Carnegie Mellon University
Research Showcase

Tepper School of Business

12-1-2008

Optimization Methods in Logic

John N. Hooker
Carnegie Mellon University, john@hooker.tepper.cmu.edu

Follow this and additional works at: http://repository.cmu.edu/tepper

Part of the Economic Policy Commons, and the Industrial Organization Commons

Recommended Citation
Hooker, John N., "Optimization Methods in Logic" (2008). *Tepper School of Business.* Paper 144.
http://repository.cmu.edu/tepper/144

This Book Chapter is brought to you for free and open access by Research Showcase. It has been accepted for inclusion in Tepper School of Business by an authorized administrator of Research Showcase. For more information, please contact research-showcase@andrew.cmu.edu.

Optimization Methods in Logic

John Hooker
Carnegie Mellon University
February 2003, Revised December 2008

1 Numerical Semantics for Logic

Optimization can make at least two contributions to boolean logic. Its solution methods can address inference and satisfiability problems, and its style of analysis can reveal tractable classes of boolean problems that might otherwise have gone unnoticed.

They key to linking optimization with logic is to provide logical formulas a numerical interpretation or semantics. While *syntax* concerns the structure of logical expressions, *semantics* gives them meaning. Boolean semantics, for instance, focuses on truth functions that capture the meaning of logical propositions. To take an example, the function $f(x_1, x_2)$ given by $f(0,1) = 0$ and $f(0,0) = f(1,0) = f(1,1) = 1$ interprets the expression $x_1 \vee \bar{x}_2$, where 0 stands for "false" and 1 for "true."

The Boolean function f does not say a great deal about the meaning of $x_1 \vee \bar{x}_2$, but this is by design. The point of formal logic is to investigate how one can reason correctly based solely on the *form* of propositions. The meaning of the atoms x_1 and x_2 is irrelevant, aside from the fact either can be true or false. Only the "or" ($\vee$) and the "not" ($^-$) require interpretation for the purposes of formal logic, and the function f indicates how they behave in the expression $x_1 \vee \bar{x}_2$. In general, interpretations of logic are chosen to be as lean as possible in order to reflect only the formal properties of logical expressions.

For purposes of solving inference and satisfiability problems, however, it may be advantageous to give logical expressions a more specific meaning. This chapter presents the idea of interpreting 0 and 1 as actual numerical values rather than simply as markers for "false" and "true." Boolean truth values signify nothing beyond the fact that there are two of them, but the numbers 0 and 1 derive additional meaning from their role in mathematics. For example, they allow Boolean expressions to be regarded as inequalities, as when $x_1 \vee \bar{x}_2$ is read as $x_1 + (1 - x_2) \geq 1$.

This maneuver makes such optimization techniques as linear and 0-1 programming available to logical inference and satisfiability problems. In addition it helps to reveal the structure of logical problems and calls attention to classes of problems that are more easily solved.

George Boole seems to give a numerical interpretation of logic in his seminal work, *The Mathematical Analysis of Logic*, since he notates disjunction and conjunction with

symbols for addition and multiplication. Yet his point in doing so is to emphasize that one can calculate with propositions no less than with numbers. The notation does not indicate a numerical interpretation of logic, since Boole's main contribution is to demonstrate a *non*numeric calculus for deductive reasoning. The present chapter, however, develops the numerical interpretation suggested by Boole's original notation.

We begin by showing how Boolean inference and satisfiability problems can be solved as optimization problems. When then use the numerical interpretation of logic to identify tractable classes of satisfiability problems. We conclude with some computational considerations.

2 Solution Methods

The boolean inference problem can be straightforwardly converted to an integer programming problem and solved in that form, and we begin by showing how. It is preferable, however, to take advantage of the peculiar structure of satisfiability problems rather than solving them as general integer programming problems. We explain how to generate specialized separating cuts based on the resolution method for inference and how to isolate Horn substructure for use in branching and Benders decomposition. We also consider Lagrangean approaches that exploit the characteristics of satisfiability problems.

2.1 The Integer Programming Formulation

Recall that a *literal* has the form x_j or $\bar{x}_j$, where x_j is an atom. A *clause* is disjunction of literals. A clause C implies clause D if and only if C *absorbs* D; that is, all the literals of C occur in D.

To check whether a Boolean expression P is satisfiable, we can convert P to conjunctive normal form or CNF (a conjunction of clauses), and write the resulting clauses as linear 0-1 inequalities. P is satisfiable if and only if the 0-1 inequalities have a feasible solution, as determined by integer programming.

Example 1 *To check the proposition $x_1\bar{x}_2 \vee x_3$ for satisfiability, write it as a conjunction of two clauses, $(x_1 \vee x_3)(\bar{x}_2 \vee x_3)$, and convert them to the inequalities*

$$\begin{aligned} &x_1 + x_3 \geq 1 \\ &(1 - x_2) + x_3 \geq 1 \end{aligned} \tag{1}$$

where $x_1, x_2, x_3 \in \{0, 1\}$. An integer programming algorithm can determine that (1) has at least one feasible solution, such as $(x_1, x_2, x_3) = (0, 1, 1)$. The proposition $x_1\bar{x}_2 \vee x_3$ is therefore satisfiable.

To state this in general, let S be the set of clauses that result when P is converted to CNF. Each clause C has the form

$$C_{AB} = \bigvee_{j \in A} x_j \vee \bigvee_{j \in B} \bar{x}_j \tag{2}$$

and can be converted to the inequality

$$C^{01} = \sum_{j \in A} x_j + \sum_{j \in B} (1 - x_j) \geq 1$$

where each $x_i \in \{0, 1\}$. It is convenient to write C^{01} using the shorthand notation

$$x(A) + \bar{x}(B) \geq 1 \tag{3}$$

If S^{01} is the set of 0-1 inequalities corresponding to clauses in S, then P is *satisfiable* if and only if S^{01} has a feasible solution.

P *implies* a given clause C_{AB} if and only if the optimization problem

$$\begin{array}{ll} \text{minimize} & x(A) + \bar{x}(B) \\ \text{subject to} & S^{01} \\ & x_j \in \{0, 1\}, \text{ all } j \end{array}$$

has an optimal value of at least 1. Alternatively, P implies C_{AB} if and only if $P\bar{C}_{AB}$ is unsatisfiable. $P\bar{C}_{AB}$ is obviously equivalent to the clause set

$$S \cup \{\bar{x}_j \mid j \in A\} \cup \{x_j \mid j \in B\}$$

which can be checked for satisfiability by using 0-1 programming.

