A Linear Programming based Satisfiability solver using a new Horn-driven search tree design

Hans van Maaren
Linda van Norden

Technical University Delft
Faculty of Information Technology and Systems
Department of Information Systems and Algorithms
Mekelweg 4 2628 CD Delft
P.O. Box 5031 2600 GA Delft
The Netherlands
Email: H.vanmaaren@its.tudelft.nl, L.vanNorden@its.tudelft.nl

May 2, 2002

1 Introduction

We will present an algorithm for the satisfiability problem, which finds its origin in the Integer Programming area, and therefore will also generalize to more general constraint programming problems. This algorithm is based on single-lookahead unit resolution (Franco (1997)), linear programming and a new search tree design (Van Maaren and Dang (2002)). The special aspect of the tree is that it is not a binary search tree. The advantage of our algorithm over a standard integer programming approach, is that we need to solve a linear program only in a very limited number of the nodes in the search tree. In every node in the search tree we first apply single-lookahead unit resolution. The unit propagation algorithm we used in our implementation is based on the watched literal strategy, described in [Van Gelder], based on [chaff]. Only in case, the unit resolution does not lead to the conclusion that the formula in the node is unsatisfiable or has a satisfying assignment, we solve a linear program. The solution of the linear program is used to split the part of the search into subparts. This splitting aims for getting a formula close to a Horn formula.

Horn clauses are clauses in which at most one variables occurs in negated form. Therefore, formulae with only Horn clauses of size 2 and larger are always satisfied if all variables are set to true. After unit resolution, all clauses have length 2 or more. The linear programming solution finds a feasible solution that is as close as possible to the all-ones vector. A more-Horn formula will therefore yield linear programming solutions that are closer to an integer-valued solution. By construction, conflicts are likely to be close to the linear programming solution.

2 Linear programming and unit propagation

Let n be the number of variables and m be the number of clauses of a satisfiability instance. An instance of the satisfiability problem can be described by a binary integer linear program. Let A be a $m \times n$ 0-1 matrix, with $a_{ij} = 1$ if variable j is contained in clause i, $a_{ij} = -1$ if the negation of variable j is contained in clause i and 0 otherwise. Let x be a 0-1 vector with $x_i = 1$ if variable i is set to TRUE and 0 otherwise. For the vector b of right hand sides, b_j equals 1 minus the number of negated variables in clause j. With this notation a satisfiable assignment corresponds to a lattice point of the polytope:

$$\{Ax \geq b, x \in \{0,1\}^n\}$$

If unit resolution yields a contradiction, none of the lattice points in the part of the search tree under consideration does satisfy the formula. This implies that the linear program for this part of the search tree is infeasible. Single lookahead unit resolution can replace solving a linear program in many cases, as follows from the following lemma.

Lemma 2.1. *If a linear programming formulation of a satisfiability instance including constraints describing the part of the search space concerned, augmented* *by the constraint $x_1 = 1$ is infeasible, then the corresponding CNF-formula augmented with X_1 gives a contradiction when unit propagation is applied.*

From this lemma follows that applying single-lookahead unit resolution suffices to prove that a part of the search tree does not contain a satisfying assignment. Only in case the single-lookahead unit resolution does not give a satisfying assignment or a contradiction we solve a linear program. By the above mentioned lemma, this linear program always has a feasible solution. This linear programming solution is used to split the search space into so-called regions.

3 Search tree design

As mentioned above, we use the linear programming solution ω as a splitting point to split the search space into $n+1$ parts. It is not obvious which objective function is most suitable for computing the splitting points. Every integer valued solution of the linear program corresponds to a satisfying assignment. Clearly, this does not mean that every objective function yields equally good splitting points. For the moment, we chose the objective function of each of the variables to be 1. This can be refined to an objective function that changes dynamically in the search tree. The aim of the splitting is to obtain a formula that is clause to a Horn formula.

The region G_i is the cone spanned by the vectors from $\{-e_0, -e_1, \ldots, -e_n\} \backslash \{-e_i\}$, in which e_i is the unit vector with a 1 in the i-th row. The regions that follow from this splitting are mentioned below. Besides, we mention the clauses the CNF-formula has to satisfy in each of the regions, following from their definitions.

The first, and often smallest region, G_0 is defined as follows

$$G_0 = \{x_i \leq \omega_i, \text{for all } i\}$$

Only lattice points can be feasible solutions to the satisfiability problem. Hence, we may replace $x_i \leq \omega_i$ by $x_i \leq \lfloor \omega_i \rfloor$. The clauses that correspond to these linear inequalities are $\neg X_i$ for all i for which $\omega_i < 1$.

