Inference Methods for a Pseudo-Boolean Satisfiability Solver

Heidi E. Dixon and Matthew L. Ginsberg
CIRL
1269 University of Oregon
Eugene, OR 97403-1269
{dixon, ginsberg}@cirl.uoregon.edu

Abstract

We describe two methods of doing inference during search for a pseudo-Boolean version of the RELSAT method. One inference method is the pseudo-Boolean equivalent of learning. A new constraint is learned in response to a contradiction with the purpose of eliminating the set of assignments that caused the contradiction. We show that the obvious way of extending learning to pseudo-Boolean is inadequate and describe a better solution. We also describe a second inference method used by the Operations Research community. The method cannot be applied to the standard resolution-based AI algorithms, but is useful for pseudo-Boolean versions of the same AI algorithms. We give experimental results showing that the pseudo-Boolean version of RELSAT outperforms its clausal counterpart on problems from the planning domain.

Introduction

Building boolean satisfiability solvers that implement strong proof systems is an important goal for the field of AI. This goal is motivated by results from the field of proof complexity showing that some proof systems are more limited than others. An inference system is limited if it is impossible to construct short proofs of unsatisfiability for certain families of problems. These results have significant consequences for systematic satisfiability solvers. Because systematic solvers can be viewed primarily as constructing proofs of unsatisfiability, it follows that these solvers are subject to the limitations of the proof systems they implement.

Throughout the AI community, satisfiability problems are typically represented as a set of constraints in conjunctive normal form (CNF) with resolution as the primary inference step. We believe that resolution-based methods are prevalent because of their simplicity. Unfortunately, many unsatisfiable problems have no short resolution proofs of unsatisfiability. An example is the pigeonhole problem which states that $n+1$ pigeons cannot be placed in n holes. The shortest resolution proof of unsatisfiability for the pigeonhole problem is exponential in the number of pigeons (Haken 1985). Because resolution is a weak proof system, resolution-based methods have poor performance on a variety of easy problems like the pigeonhole problem. They may also be unnecessarily slow on structured problems from areas like planning and scheduling where embedded pigeonhole problems are common. Refining current algorithms may yield improvements, but can never provide polynomial time scaling on these problems unless the underlying representation and inference used by the solver is changed.

One approach to addressing the problem of representation is to adapt a successful resolution-based method to use a stronger representation. The work done on resolution-based methods has yielded successful strategies and produced significant improvements in performance. The hope is that the progress made will carry over into new representations. The growing number of solvers taking this approach show that such implementations are possible. This is the approach used in FDPLL (Baumgartner 2000); a version of the Davis-Putnam-Logeman-Loveland (DPLL) procedure lifted to solve first-order logic problems. This is also the approach used in OPBDP (Barth 1995), PRS (Dixon & Ginsberg 2000) and SATIRE (Whittemore, Kim, & Sakallah 2001). These solvers borrow from the Operations Research using pseudo-Boolean (PB) representation.

This paper discusses some of the challenges and benefits of lifting the RELSAT style of learning to use pseudo-Boolean representation. The algorithm RELSAT is an extension of the DPLL method. Both algorithms work by taking a valid partial assignment and attempting to extend it to a valid full assignment by incrementally assigning values to variables. An important difference between the algorithms is that RELSAT infers new constraints during search by using resolution. It uses the technique of relevance-bounded learning to keep the size of the constraint set manageable (Bayardo & Schrag 1997; Ginsberg 1993). Learned constraints are removed from the constraint set when they are less likely to be needed. These added abilities give RELSAT a dramatic advantage over DPLL on structured problems.

Our primary focus will be on the way inference is used during search and how this role changes when we move from CNF to pseudo-Boolean representation. RELSAT will infer a new clause every time a contradiction is encountered. The purpose of the new clause is to eliminate the set of assignments that caused the contradiction. The resolution inference rule happens to be perfect for this purpose.