Example 2 *The proposition*

$$P = (x_1 \vee x_3 \vee x_4)(x_1 \vee x_3 \vee \bar{x}_4)(x_2 \vee \bar{x}_3 \vee x_4)(x_2 \vee \bar{x}_3 \vee \bar{x}_4)$$

implies $x_1 \vee x_2$ *if and only if the following problem has an optimal value of at least 1:*

$$\begin{array}{ll} \textit{minimize} & x_1 + x_2 \\ \textit{subject to} & x_1 + x_3 + x_4 \geq 1 \\ & x_1 + x_3 + (1 - x_4) \geq 1 \\ & x_2 + (1 - x_3) + x_4 \geq 1 \\ & x_2 + (1 - x_3) + (1 - x_4) \geq 1 \\ & x_1, \ldots, x_4 \in \{0, 1\} \end{array}$$

An integer programming algorithm can determine that the optimal value is 1.

Alternatively, P implies $x_1 \vee x_2$ if the clause set

$$\begin{array}{l} x_1 \vee x_3 \vee x_4 \\ x_1 \vee x_3 \vee \bar{x}_4 \\ x_2 \vee \bar{x}_3 \vee x_4 \\ x_2 \vee \bar{x}_3 \vee \bar{x}_4 \\ \neg x_1 \\ \neg x_2 \end{array} \tag{4}$$

is infeasible. An integer programming algorithm can determine that the corresponding 0-1 constraint set is infeasible:

$$\begin{array}{l} x_1 + x_3 + x_4 \geq 1 \\ x_1 + x_3 + (1 - x_4) \geq 1 \\ x_2 + (1 - x_3) + x_4 \geq 1 \\ x_2 + (1 - x_3) + (1 - x_4) \geq 1 \\ (1 - x_1) \geq 1 \\ (1 - x_2) \geq 1 \\ x_1, \ldots, x_4 \in \{0, 1\} \end{array} \tag{5}$$

It follows that (4) is unsatisfiable and that P implies $x_1 \vee x_2$.

A Boolean expression can be written as an inequality without first converting to CNF, but this is normally impractical for purposes of optimization. For example, one could write the proposition $x_1\bar{x}_2 \vee x_3$ as the inequality

$$x_1(1 - x_2) + x_3 \geq 1$$

This nonlinear inequality results in a much harder 0-1 programming problem than the linear inequalities that represent clauses.

The most popular solution approach for linear 0-1 programming is *branch and cut*. In its simplest form it begins by solving the *linear programming (LP) relaxation* of the problem, which results from replacing the binary conditions $x_j \in \{0, 1\}$ with ranges $0 \leq x_j \leq 1$. If all variables happen to have integral values in the solution of the relaxation, the problem is solved. If the relaxation has a nonintegral solution, the algorithm *branches* on a variable x_j with a nonintegral value. To branch is to repeat the process just described, once using the LP relaxation with the additional constraint $x_j = 0$, and a second time with the constraint $x_j = 1$. The recursion builds a finite but possibly large binary tree of LP problems, each

of which relaxes the original problem with some variables fixed to 0 or 1. The original LP relaxation lies at the root of the tree.

The leaf nodes of the search tree represent LP relaxations with an integral solution, a dominated solution, or no solution. A dominated solution is one whose value is no better than that of the best integral solution found so far (so that there is no point in branching further). The best integral solution found in the course of the search is optimal in the original problem. If no integral solution is found, the problem is infeasible. The method is obviously more efficient if it reaches leaf nodes before descending too deeply into the tree.

The branching search is often enhanced by generating *valid cuts* or *cutting planes* at some nodes of the search tree. These are inequality constraints that are added to the LP relaxation so as to "cut off" part of its feasible polyhedron without cutting off any of the feasible 0-1 points. Valid cuts tighten the relaxation and increase the probability that its solution will be integral or dominated.

The practical success of branch and bound rests on two mechanisms that may result in shallow leaf nodes.

Integrality. LP relaxations may, by luck, have integral solutions when only a few variables have been fixed to 0 or 1, perhaps due to the polyhedral structure of problems that typically arise in practice.

Bounding. LP relaxations may become tight enough to be dominated before one descends too deeply into the search tree, particularly if strong cutting planes are available.

In both cases the nature of the relaxation plays a central role. We therefore study the LP relaxation of a boolean problem.

2.2 The Linear Programming Relaxation

Linear programming provides an incomplete check for boolean satisfiability. For clause set S, let S^{LP} denote the LP relaxation of the 0-1 formulation S^{01}. If S^{LP} is infeasible, then S is unsatisfiable. If S^{LP} is feasible, however, the satisfiability question remains unresolved.

This raises the question, how much power does linear programming have to detect unsatisfiability when it exists? It has the same power as a simple inference method known as unit resolution, in the sense that the LP relaxation is infeasible precisely when unit resolution proves unsatisfiability.

Unit resolution is a linear-time inference procedure that essentially performs back substitution. Each step of unit resolution fixes a *unit clause* $U \in S$ (i.e., a single-literal clause) to true. It then eliminates U and $\bar{U}$ from the problem by removing from S all clauses that

contain the literal U, and removing the literal $\bar{U}$ from all clauses in S that contain it. The procedure repeats until no unit clauses remain. It may result in an *empty clause* (a clause without literals), which is a sufficient condition for the unsatisfiability of S.

Since unit resolution is much faster than linear programming, it makes sense to apply unit resolution before solving the LP relaxation. Linear programming is therefore used only when the relaxation is known to be feasible and is not a useful check for satisfiability. It is more useful for providing bounds and finding integral solutions. Let a *unit refutation* be a unit resolution proof that obtains the empty clause.

Proposition 1 *A clause set S has a unit refutation if and only if S^{LP} is infeasible.*

Two examples motivate the proof of the proposition. The first illustrates why the LP relaxation is feasible when unit resolution fails to detect unsatisfiability.