Elements x in the second region G_1 satisfy

$$x_1 \geq \omega_1 \tag{1}$$
$$x_1 - x_2 \geq \lceil \omega_1 - \omega_2 \rceil \tag{2}$$
$$\ldots \tag{3}$$
$$x_1 - x_n \geq \lceil \omega_1 - \omega_n \rceil \tag{4}$$

$x_1 \geq \omega_1$ may be replaced by $x_1 \geq \lfloor \omega_1 \rfloor + 1$ because $x_1 = \omega_1$ was already included in G_0. We directly note that this region is empty if $\omega_1 = 1$. Otherwise, X_1 can be added to the CNF-formula for this region. Note that region 1 is empty or $x_1 = 1$, i.e. X_1 is true. In case the region is not empty, we use the fact that $x_1 = 1$ to get $x_k = 0$ if $\omega_1 - \omega_k \in (0, 1]$.

In general, region G_i is defined as

$$x_i \geq \lfloor \omega_i \rfloor + 1 \tag{5}$$
$$x_i - x_k \geq \lfloor \omega_i - \omega_k \rfloor + 1, \quad k < i \tag{6}$$
$$\dots \tag{7}$$
$$x_i - x_k \geq \lceil \omega_i - \omega_k \rceil, \quad k > i \tag{8}$$

Clearly, the region is empty if $\omega_i = 1$. Otherwise, we have $x_i = 1$ or X_i is true. For the case $k > i$ we have $\neg X_k$ if $\lceil \omega_i - \omega_k \rceil = 1$. For the case $k < i$, we have $\neg X_k$ if $\lfloor \omega_i - \omega_k \rfloor$ is 0 and we can conclude that the region is empty if it is 1.

Thus, generally, sets of unit literal clauses define the lattice part of the regions.

Example To illustrate this approach, take for example, $\omega = (1, \frac{3}{4}, \frac{3}{4}, 0)$. In this case the regions are as follows

$$G_0 = \{x_1 \leq 1, x_2 = x_3 = x_4 = 0\} \tag{9}$$
$$G_1 = \emptyset \tag{10}$$
$$G_2 = \{x_2 = 1, x_4 = 0, x_1, x_3 \text{ free}\} \tag{11}$$
$$G_3 = \{x_3 = 1, x_2 = x_4 = 0, x_1 \text{ free}\} \tag{12}$$
$$G_4 = \{x_4 = 1, x_1, x_2, x_3 \text{ free}\} \tag{13}$$

To see that G_1 is empty note that $\omega_1 = 1$. $x_1 \geq \lfloor \omega_1 \rfloor + 1$ implies $x_1 \geq 2$. For example, for G_2, $x_4 = 0$ follows from $\lceil \omega_2 - \omega_4 \rceil = 1$, hence $x_2 - x_4 \geq 1$. Combining with $x_2 = 1$ this gives $x_4 = 0$. Note that G_4 is much larger than the other regions.

4 Algorithm

Before starting the algorithm we apply some preprocessing procedures on the formula. The first one is, that every variable that occurs more often negated than positive is replaced by its opposite. In this way, generally, one obtains formulae with a larger number of Horn clauses. The advantage of this so-called flipping is that each variable is more likely to be true than false. Preliminary computational results showed that in some cases computation time is substantially decreased by this pre-processing step.

If we consider a particular region, we first add the corresponding clauses to the CNF-formula and apply unit propagation. If this does not lead to the conclusion that the formula is unsatisfiable, we apply single lookahead unit propagation. If even this is not succesfull in proving unsatisfiability or finding a satisfying assingment, we add the constraints imposed by the region to the linear program and solve it. This gives us a new splitting point.

Below we give a brief overview of the algorithm:
INPUT: a CNF-formula in DIMACS-format
OUTPUT: 1 if the formula is satisfiable and 0 otherwise

1. Flip all variables that now occur more negative than positive

2. Apply single look-ahead unit propagation in order to simplify the formula as much as possible
3. Flip all variables that occur more often negative than positive
4. Transform the simplified CNF-formula to a set of linear constraints as described above
5. Solve the linear program, this gives a solution vector ω
6. Starting from region 0 and continuing to region n explore any of the regions. This is done by adding the assignment made in the region to the CNF-formula and applying unit propagation. If this does not lead to a contradiction, a single unit look-ahead loop is done. If also this does not lead to a contradiction, add the linear constraints that describe the region to the LP and go to step, 5 Else, backtrack to the next unexplored node.

References

Franco J., 1997. *Relative size of certain polynomial time solvable subclasses of satisfiability*, volume 35 of *DIMACS Series Discrete Math.Theoret.Computer Science*, 211–233. Amer.Math.Soc., Piscataway, NJ.

Van Maaren H. and Dang C., 2002. Simplicial pivoting algorithms for a tractable class of integer programs. *Journal of Combinatorial Optimization*, 6, 133–142.