A comparable pseudo-Boolean version of this method should achieve the same goals. In the pseudo-Boolean case we have more options to consider when generating a new

constraint in response to a contradiction. The correct way to infer new constraints is less clear. Pseudo-Boolean inference has added benefits that result directly from the more expressive nature of pseudo-Boolean constraints. In some cases, a new constraint may eliminate assignments corresponding to parts of the search space not yet explored, in addition to eliminating the set of assignments that caused a contradiction. This, in a sense, eliminates mistakes before they are made. In other cases, care must be taken to ensure that the generated constraint actually eliminates the set of assignments that led to the contradiction. We will give explicit examples showing each of these cases, and we will describe a learning method for the pseudo-Boolean case that meets the requirement of eliminating a specific set of assignments in response to a contradiction. We present experimental results comparing the performance of the clausal version of RELSAT to the pseudo-Boolean version PRS.

Another benefit of using pseudo-Boolean representation is that new kinds of inferences are possible that are not possible in a resolution system. A more expressive constraint can be inferred from a set of simple disjunctive constraints. Automating these inferences present a new challenge because they have no precedents in resolution-based methods. We describe a technique from the field of Operations Research that can be used to automate this kind of inference and show how it can be used in an AI style solver.

Preliminaries

Representation and Inference

Conjunctive normal form and resolution Within the AI community, satisfiability problems are typically represented as a set of constraints in conjunctive normal form (CNF). A constraint or clause is a disjunction of literals and a problem instance is the conjunction over a list of clauses. The primary inference step is resolution.

$$\frac{\begin{array}{c} a_1 \vee \cdots \vee a_k \vee l \\ b_1 \vee \cdots \vee b_m \vee \bar{l} \end{array}}{a_1 \vee \cdots \vee a_k \vee b_1 \vee \cdots \vee b_m}$$

Two clauses resolve if there is exactly one literal l that appears positively in one clause and negatively in the other. A new clause is derived by disjoining the two clauses and removing both l and $\neg l$. If a literal appears twice in the resulting clause, the clause can be rewritten with the literal appearing only once. This is known as factoring.

Linear inequalities and cutting planes Pseudo-Boolean representation comes from the Operations Research community and is a subset of their general representation in which constraints are expressed as linear inequalities

$$\sum a_j x_j \geq k$$

Here, the x_i are non-negative integer variables and the a_i and k are integers. The corresponding inference system is called the cutting plane system (CP). There are two rules of inference: (i) derive a new inequality by taking a linear combination of a set of inequalities, (ii) given an inequality $\sum a_j x_j \geq k$ derive $\sum \lceil \frac{a_j}{d} \rceil x_j \geq \lceil \frac{k}{d} \rceil$, where d is a positive integer. The notation $\lceil q \rceil$ denotes the least integer greater than or equal to q. If the inequality $0 \geq 1$ is derived then the original set of inequalities is inconsistent. If the integer variables are restricted to the domain $\{0, 1\}$, then the inequality is called *pseudo-Boolean*. The expression $\overline{x}$ refers to the negation of the variable x, so that for all literals x, $\overline{x} = 1 - x$.

The CP inference system is properly stronger than resolution. The existence of a polynomial-length resolution proof implies the existence of a polynomial-length cutting plane proof, but the reverse does not hold (Cook, Coullard, & Turán 1987). The pigeonhole problem which has only exponential-length proofs in resolution has polynomial-length proofs in CP (Cook, Coullard, & Turán 1987).

Translating between representations A disjunction of literals $x_0 \vee x_1 \vee \cdots \vee x_n$, can be equivalently written as a linear pseudo-Boolean inequality.

$$x_0 + x_1 + \cdots + x_n \geq 1$$

Pseudo-Boolean inequalities with a right hand side that is equal to 1 are called clausal inequalities.

We can translate a pseudo-Boolean constraint into a set of clauses in CNF as follows. Given a constraint

$$\sum_{i=1}^{n} a_i x_i \geq k \tag{1}$$

Let $L = \{x_1, x_2, \ldots, x_n\}$ be the set of literals in (1). For any set $S = \{y_1, y_2, \ldots, y_j\}$ such that $S \subseteq L$, if

$$\sum_{x_i \notin S} a_i \leq k - 1$$

then the disjunction $y_1 \vee y_2 \vee \cdots \vee y_j$ is implied by the constraint (1). In other words, if the sum over the coefficients of the remaining literals cannot satisfy the constraint, then at least one of the literals in S must be true. For example, given the constraint

$$2a + b + c \geq 2$$

we can derive the clause $a \vee b$, since c alone is not enough to satisfy the constraint. The constraint (1) is logically equivalent to the conjunction over the set of all disjunctions generated this way (Benhamou, Sais, & Siegel 1994).