Example 3 *Consider the clause set (4). Unit resolution fixes $x_1 = x_2 = 0$ and leaves the clause set*

$$\begin{array}{l} x_3 \vee x_4 \\ x_3 \vee \bar{x}_4 \\ \bar{x}_3 \vee x_4 \\ \bar{x}_3 \vee \bar{x}_4 \end{array} \tag{6}$$

Unit resolution therefore fails to detect the unsatisfiability of (4). To see that the LP relaxation (5) is feasible, set the unfixed variables x_3 and x_4 to 1/2. This is a feasible solution (regardless of the values of the fixed variables) because every inequality in (5) has at least two unfixed terms of the form x_j or $1 - x_j$.

We can also observe that unit resolution has the effect of summing inequalities.

Example 4 *Suppose we apply unit resolution to the first three clauses below to obtain the following .*

$$\begin{array}{lll} \bar{x}_1 & (a) & \\ x_1 \vee x_2 & (b) & \\ x_1 \vee \bar{x}_2 & (c) & \\ \hline x_2 & (d) & \textit{from } (a) + (b) \\ \bar{x}_2 & (e) & \textit{from } (a) + (c) \\ \emptyset & & \textit{from } (d) + (e) \end{array} \tag{7}$$

Resolvent (d) corresponds to the sum of $1 - x_1 \geq 1$ and $x_1 + x_2 \geq 1$, and similarly for resolvent (e). Now x_2 and $\bar{x}_2$ resolve to produce the empty clause, which corresponds to summing $x_2 \geq 1$ and $(1 - x_2) \geq 1$ to get $0 \geq 1$. Thus applying unit resolution to (7) has the effect of taking a nonnegative linear combination $2 \cdot (a) + (b) + (c)$ to yield $0 \geq 1$.

Proof of Proposition 1. If unit resolution fails to demonstrate unsatisfiability of S, then it creates no empty clause, and every remaining clause contains at least two literals. Thus every inequality in S^{LP} contains at least two unfixed terms of the form x_j or $1 - x_j$ and can be satisfied by setting the unfixed variables to 1/2.

Conversely, if unit resolution proves unsatisfiability of S, then some nonnegative linear combination of inequalities in S^{01} yields $0 \geq 1$. This means that S^{LP} is infeasible.

There are special classes of boolean problems for which unit resolution always detects unsatisfiability when it exists. Section 3 shows how polyhedral analysis can help identify such classes.

We can now consider how the branch and cut mechanisms discussed above might perform in the context of Boolean methods.

Integrality. Since one can always solve a feasible LP relaxation of a satisfiability problem by setting unfixed variables to 1/2, it may appear that the integrality mechanism will not work in the Boolean case. However, the simplex method for linear programming finds a solution that lies at a vertex of the feasible polyhedron. There may be many integral vertex solutions, and the solution consisting of 1/2's may not even be a vertex. For instance, the fractional solution $(x_1, x_2, x_3) = (1/2, 1/2, 1/2)$ is feasible for the LP relaxation of (1), but it is not a vertex solution. In fact, all of the vertex solutions are integral (i.e., the LP relaxation defines an *integral polyhedron*). The integrality mechanism can therefore be useful in a Boolean context, albeit only empirical investigation can reveal how useful it is.

Bounding. The bounding mechanism is more effective if we can identify cutting planes that exploit the structure of Boolean problems. In fact we can, as shown in the next section.

2.3 Cutting Planes

A cutting plane is a particular type of logical implication. We should therefore expect to see a connection between cutting plane theory and logical inference, and such a connection exists. The most straightforward link is the fact that the well-known resolution method for inference generates cutting planes.

Resolution generalizes the unit resolution procedure discussed above (see Chapter 3 of Volume I). Two clauses C, D have a *resolvent* R if exactly one variable x_j occurs as a positive literal x_j in one clause and as a negative literal $\bar{x}_j$ in the other. The resolvent R consists of all the literals in C or D except x_j and $\bar{x}_j$. C and D are the *parents* of R. For example, the clauses $x_1 \vee x_2 \vee x_3$ and $\bar{x}_1 \vee x_2 \vee \bar{x}_4$ yield the resolvent $x_2 \vee x_3 \vee \bar{x}_4$.

Given a clause set S, each step of the resolution algorithm finds a pair of clauses in S that have a resolvent R that is absorbed by no clause in S, removes from S all clauses absorbed by R, and adds R to S. The process continues until no such resolvents exist. Quine [66, 67] showed that the resolution procedure yields precisely the prime implicates of S. It yields the empty clause if and only if S is unsatisfiable.

Unlike a unit resolvent, a resolvent R of C and D need not correspond to the sum of C^{01} and D^{01}. However, it corresponds to a cutting plane. A *cutting plane* of a system $Ax \geq b$ of 0-1 inequalities is an inequality that is satisfied by all 0-1 points that satisfy $Ax \geq b$. A *rank 1* cutting plane has the form $\lceil uA \rceil x \geq \lceil ub \rceil$, where $u \geq 0$ and $\lceil \alpha \rceil$ rounds α up to the nearest integer. Thus rank 1 cuts result from rounding up a nonnegative linear combination of inequalities.

Proposition 2 *The resolvent of clauses C, D is a rank 1 cutting plane for C^{01}, D^{01} and bounds $0 \leq x_j \leq 1$.*

The reasoning behind the proposition is clear in an example.

Example 5 *Consider again the clauses $x_1 \vee x_2 \vee x_3$ and $\bar{x}_1 \vee x_2 \vee \bar{x}_4$, which yield the resolvent $x_2 \vee x_3 \vee \bar{x}_4$. Consider a linear combination of the corresponding 0-1 inequalities and 0-1 bounds, where each inequality has a multiplier 1/2:*

$$\begin{array}{rl}
x_1 + x_2 + x_3 & \geq 1 \\
(1-x_1) + x_2 + (1-x_4) & \geq 1 \\
x_3 & \geq 0 \\
(1-x_4) & \geq 0 \\
\hline
x_2 + x_3 + (1-x_4) & \geq 1/2
\end{array}$$

Rounding up the right-hand side of the resulting inequality (below the line) yields a rank 1 cutting plane that corresponds to the resolvent $x_2 \vee x_3 \vee \bar{x}_4$.

