP representations can be expected to speed the search for symmetries. In some cases, the pseudo-Boolean and QPROP representations can themselves make symmetries manifest as, for example, QPROP constraints can be regarded as a set of ground constraints related by a particular form of permutation symmetry between variables. In general, we expect the pseudo-Boolean and QPROP representations to be compatible with symmetry exploitation.

5 Conclusion

In this paper, we have discussed two representational extensions that seem likely to be incorporated into high-performance satisfiability engines in the near future. Each of these extensions incurs at worst minimal computational cost in its implementation while being capable of providing very substantial improvements in reasoning efficiency.

At some level, both of these extensions work by exploiting problem structure that has been obscured by a conventional Boolean satisfiability encoding. In the pseudo-Boolean case, the hidden structure involves counting in one form or another, saying that at least n variables from a given set are true (or false). Counting arguments are so fundamental to the operation of microprocessors that it seems reasonable to expect pseudo-Boolean encodings to be useful in this domain; we also provided specific evidence of applicability on Velev's benchmark suite of problems.

Quantified propositional (QPROP) encodings exploit the structure that is lost when a universally quantified axiom is replaced by its set of ground instances. Once again, virtually all naturally occurring problems have their roots in domains described by some set of physical laws – and these laws are typically represented by universally quantified axioms. So here, too, we would expect the representational improvements that we have discussed to be of direct interest to the microprocessor testing and verification community.

Acknowledgments

This work was sponsored in part by grants from Defense Advanced Research Projects Agency (DARPA), number F30602-98-2-0181, and DARPA and Air Force Research Laboratory, Rome, NY, under agreement numbered F30602-00-2-0534. The U.S. Government is authorized to reproduce and distribute reprints for Government purposes notwithstanding any copyright annotation thereon. The views and conclusions contained herein are those of the authors and should not be

interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of DARPA, Rome Laboratory, or the U.S. Government.

References

- [1] F. Aloul, A. Ramani, I. Markov, and K. Sakallah. Solving difficult sat instances in the presence of symmetry. In *Proc. of the Design Automation Conference (DAC)*, 2002.
- [2] P. Barth. A Davis-Putnam based enumeration algorithm for linear pseudo-boolean optimization. Technical Report MPI-I-95-2-003, Max Planck Institut für Informatik, Saarbrücken, Germany, 1995.
- [3] R. J. Bayardo and R. C. Schrag. Using CSP look-back techniques to solve real-world SAT instances. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence*, pages 203–208, 1997.
- [4] J. Burch, E. Clarke, D. Long, K. MacMillan, and D. Dill. Symbolic model checking for sequential circuit verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 13(4):401–424, 1994.
- [5] S. A. Cook. The complexity of theorem-proving procedures. In *Proceedings of the 3rd Annual ACM Symposium on the Theory of Computing*, pages 151–158, 1971.
- [6] F. Copt, L. Fix, E. Giunchiglia, G. Kamhi, A. Tacchella, and M. Vardi. Benefits of bounded model checking in an industrial setting. In *13th Conference on Computer Aided Verification, CAV'01*, Paris, France, July 2001.
- [7] J. Crawford, M. Ginsberg, E. Luks, and A. Roy. Exploiting symmetry by adding symmetry breaking predicates. In *Principles of Knowledge Representation and Reasoning: Proceedings of the Fifth International Conference (KR-96)*, Boston, MA, 1996.
- [8] J. M. Crawford. A theoretical analysis of reasoning by symmetry in first-order logic (extended abstract). In *AAAI Workshop on Tractable Reasoning*, 1992.
- [9] R. Dechter. Enhancement schemes for constraint processing: Backjumping, learning, and cutset decomposition. *Artificial Intelligence*, 41:273–312, 1990.
- [10] H. E. Dixon and M. L. Ginsberg. Inference methods for a pseudo-boolean satisfiability solver. In *Proceedings of the Eighteenth National Conference on Artificial Intelligence (AAAI-02)*, 2002. To appear.
- [11] M. L. Ginsberg. Dynamic backtracking. *Journal of Artificial Intelligence Research*, 1:25–46, 1993.

- [12] M. L. Ginsberg and A. J. Parkes. Search, subsearch and QPROP. In *Proceedings of the Seventh International Conference on Principles of Knowledge Representation and Reasoning*, Breckenridge, Colorado, 2000.
- [13] M. Guignard and K. Spielberg. Logical reduction methods in zero-one programming. *Operations Research*, 29, 1981.
- [14] A. Haken. The intractability of resolution. *Theoretical Computer Science*, 39:297–308, 1985.
- [15] D. Joslin and A. Roy. Exploiting symmetry in Lifted CSPs. In *Proceedings of the Fourteenth National Conference on Artificial Intelligence (AAAI-97)*, pages 197–202, 1997.
- [16] H. Kautz, D. McAllester, and B. Selman. Encoding plans in propositional logic. In *Proceedings of the Fifth International Conference on Principles of Knowledge Representation and Reasoning*, Boston, MA, 1996.
- [17] B. Krishnamurthy. Short proofs for tricky formulas. *Acta Informatica*, 1985.
- [18] M. Moskewicz, C. Madigan, Y. Zhao, L. Zhang, and S. Malik. Chaff: Engineering an efficient SAT solver. In *39th Design Automation Conference*, 2001.
- [19] G. L. Nemhauser and L. A. Wolsey. *Integer and Combinatorial Optimization*. John Wiley & Sons Ltd, 1988.
- [20] J.-F. Puget. On the satisfiability of symmetrical constrained satisfaction problems. In *In J. Komorowski and Z.W. Ras, editors, Proceedings of ISMIS'93, pages 350–361. Springer-Verlag, 1993. Lecture Notes in Artificial Intelligence 689.*, 1993.
- [21] M. W. P. Savelsbergh. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing*, 6:445–454, 1994.
- [22] R. M. Stallman and G. J. Sussman. Forward reasoning and dependency-directed backtracking in a system for computer-aided circuit analysis. *Artificial Intelligence*, 9:135–196, 1977.
- [23] H. Zhang and M. Stickel. Implementing the davis-putnam method. In I. Gent, H. van Maaren, and T. Walsh, editors, *SAT 2000*, pages 309–326. IOS Press, 2000.