Understanding RELSAT

The algorithm RELSAT (Bayardo & Schrag 1997) is a version of the classic Davis-Putnam-Logeman-Loveland method (Davis & Putnam 1960; Loveland 1978) with the addition of relevance-bounded learning. If the relevance-bounded learning feature is disabled during execution, then RELSAT becomes a DPLL implementation.

Davis-Putnam-Logeman-Loveland DPLL takes a valid partial assignment and attempts to extend it to a valid full assignment by incrementally assigning values to variables. This creates a binary tree where each node corresponds to a set of assignments. DPLL explores the tree using depth first

search with backtracking. A backtrack occurs when a contradiction is encountered. The algorithm terminates when a solution is found or when the entire space has been explored.

Procedure 0.1 Davis-Putnam-Logeman-Loveland *Given a SAT problem S and a partial assignment of values to variables P, to compute* `solve`(C, P)*:*

if `unit-propagate`(P) fails, **then** return failure
else set $P :=$ `unit-propagate`(P)
if all clauses are satisfied by P, **then** return P
$v :=$ an atom not assigned a value by P
if `solve`$(C, P \cup (v :=$ `true`$))$ succeeds,
  then return $P \cup (v :=$ `true`$))$
else return `solve`$(C, P \cup (v :=$ `false`$))$

Variables are assigned values in two ways. In the first way, unit propagation, clauses are identified that have no satisfied literals and exactly one unvalued literal. In each such clause, the unvalued literal is valued favorably. This process is repeated until a contradiction is encountered, a solution is found, or no more clauses meet the necessary conditions. If the unit propagation function terminates without reaching a contradiction or finding a solution, then a variable is selected and assigned a value by a branching heuristic.

Procedure 0.2 Unit propagation
to compute `unit-propagate`(P)*:*

while there is a currently unsatisfied clause $c \in C$
        that contains at most one literal
        unassigned a value by P **do**
    if every atom in c is assigned a value by P,
    then return failure
    else $a :=$ the atom in c unassigned by P
        augment P by valuing a so that c is satisfied
    end if
end while
return P

Learning and relevance-bounded learning One way that solvers use inference is through learning (Stallman & Sussman 1977). A drawback to simple backtracking algorithms like DPLL is that they may end up solving the same subproblems repeatedly. Learning new valid constraints can prevent this from happening. When a contradiction is encountered, the set of assignments that caused the contradiction are identified. We will call this set the *conflict set*. A new constraint is constructed that excludes the assignments in the conflict set. The constraint is added to the constraint database to ensure that the faulty set of assignments will be avoided in the future. An example might look like this. Given the partial assignment $\{a =$ `true`$, b =$ `false`$, d =$ `true`$, e =$ `false`$\}$ (which we write somewhat more compactly as $\{a, \overline{b}, d, \overline{e}\}$), together with the two clauses

$$\overline{a} \vee b \vee c \vee e$$

$$\overline{c} \vee \overline{d}$$

We encounter a contradiction for variable c. The first clause requires c, while the second requires $\overline{c}$. The conflict set $\{a, \overline{b}, d, \overline{e}\}$ is the union of the unfavorable assignments for each clause. Before we backtrack, we construct a new clause that is the resolvent of the preceding two clauses.

$$\overline{a} \vee b \vee e \vee \overline{d} \tag{2}$$

This clause has the property of being unsatisfied by the current partial assignment. It disallows the assignments that caused the contradiction, and we can use (2) to determine how far we must backtrack before we can safely move forward again. The derived clause "fixes" the mistake that was made in that we are protected from repeating the mistake as long as we keep the new clause in our constraint database.