Proof of Proposition 2. Let $C = C_{A \cup \{k\}, B}$ and $D = D_{A', B' \cup \{k\}}$, so that the resolution takes place on variable x_k. The resolvent is $R = R_{A \cup A', B \cup B'}$. Consider the linear combination of C^{01}, D^{01}, $x_j \geq 0$ for $j \in A \Delta A'$, and $(1 - x_j) \geq 0$ for $j \in B \Delta B'$ in which each inequality has weight 1/2, and $A \Delta A'$ is the symmetric difference of A and A'. This linear combination is $\sum_{j \in A \cup A'} x_j + \sum_{j \in B \cup B'} (1 - x_j) \geq 1/2$. By rounding up the right-hand side, we obtain R^{01}, which is therefore a rank 1 cut.

Let the *input resolution* algorithm for a given clause set S be the resolution algorithm applied to S with the restriction that at least one parent of every resolvent belongs to the original set S. The following is proved in [38].

Proposition 3 *The input resolution algorithm applied to a clause set S generates precisely the set of clauses that correspond to rank 1 cuts of S^{01}.*

One way to obtain cutting planes for branch and cut is to generate resolvents from the current clause set S. The number of resolvents tends to grow very rapidly, however, and most of them are likely to be unhelpful. Some criterion is needed to identify useful resolvents, and the numerical interpretation of logic provides such a criterion. One can generate only *separating cuts*, which are cutting planes that are violated by the solution of the current LP relaxation. Separating cuts are so called because they cut off the solution of the LP relaxation and thereby "separate" it from the set of feasible 0-1 points.

The aim, then, is to identify resolvents R for which R^{01} is a separating cut. In principle this can be done by screening all resolvents for separating cuts, but there are better ways. It is straightforward, for example, to recognize a large class of clauses that cannot be the parent of a separating resolvent. Suppose that all clauses under discussion have variables in $\{x_1, \ldots, x_n\}$.

Proposition 4 *Consider any clause C_{AB} and any $x \in [0,1]^n$. C_{AB} can be the parent of a separating resolvent for x only if $x(A) + \bar{x}(B) - x(\{j\}) < 1$ for some $j \in A$ or $x(A) + \bar{x}(B) - \bar{x}(\{j\}) < 1$ for some $j \in B$.*

Proof. The resolvent on x_j of clause C_{AB} with another clause has the form $R = C_{A'B'}$, where $A \setminus \{j\} \subset A'$ and $B \setminus \{j\} \subset B'$. R^{01} is separating when $x(A') + \bar{x}(B') < 1$. This implies that $x(A) + \bar{x}(B) - x(\{j\}) < 1$ if $j \in A$ and $x(A) + \bar{x}(B) - \bar{x}(\{j\}) < 1$ if $j \in B$.

Example 6 *Suppose $(x_1, x_2, x_3) = (1, 0.4, 0.3)$ in the solution of the current LP relaxation of an inference or satisfiability problem, and let $C_{AB} = x_1 \vee x_2 \vee \bar{x}_3$. Here $x(A) + \bar{x}(B) = 2.1$. C_{AB} cannot be the parent of a separating resolvent because $x(A) + \bar{x}(B) - x(\{1\}) = 1.1 \geq 1$, $x(A) + \bar{x}(B) - x(\{2\}) = 1.7 \geq 1$, and $x(A) + \bar{x}(B) - \bar{x}(\{3\}) = 1.4 \geq 1$.*

Proposition 4 suggests that one can apply the following separation algorithm at each node of the branch-and-cut tree. Let S_0 consist of the clauses in the satisfiability or inference problem at the current node, including unit clauses imposed by branching. Simplify S_0 by applying unit resolution, and solve S_0^{LP} (unless S_0 contains the empty clause). If the LP solution is nonintegral and undominated, generate cutting planes as follows. Let set S^{SR} (initially empty) collect separating resolvents. Remove from S_0 all clauses that cannot be parents of separating resolvents, using the criterion of Proposition 4. In each iteration k, beginning with $k = 1$, let S_k be initially empty. Generate all resolvents R that have both parents in S_{k-1} and are not dominated by any clause in S_{k-1}. If R^{01} is

separating, remove from S^{SR} all clauses absorbed by R, and put R into S_k and S^{SR}; if R^{01} is not separating, put R into S_k if R meets the criterion in Proposition 4. If the resulting set S_k is nonempty, increment k by one and repeat. The following is proved in [45].

Proposition 5 *Given a clause set S and a solution x of S^{LP}, every clause corresponding to a rank 1 separating cut of S^{01} is absorbed by a clause in S^{SR}.*

S^{SR} may also contain cuts of higher rank.

Example 7 *Consider the problem of checking whether x_1 follows from the following clause set S:*

$$\begin{array}{ll} x_1 \vee x_2 \vee x_3 & (a) \\ x_1 \vee x_2 \vee \bar{x}_3 \vee x_4 & (b) \\ x_1 \vee \bar{x}_2 \qquad \vee x_4 & (c) \\ \qquad \bar{x}_2 \vee x_3 \vee \bar{x}_4 & (d) \end{array} \tag{8}$$

The 0-1 problem is to minimize x_1 subject to S^{01}. At the root node of the branch-and-cut tree, we first apply unit resolution to $S_0 = S$, which has no effect. We solve S_0^{LP} to obtain $(x_1, x_2, x_3, x_4) = (0, 1/3, 2/3, 1/3)$. Only clauses (a)-(c) satisfy Proposition 4, and we therefore remove (d) from S_0. In iteration $k = 1$ we generate the following resolvents from S_0:

$$\begin{array}{ll} x_1 \vee x_2 \qquad \vee x_4 & (e) \\ x_1 \qquad \vee \bar{x}_3 \vee x_4 & (f) \\ x_1 \qquad \vee x_3 \vee x_4 & (g) \end{array}$$

Resolvents (e) and (f) are separating and are added to both S_1 and S^{SR}. Resolvent (g) passes the test of Proposition 4 and is placed in S_1, which now contains (e), (f) and (g).

In iteration $k = 2$ we generate the resolvent $x_1 \vee x_4$ from S_1. Since it is separating and absorbs both (e) and (f), the latter two clauses are removed from S^{SR} and $x_1 \vee x_4$ is added to S^{SR}. Also $x_1 \vee x_4$ becomes the sole element of S_2. Clearly S_3 is empty, and the process stops with one separating clause in S^{SR}, namely $x_1 \vee x_4$. It corresponds to the rank 1 cut $x_1 + x_4 \geq 1$.

At this point the cut $x_1 + x_4 \geq 1$ can be added to S_0^{LP}. If the LP is re-solved, an integral solution $(x_1, \ldots, x_4) = (0, 0, 1, 1)$ results, and the 0-1 problem is solved without branching. Since $x_1 = 0$ in this solution, x_1 does not follow from S.

2.4 Horn Problems

A promising approach to solving Boolean satisfiability problems is to exploit the fact that they often contain large "renamable Horn" subproblems. That is, fixing to true or false a

few atoms $x_1, \ldots, x_p$ may create a renamable Horn clause set, which unit resolution can check for satisfiability in linear time. Thus when using a branching algorithm to check for satisfiability, one need only branch on variables $x_1, \ldots, x_p$. After one branches on these variables, the remaining subproblems are all renamable Horn and can be solved without branching.

Originally proposed by Chandru and Hooker [16], this approach is a special case of the more general strategy of finding a small set of variables that, when fixed, simplify the problem at hand. The idea later re-emerged in the literature under the name of finding a *backdoor* [25, 52, 51, 59, 64, 65, 72, 76] for boolean satisfiability and other problems. Complexity results have been derived for several types of backdoor detection [25, 59, 76].

A *Horn clause* is a clause with at most one positive literal, such as $\bar{x}_1 \vee \bar{x}_2 \vee x_3$. A clause set H is Horn if all of its clauses are Horn. H is *renamable Horn* if it becomes Horn when zero or more variables x_j are complemented by replacing them with $\bar{x}_j$ (and $\bar{x}_j$ with x_j). Horn problems are discussed further in Chapter 6 of Volume I.

Example 8 *The clause set*

$$\begin{array}{c} x_1 \vee x_2 \\ \bar{x}_1 \vee \bar{x}_2 \end{array}$$

is renamable Horn because complementing x_1 makes it Horn. On the other hand, the following clause set is not renamable Horn.

$$\begin{array}{c} x_1 \vee x_2 \vee x_3 \\ \bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \end{array} \tag{9}$$

Let *positive unit resolution* be a unit resolution algorithm in which every resolvent has at least one parent that is a positive unit clause.

Proposition 6 *A Horn set is satisfiable if and only if there is no positive unit refutation. A renamable Horn set is satisfiable if and only if there is no unit refutation.*

Proof. Suppose that positive unit resolution applied to Horn clause set S generates no empty clause. Let S' be the clause set that remains. Then every clause in S' is a negative unit cause or contains two or more literals, at least one of which is negative. The clauses in S can be satisfied by setting all variables in S' to false and all variables on which the algorithm resolved to true. Now suppose that unit resolution applied to a renamable Horn clause set S generates no empty clause. For some renaming of the variables in S, the resulting clause set S^+ is Horn. There is no positive unit refutation of S^+, because otherwise, un-renaming the variables in this proof would yield a unit refutation of S. Thus S^+ is satisfiable, which means that S is satisfiable.

From this and Proposition 1 we immediately infer:

Proposition 7 *A renamable Horn set S is satisfiable if and only if S^{LP} is feasible.*

Given a clause set S, we would like to find a smallest backdoor set. In particular, we seek the shortest vector of variables $x' = (x_{j_1}, \ldots, x_{j_p})$ that, when fixed to any value, simplifies S to renamable Horn set. Following [16], let $v = (v_1, \ldots, v_p)$ be a 0-1 vector, and define $S(x', v)$ to be the result of fixing each x_{j_ℓ} in x' to v_ℓ. Thus if $v_\ell = 1$, all clauses in S containing literal x_{j_ℓ} are removed from S, and all literals $\bar{x}_{j_\ell}$ are removed from clauses in S, and analogously if $v_\ell = 0$. Thus we wish to find the smallest variable set x' such that $S(x', v)$ is renamable Horn for all valuations v.

Let $S(x')$ be the result of removing from clauses in S every literal that contains a variable in x'. Then since $S(x', v) \subset S(x')$ for every v, $S(x', v)$ is renamable Horn for every v if $S(x')$ is renamable Horn. To find the shortest x' for which $S(x')$ is renamable Horn, we introduce 0-1 variables $y_j, \bar{y}_j$. Let $y_j = 1$ when x_j is not renamed, $\bar{y}_j = 1$ when x_j is renamed, and $y_j = \bar{y}_j = 0$ when x_j is removed from S by including it in x'. Then we wish to solve the set packing problem,

$$\begin{array}{ll} \text{maximize} & \sum_j (y_j + \bar{y}_j) \\ \text{subject to} & y(A) + \bar{y}(B) \leq 1, \quad \text{all clauses } C_{AB} \in S \\ & y_j + \bar{y}_j \leq 1, \quad \text{all } j \\ & y_j, \bar{y}_j \in \{0, 1\} \end{array} \tag{10}$$

The first constraint ensures that each renamed clause in S contains at most one positive literal.

A set packing problem can always be solved as a maximum clique problem. In this case, we define an undirected graph G that contains a vertex for each literal $x_j, \bar{x}_j$ and an edge between two vertices whenever the corresponding literals never occur in the same clause of S and are not complements of each other. A clique of G is a set of vertices in which every pair of vertices is connected by an edge. If W is a clique of maximum size, then we put x_j into x' when $x_j, \bar{x}_j \notin W$. The maximum clique problem is NP-hard, but there are numerous exact and approximate algorithms for it [2, 3, 4, 5, 9, 12, 27, 28, 48, 61, 62, 70, 79]. One can also solve the maximum independent set problem on the complementary graph [11, 26, 37, 53, 63, 73]. Two graph-based algorithms specifically for finding a small backdoor set are presented in [52], and heuristic methods in [64, 65].