Learning new constraints reduces the size of the search space by eliminating parts of the space that cannot contain solutions. Unfortunately, reducing the size of the search space does not always correspond to a reduction is execution time. The number of constraints learned can be exponential in the size of the problem. This can exhaust memory resources. The algorithm spends more time managing its large database of constraints and performance degrades. The learning process must be restricted in some way to prevent an unmanageable number of constraints from accumulating.

Relevance-bounded learning is a restricted version of learning. Our focus is not relevance-bounded learning, so we give only a high level description of it. Relevance-bounded learning defines an integer measure of how relevant a learned constraint is in relation to the current position in the search space. The relevance of a constraint estimates the likelihood that the constraint can be used to prune the search space. Constraints with low values are more relevant then those with higher values. As the position in the search space changes the relevance of the clause will change in response. A *relevance bound* is established, and constraints are discarded when their relevance exceeds the bound.

Learning with pseudo-Boolean constraints

The pseudo-Boolean case is similar to the clausal case in that a contradiction occurs when a partial assignment together with two constraints causes a variable to be labelled both 1 and 0. A good solution should generate a constraint that disallows the set of assignments that caused the contradiction. We will describe two ways of generating a new constraint in response to a contradiction. The first way generates a complex constraint that may eliminate extra sets of assignments in addition to those in the conflict set. Unfortunately, in certain cases it may not eliminate the exact set of assignments in the conflict set. The second way eliminates exactly the set of assignments in the conflict set. A good learning strategy can be built by combining the two methods.

One way of generating a new constraint is to do a pseudo-Boolean version of resolution, taking a linear combination of the two constraints in a way that causes the contradiction variable to be canceled out of the resulting constraint. Consider the following example. Suppose we have a partial assignment $\{c = 1, e = 1, b = 0, d = 0\}$, and constraints

$$a + d + \overline{e} \geq 1 \tag{3}$$

$$\overline{a} + b + c \geq 2 \tag{4}$$

These cause the variable a to be simultaneously 1 and 0. We generate a new constraint by adding (3) and (4) to get $d + \overline{e} + b + c \geq 2$.

By inspection, we can see that the conflict set is $\{b = 0, d = 0, e = 1\}$, and that the derived constraint excludes this assignment. This constraint also eliminates some additional bad assignments. For example, it also eliminates the assignment $\{c = 0, d = 0, e = 1\}$. In addition to fixing the current assignment error, we've learned something new about a different part of the search space. In the clausal version, learned constraints prevent us from repeating a mistake. Here we have the potential to prevent mistakes before they happen.

Unfortunately it is possible to construct cases where the constraint derived does not exclude the set of assignments causing the contradiction. Given the partial assignment $\{c = 1, e = 1, b = 0, d = 0\}$ and constraints

$$2a + d + e \geq 2$$

$$2\overline{a} + b + c \geq 2$$

Adding gives $d+e+b+c \geq 2$, which still allows the set of assignments in the conflict set $\{b = 0, d = 0\}$. We may make the same bad assignment again later in the search. Also this constraint does not give any direction to the backtrack since it is satisfied under the current partial assignment.

We define a second method of generating a constraint in response to a contradiction that is guaranteed to eliminate the necessary set of assignments. We begin by constructing a weakening of each parent constraint into a clausal inequality using the method described earlier for generating valid CNF clauses from a pseudo-Boolean constraint. Each constraint will satisfy two properties: it contains the contradiction variable in the same form as it appears in the parent constraint, and all other literals are unsatisfied under the current partial assignment. When choosing literals to add to the constraint, priority is given to literals whose assignments occur earliest in the partial assignment. The two constraints generated can be resolved together to create a new valid constraint that is unsatisfied under the current partial assignment. The size of the resulting backjump will be maximized because the learned constraint contains the failed literals valued earliest in the partial assignment. In the previous example the two weakened constraints would be $a + d \geq 1$ and $\overline{a} + b \geq 1$, which we could then resolve together to get $d + b \geq 1$. This constraint correctly eliminates the assignments in the conflict set and gives direction to the backtrack.

It can be determined before hand by inspecting the parent constraints if a constraint generated with the linear combination method will eliminate the conflict set. We consider the coefficients of the contradiction variable in each constraint. If either of these coefficients is equal to 1 then the constraint generated by the first method will subsume the constraint generated by the second method and therefore eliminate the conflict set. If neither coefficient is equal to 1, then it is undetermined whether the conflict set will be eliminated. In our implementation we use the first method when it is guaranteed to eliminate the conflict set. In all other cases we generate a constraint with both methods and choose the constraint that causes the larger backtrack.

Constraint strengthening

An advantage of using pseudo-Boolean representation is that some interesting new inference techniques become possible. The following method is from the Operations Research field and is used to preprocess mixed integer programming problems (Savelsbergh 1994; Guignard & Spielberg 1981).

Suppose we make the assumption $\{x_0 = 1\}$ and, applying some form of propagation to our constraint set, we discover that under this assumption a constraint $\sum a_i x_i \geq r$ becomes oversatisfied by an amount s in that the sum of the left hand side is greater (by s) than the amount required by the right hand side of the inequality. The oversatisfied constraint can be replaced by the following:

$$s\overline{x}_0 + \sum a_i x_i \geq r + s \tag{5}$$

If $x_0 = 1$, we know that $\sum a_i x_i \geq r + s$, so (5) holds. If $x_0 = 0$, then $s\overline{x}_0 = s$ and we still must satisfy the original constraint $\sum a_i x_i \geq r$, so (5) still holds. The new constraint implies the original one, so no information is lost in the replacement. The OR community uses this technique during preprocessing. A literal is fixed, propagation is applied, and any oversatisfied constraint is strengthened. Consider the following set of clauses:

$$a + b \geq 1$$

$$a + c \geq 1$$

$$b + c \geq 1$$

If we set $\{a = 0\}$, we must then value $\{b = 1, c = 1\}$ or the first two constraints will become unsatisfied. The third constraint is oversatisfied and can thus be replaced by

$$a + b + c \geq 2.$$

The power of this method is that it allows us to build more complex statements from a set of simple statements. The strengthened constraint will often subsume some or all of the constraints involved in generating it. In this case the new constraint subsumes all three of the generating constraints.

This rule can be generalized as follows. Given any set of assumptions $A = \{x_0, x_1, \ldots, x_k\}$, if we apply some form of propagation and discover that under these assumptions the constraint $\sum a_i x_i \geq r$ becomes oversatisfied by an amount s, we can add to our constraint set the constraint

$$s\sum_{i=1}^{k} \overline{x}_i + \sum a_i x_i \geq r + s \tag{6}$$

In the case where all the assumptions hold, we know that $\sum a_i x_i \geq r + s$, so (6) holds. If any assumption x_j fails, then $s\overline{x}_j \geq s$ and $\sum a_i x_i \geq r$, so (6) still holds.

In addition to use during preprocessing, this method can be applied during search as well. When a constraint becomes oversatisfied under the current partial assignment, the set of assignments that caused the constraint to be oversatisfied can be determined in time $O(n^2)$. The constraint is strengthened or a new constraint is learned. We have implemented this method both as a preprocessor and as an inference method during search. We have not yet run extensive

	RELSAT		Pre.	PRS	
Instance	sec	nodes	sec	sec	nodes
hole8.cnf	2	26670	0	0	11
hole9.cnf	29	270726	0	0	12
hole10.cnf	393	3049835	0	0	17
hole11.cnf	7488	37573080	0	0	15
hole12.cnf			0	0	20
hole20.cnf			0	0	34
hole30.cnf			4	0	52
hole40.cnf			25	0	75
hole50.cnf			95	0	95

Table 1: Run time (seconds) and no. of node expansions

experiments, but we suspect that for most problems attempting a strengthening for every occurrence of an oversatisfied constraint will be far too expensive. It is unclear whether an efficient implementation will provide benefits beyond those gained by preprocessing alone. However, excessive preprocessing can be expensive, so it may be valuable to let the search direct the strengthening process. This would also allow the possibility of strengthening constraints learned in response to contradictions.

Experimental Results

We have implemented the described learning methods in the algorithm PRS (Pseudo-boolean RelSat) and we compare its performance to its clausal counterpart RELSAT. Using pseudo-Boolean constraints has computational cost, although in theory this cost is linear. In practice we find an increase in the run time of unit propagation of a factor of 2 to 5. A discussion of these costs and other implementation details can be found elsewhere (Dixon & Ginsberg 2000).