Example 9 *Let us check again whether x_1 follows from the clause set (8). This time we*

do so by checking the satisfiability of

$$\begin{array}{l} x_1 \vee x_2 \vee x_3 \\ x_1 \vee x_2 \vee \bar{x}_3 \vee x_4 \\ x_1 \vee \bar{x}_2 \qquad \vee x_4 \\ \qquad \bar{x}_2 \vee x_3 \vee \bar{x}_4 \\ \bar{x}_1 \end{array} \tag{11}$$

To find the shortest vector x' of variables we must fix to obtain a Horn problem, we solve the set packing problem

$$\begin{array}{lllllllll} \textit{maximize} & y_1 + y_2 + y_3 + y_4 + \bar{y}_1 + \bar{y}_2 + \bar{y}_3 + \bar{y}_4 & & & & & & & \\ \textit{subject to} & y_1 + y_2 + y_3 & & & & & & & \leq 1 \\ & y_1 + y_2 & & + y_4 & & & + \bar{y}_3 & & \leq 1 \\ & y_1 & & + y_4 & & + \bar{y}_2 & & & \leq 1 \\ & & y_3 & & & + \bar{y}_2 & & + \bar{y}_4 & \leq 1 \\ & & & & \bar{y}_1 & & & & \leq 1 \\ & y_j + \bar{y}_j \leq 1, \quad j = 1, \ldots, 4 & & & & & & & \\ & y_j \in \{0, 1\} & & & & & & & \end{array}$$

An optimal solution is $y = (0, 0, 0, 0)$, $\bar{y} = (1, 1, 1, 0)$. The solution indicates that only x_4 need be fixed to obtain a renamable Horn problem, which is converted to Horn by renaming x_1, x_2, x_3. Alternatively, we can find a maximum clique in the graph of Fig. 1. One such clique is $\{\bar{x}_1, \bar{x}_2, \bar{x}_3\}$, which again indicates that x_4 can be fixed and x_1, x_2, x_3 renamed to obtain a Horn problem.

We therefore branch only on x_4. If we set $x_4 = 0$, the resulting problem is renamable Horn:

$$\begin{array}{l} x_1 \vee x_2 \vee x_3 \\ x_1 \vee x_2 \vee \bar{x}_3 \\ x_1 \vee \bar{x}_2 \\ \bar{x}_1 \end{array} \tag{12}$$

and unit resolution proves unsatisfiability. Taking the $x_4 = 1$ branch, we obtain the renamable Horn problem

$$\begin{array}{l} x_1 \vee x_2 \vee x_3 \\ \qquad \bar{x}_2 \vee x_3 \\ \bar{x}_1 \end{array} \tag{13}$$

Unit resolution fixes $x_1 = 0$ and leaves the clause set $\{x_2 \vee x_3, \bar{x}_2 \vee x_3\}$, whose clauses we satisfy by setting its variables to 0 in the renamed problem (which is Horn). Since x_2, x_3 are renamed in this case, we set $(x_1, x_2, x_3) = (0, 1, 1)$, which with $x_4 = 1$ is a satisfying solution for (11).

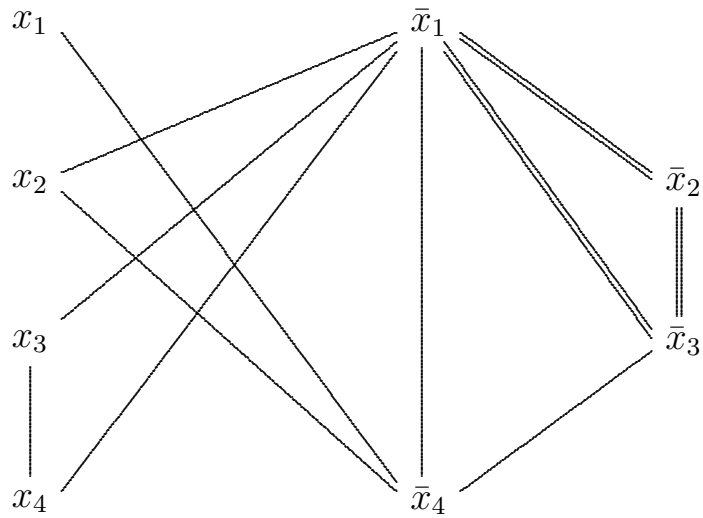

Figure 1: *Maximum clique problem for finding renamable Horn substructure. One solution is shown in double lines.*

2.5 Benders Decomposition

Benders decomposition [6, 29] is a well-known optimization technique that can be applied to Boolean satisfiability problems, particularly those with significant renamable Horn substructure.

Benders decomposition normally applies to optimization problems, but we focus on feasibility problems of the form

$$\begin{array}{l} Ax + g(y) \geq b \\ x \in R^n,\ y \in Y \end{array} \tag{14}$$

where $g(y)$ is a vector of functions $g_i(y)$. The method enumerates some values of $y \in Y$ and, for each, seeks an x such that (x, y) is feasible. Thus for each trial value $\hat{y}$, we solve the linear programming *subproblem* that results from fixing y to $\hat{y}$ in (14):

$$\begin{array}{l} Ax \geq b - g(\hat{y}) \\ x \in R^n \end{array} \tag{15}$$

If a solution $\hat{x}$ exists, then $(\hat{x}, \hat{y})$ solves (14). If the subproblem is infeasible, then by the well-known Farkas Lemma there is a row vector $\hat{u} \geq 0$ such that

$$\begin{array}{l} \hat{u}A = 0 \\ \hat{u}(b - g(\hat{y})) > 0 \end{array}$$

Thus to obtain a feasible subproblem, we must find a y such that

$$\hat{u}(b - g(y)) \leq 0 \tag{16}$$

We therefore impose (16) as *Benders cut* that any future trial value of y must satisfy.

In iteration k of the Benders algorithm we obtain a trial value $\hat{y}$ of y by solving a *master problem* that contains all Benders cuts so far generated:

$$\begin{array}{l} \hat{u}^\ell(b - g(y)) \leq 0, \ \ell = 1, \ldots, k-1 \\ y \in Y \end{array} \tag{17}$$

Thus $\hat{u}^1, \ldots, \hat{u}^{k-1}$ are the vectors $\hat{u}$ found in the previous subproblems. We then formulate a new subproblem (15) and continue the process until the subproblem is feasible or until the master problem becomes infeasible. In the latter case, the original problem (14) is infeasible.