The pigeonhole problem is an important benchmark for evaluating pseudo-Boolean solvers because short cutting plane proofs of unsatisfiability exist for the problem. Excellent performance on these problems should be a base requirement for a systematic pseudo-Boolean solver. We present some results comparing performance of RELSAT and PRS on the pigeonhole problem. Both algorithms used the same CNF descriptions of the problems. PRS inputs files in CNF and represents each clause as a clausal inequality. The pseudo-Boolean version used the preprocessing method described above before solving the problems. All experiments were run on a 900 Mhz AMD Athlon processor. The times shown are for optimal choices of relevance bounds. For RELSAT this is a bound of 0, and for PRS this is a bound of 1. The times are an average over 10 trials. The first two columns give time in seconds and number of nodes explored for RELSAT. The next three columns show preprocessing time in seconds, solution time in seconds, and the number of nodes explored for PRS.

PRS with preprocessing dramatically outperformed RELSAT on the pigeonhole problem. This result is not surprising. There are two things to note here: first, that PRS fulfills the basic requirement of efficiently solving pigeonhole problems, and second that the constraint strengthening inference is required to build the pseudo-Boolean constraints that are needed to improve performance. Without the preprocessing phase the performance of PRS on the same problems is similar to the performance for RELSAT.

A more interesting experiment considers logistics planning problems (Kautz & Selman 1996). The original problems are too easy for current solvers and are only available in CNF. The problem domain involves using a set of planes to move a collection of packages from their initial locations to final locations during a given time period. These problems contain a number of constraints that are easy to encode in pseudo-Boolean. For instance, the constraint that says a plane can be in only one location at a time can be written as

$$\overline{p}_{i1k} + \overline{p}_{i2k} + \cdots + \overline{p}_{ink} \geq n - 1 \qquad (7)$$

The variable p_{ijk} represents plane i being in location j at time k, and n is the number of locations. The equivalent expression in CNF requires $\binom{n}{2}$ binary clauses. The problems were randomly generated with a variety of parameter values. We discarded problems that were trivial for both solvers or were satisfiable. Because these problems were generated by hand, it is difficult to know how hard these problems are relative to an easy/hard phase transition.

A design goal was to focus the comparison on the effect of learning methods. Ideally we'd like to eliminate differences in performance due to branching heuristics. Branching heuristics play an important role in reducing the size of the search space for the clausal versions of DPLL and RELSAT. We believe that they will also be important for pseudo-Boolean versions as well. We chose to use the $PROP_{31}$ heuristic (Li & Anbulagan 1997) for both algorithms. More recent heuristics have shown better results for some problem domains, but $PROP_{31}$ is a good heuristic and the implementations for each algorithm were similar. It is unclear whether this heuristic is equally good when used on pseudo-Boolean constraints. We still felt that this choice would bias our results less than abandoning branching heuristics altogether, because branching heuristics are so important for successful solvers.

For each instance we generated a CNF version and two pseudo-Boolean versions. The first pseudo-Boolean version was generated by using cardinality constraints like (7) to express sets of constraints more concisely. The rest of the constraints were written as clausal inequalities. The second pseudo-Boolean version was generated by running the strengthening preprocessor directly on the clausal version.

We report the average execution time over ten trials for each instance. Removing the highest and lowest times (to test for outliers) did not significantly affect the results.

Although PRS needs to manage a more complex representation, the benefit on these instances of using pseudo-Boolean representation outweighs the cost. Both pseudo-Boolean formulations provided better overall performance. In most cases the cost of preprocessing was made up for by reduced solution time.