The classical Benders method does not apply to general Boolean satisfiability problems, because the Benders subproblem must be a continuous linear or nonlinear problem for which the multipliers u can be derived. Logic-based Benders decomposition [80, 40, 46], an extension of the method, allows solution of general Boolean problems, but here we consider a class of Boolean problems in which the logic-based method reduces to the classical method. Namely, we solve Boolean problems in which the subproblem is a renamable Horn problem and therefore equivalent to an LP problem (Proposition 7).

To obtain a renamable Horn subproblem, we must let the master problem contain variables that, when fixed, result in a renamable Horn structure. This can be accomplished by the methods of the previous section.

It is inefficient to solve the subproblem with an LP solver, since unit resolution is much faster. Fortunately, we can obtain u from a unit refutation, as illustrated by Example 4. Recall that in this example the empty clause was obtained by taking the linear combination $2 \cdot (a) + (b) + (c)$, where the coefficients represent the number of times each of the original clauses are "used" in the unit refutation. To make this precise, let C be any clause that occurs in the unit refutation, and let $n(C)$ be the number of times C is used in the refutation. Initially each $n(C) = 0$ and we execute the following recursive procedure by calling count($\emptyset$), where $\emptyset$ denotes the empty clause.

Procedure count(C)
  Increase $n(C)$ by 1.
  If C has parents P, Q in the unit refutation, then
    Perform count(P) and count(Q).

The following is a special case of a result of Jeroslow and Wang [50].

Proposition 8 *Suppose the feasibility problem $Ax \geq b$ represents a renamable Horn satisfiability problem, where each row i of $Ax \geq b$ corresponds to a clause C_i. If the satisfiability problem has a unit refutation, then $uA = 0$ and $ub = 1$, where each $u_i = n(C_i)$.*

Proof. When C is the resolvent of P, Q in the above recursion, the inequality C^{01} is the sum of P^{01} and Q^{01}. Thus $\emptyset^{01}$, which is the inequality $0 \geq 1$, is the sum of $n(C_i) \cdot C_i^{01}$ over all i. Because C_i^{01} is row i of $Ax \geq b$, this means that $uA = 0$ and $ub = 1$ if we set $u_i = n(C_i)$ for all i.

Benders decomposition can now be applied to a Boolean satisfiability problem. First put the problem in the form of a 0-1 programming problem (14). The variables y are chosen so that, when fixed, the resulting subproblem (15) represents a renamable Horn satisfiability problem and can therefore be regarded as an LP problem. The multipliers $\hat{u}$ are obtained as in Proposition 8 and the Benders cut (16) formed accordingly.

Example 10 *Consider again the satisfiability problem (11). We found in Example 9 that the problem becomes renamable Horn when x_4 is fixed to any value. We therefore put x_4 in the master problem and x_1, x_2, x_3 in the subproblem. Now (11) becomes the 0-1 feasibility problem*

$$\begin{array}{llllll}
x_1 & +x_2 & +x_3 & & & \geq 1 \\
x_1 & +x_2 & +(1-x_3) & +x_4 & & \geq 1 \\
x_1 & +(1-x_2) & & +x_4 & & \geq 1 \\
 & (1-x_2) & +x_3 & & +(1-x_4) & \geq 1 \\
(1-x_1) & & & & & \geq 1
\end{array}$$

where x_4 plays the role of y_1 in (14). This problem can be written in the form (14) by bringing all constants to the right-hand side:

$$\begin{array}{llll}
x_1 & +x_2 & +x_3 & & \geq 1 \\
x_1 & +x_2 & -x_3 & +x_4 & \geq 0 \\
x_1 & -x_2 & & +x_4 & \geq 0 \\
 & -x_2 & +x_3 & -x_4 & \geq -1 \\
-x_1 & & & & \geq 0
\end{array}$$

Initially the master problem contains no constraints, and we arbitrarily set $\hat{x}_4 = 0$. This yields the satisfiability subproblem (12), which has the 0-1 form

$$\begin{array}{llll}
x_1 & +x_2 & +x_3 & \geq 1 & (a) \\
x_1 & +x_2 & -x_3 & \geq 0 & (b) \\
x_1 & -x_2 & & \geq 0 & (c) \\
 & -x_2 & +x_3 & \geq -1 & (d) \\
-x_1 & & & \geq 0 & (e)
\end{array}$$

(Note that clause (d) could be dropped because it is already satisfied.) This is a renamable Horn problem with a unit refutation, as noted in Example 9. The refutation obtains the empty clause from the linear combination $(a) + (b) + 2 \cdot (c) + 4 \cdot (e)$*, and we therefore let* $\hat{u} = (1, 1, 2, 0, 4)$*. The Benders cut (16) becomes*

$$[1\ 1\ 2\ 0\ 4] \left(\begin{bmatrix} 1 \\ 0 \\ 0 \\ -1 \\ 0 \end{bmatrix} - \begin{bmatrix} 0 \\ x_4 \\ x_4 \\ -x_4 \\ 0 \end{bmatrix} \right) \leq 0$$

or $3x_4 \geq 1$*. The master problem (17) now consists of the Benders cut* $3x_4 \geq 1$ *and has the solution* $x_4 = 1$*. The next subproblem is therefore (13), for which there is no unit refutation. The subproblem is solved as in Example 9 by setting* $(x_1, x_2, x_3) = (0, 1, 1)$*, and the Benders algorithm terminates. These values of* x_j *along with* $x_4 = 1$ *solve the original problem.*

Rather than solve each master problem from scratch as in the classical Benders method, we can conduct a single branch-and-cut search and solve the subproblem each time a feasible solution is found. When a new Benders cut is added, the branch-and-cut search resumes where it left off with the new Benders cut in the constraint set. This approach was proposed in [40] and tested computationally on machine scheduling problems in [74], where it resulted in at least an order-of-magnitude speedup.

Although we have used Benders to take advantage of Horn substructure, it can also exploit other kinds of structure by isolating it in the subproblem. For example, a problem may decompose into separate problems when certain variables are fixed [46].

2.6 Lagrangean Relaxation

One way to strengthen the relaxations solved at nodes of a search tree is to replace LP relaxations with Lagrangean relaxations. A Lagrangean relaxation removes or "dualizes" some of the constraints by placing penalties for their violation in the objective function. The dualized constraints are typically chosen in such a way that the remaining constraints decouple into small problems, allowing rapid solution despite the fact that they are discrete. Thus while Benders decomposition decouples the problem by removing variables, Lagrangean relaxation decouples by removing constraints.