Conclusion and future work

As we begin to experiment with more complex representations in our solvers, we deepen our understanding of the relationship between search and inference. The familiar learning technique used by DPLL style algorithms can be

Instance	V	clausal C	clausal RELSAT	PB formulation 1 C	PB formulation 1 PRS	PB formulation 2 C	PB formulation 2 Pre.	PB formulation 2 PRS
log8.15.6.5.2	1418	21452	6	13612	2	12772	6	1
log8.15.6.5.4	1418	21452	9	13612	1	12772	6	1
log8.15.6.5.6	1418	21452	11	13612	3	12772	6	2
log8.15.6.5.3	1418	21452	115	13612	15	12772	6	30
log10.13.7.5.5	1625	24687	52	17142	16	16142	7	17
log14.8.4.6.42	1424	15096	30	11760	19	10752	2	26
log14.8.4.6.4	1424	15096	86	11760	36	10752	2	45
log8.16.7.5.1	1616	26351	46	16626	10	15706	10	7
log9.15.7.5.2	1668	26830	139	17610	15	16602	9	16

Table 2: Number of variables (V), clauses (C), and run time (seconds)

adapted to use pseudo-Boolean representation. However, the role played by learning in response to a contradiction has changed. A learned pseudo-Boolean constraint may eliminate assignments in parts of the search space not yet explored in addition to eliminating the set of assignments that caused the contradiction. Further work on learning methods is needed if we are to understand this new role. Branching heuristics and relevance policies will also need to be revisited with respect to pseudo-Boolean representation. Lazy data structures such as the watched literal implementation (M.Moskewicz *et al.* 2001) also need to be investigated. It is encouraging that our preliminary implementation of pseudo-Boolean RELSAT outperformed its clausal counterpart in the planning domain despite the large number of unanswered questions. The constraint strengthening technique is an important missing link to the puzzle. It provides a way to construct the more expressive constraints that are needed to improve performance. Further work is needed to understand how this form of inference can be incorporated into the search process.

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

Barth, P. 1995. 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.

Baumgartner, P. 2000. FDPLL - a first-order Davis-Putnam-Logeman-Loveland procedure. In P. Baumgartner, C. Fermuller, N. P., and Zhang, H., eds., *CADE 2000*.

Bayardo, R. J., and Schrag, R. C. 1997. Using CSP look-back techniques to solve real-world SAT instances. In *Proc. AAAI-97*.

Benhamou, B.; Sais, L.; and Siegel, P. 1994. Two proof procedures for a cardinality based language in propositional calculus. In *Proceedings of STACS94, volume 775 de Lecture Notes in Computer Science*.

Cook, W.; Coullard, C.; and Turán, G. 1987. On the complexity of cutting plane proofs. *Journal of Discrete Applied Math* 18:25–38.

Davis, M., and Putnam, H. 1960. A computing procedure for quantification theory. *Journal of the Association for Computing Machinery* 7:201–215.

Dixon, H. E., and Ginsberg, M. L. 2000. Combining satisfiability techniques from AI and OR. *The Knowledge Engineering Review* 15(1).

Ginsberg, M. L. 1993. Dynamic backtracking. *Journal of Artificial Intelligence Research* 1:25–46.

Guignard, M., and Spielberg, K. 1981. Logical reduction methods in zero-one programming. *Operations Research* 29.

Haken, A. 1985. The intractability of resolution. *Theoretical Computer Science* 39:297–308.

Kautz, H., and Selman, B. 1996. Pushing the envelope: Planning, propositional logic, and stochastic search. In *Proc. AAAI-96*.

Li, C. M., and Anbulagan. 1997. Heuristics based on unit propagation for satisfiability problems. In *Proc. IJCAI-97*.

Loveland, D. W. 1978. *Automated Theorem Proving: A Logical Basis*. North Holland.

M.Moskewicz; C.Madigan; Zhao, Y.; Zhang, L.; and Malik, S. 2001. Chaff: Engineering an efficient SAT solver. In *Proc. of the Design Automation Conference*.

Savelsbergh, M. W. P. 1994. Preprocessing and probing for mixed integer programming problems. *ORSA Journal on Computing* 6:445–454.

Stallman, R. M., and Sussman, G. J. 1977. Forward reasoning and dependency directed backtracking in a system for computer aided circuit analysis. *Artificial Intelligence* 9(2):135–196.

Whittemore, J.; Kim, J.; and Sakallah, K. 2001. SATIRE: A new incremental satisfiability engine. In *Proc. of the Design Automation Conference*.