Consider an optimization problem:

$$\begin{array}{ll} \text{minimize} & cx \\ \text{subject to} & Ax \geq b \\ & x \in X \end{array} \tag{18}$$

where $Ax \geq b$ represent the constraints to be dualized, and $x \in X$ represents constraints that are easy in some sense, perhaps because they decouple into small subsets of constraints that can be treated separately. We regard $Ax \geq b$ as consisting of rows $A_i x \geq b_i$ for $i = 1, \ldots, m$. Given *Lagrange multipliers* $\lambda_1, \ldots, \lambda_m \geq 0$, the following is a *Lagrangean relaxation* of (18):

$$\begin{array}{ll} \text{minimize} & cx + \sum_i \lambda_i (b_i - A_i x) \\ \text{subject to} & x \in X \end{array} \tag{19}$$

The constraints $Ax \geq b$ are therefore dualized by augmenting the objective function with weighted penalties $\lambda_i(b_i - A_i x)$ for their violation.

As for the choice of multipliers $\lambda = (\lambda_1, \ldots, \lambda_m)$, the simplest strategy is to set each to some convenient positive value, perhaps 1. One can also search for values of λ that yield a tighter relaxation. If $\theta(\lambda)$ is the optimal value of (19), the best possible relaxation is obtained by finding a λ that solves the *Lagrangean dual* problem $\max_{\lambda \geq 0} \theta(\lambda)$. Since $\theta(\lambda)$ is a concave function of λ, it suffices to find a local maximum, which is a global maximum.

Subgradient optimization is commonly used to solve the Lagrangean dual ([78], pp. 174–175). Each iteration k begins with the current estimate λ^k of λ. Problem (19) is solved with $\lambda = \lambda^k$ to compute $\theta(\lambda^k)$. If x^k is the solution of (19), then $b - Ax^k$ is a subgradient of $\theta(\lambda)$ at λ^k. The subgradient indicates a direction in which the current value of λ should move to achieve the largest possible rate of increase in $\theta(\lambda)$. Thus we set $\lambda^{k+1} = \lambda^k + \alpha_k(b - Ax^k)$, where α_k is a stepsize that decreases as k increases. Various stopping criteria are used in practice, and the aim is generally to obtain only an approximate solution of the dual.

For a Boolean satisfiability problem, $c = 0$ in (18), and the constraints $Ax \geq b$ are 0-1 formulations of clauses. The easy constraints $x \in X$ include binary restrictions $x_j \in \{0, 1\}$ as well as "easy" clauses that allow decoupling. When the Lagrangean relaxation has a positive optimal value $\theta(\lambda)$, the original problem is unsatisfiable, whereas $\theta(\lambda) \leq 0$ leaves the optimality question unresolved.

Example 11 *Consider the unsatisfiable clause set S below:*

$$\begin{array}{lll} \bar{x}_1 \vee \bar{x}_2 \vee x_3 & & (a) \\ \bar{x}_1 \vee \bar{x}_2 & \vee \bar{x}_4 & (b) \\ x_1 \vee x_2 & & (c) \\ x_1 \vee \bar{x}_2 & & (d) \\ \bar{x}_1 \vee x_2 & & (e) \\ & \bar{x}_3 \vee x_4 & (f) \\ & x_3 \vee \bar{x}_4 & (g) \end{array} \tag{20}$$

S^{LP} is feasible and therefore does not demonstrate unsatisfiability. Thus if we use the LP relaxation in a branch-and-cut algorithm, branching is necessary. A Lagrangean relaxation, however, can avoid branching in this case. Constraints (a) and (b) are the obvious ones to dualize, since the remainder of the problem splits into two subproblems that can be solved separately. The objective function of the Lagrangean relaxation (19) becomes

$$\lambda_1(-1 + x_1 + x_2 - x_3) + \lambda_2(-2 + x_1 + x_2 + x_4)$$

Collecting terms in the objective function, the relaxation (19) is

$$\begin{array}{lrrrrl} \textit{minimize} & (\lambda_1 + \lambda_2)x_1 & + (\lambda_1 + \lambda_2)x_2 & -\lambda_1 x_3 & + \lambda_2 x_4 & -\lambda_1 - 2\lambda_2 \\ \textit{subject to} & x_1 & + x_2 & & & \geq 1 \\ & x_1 & - x_2 & & & \geq 0 \\ & -x_1 & + x_2 & & & \geq 0 \\ & & & -x_3 & + x_4 & \geq 0 \\ & & & x_3 & - x_4 & \geq 0 \\ & x_j \in \{0,1\} & & & & \end{array} \tag{21}$$

The relaxation obviously decouples into two separate subproblems, one containing x_1, x_2 and one containing x_3, x_4.

Starting with $\lambda^0 = (1,1)$, we obtain an optimal solution $x^0 = (1,1,0,0)$ of (21) with $\theta(\lambda^0) = 1$. This demonstrates that (20) is unsatisfiable, without the necessity of branching.

Bennaceur et al. [7] use a somewhat different Lagrangean relaxation to help solve satisfiability problems. They again address the problem with a branching algorithm that solves a Lagrangean relaxation at nodes of the search tree (or at least at certain nodes). This time, the dualized constraints $Ax \geq b$ are the clauses violated by the currently fixed variables, and the remaining constraints $x \in X$ are the clauses that are already satisfied. The multipliers λ_i are all set to 1, with no attempt to solve the Lagrangean dual. A

local search method approximately solves the resulting relaxation (19). If the solution $\hat{x}$ satisfies additional clauses, the process is repeated while dualizing only the clauses that remain violated. This continues until no additional clauses can be satisfied. If all clauses are satisfied, the algorithm terminates. Otherwise, branching continues in the manner indicated by $\hat{x}$. That is, when branching on x_j, one first takes the branch corresponding to $x_j = \hat{x}_j$. This procedure is combined with intelligent backtracking to obtain a competitive satisfiability algorithm, as well as an incremental satisfiability algorithm that re-solves the problem after adding clauses. The details may be found in [7].

3 Tractable Problem Classes

We now turn to the task of using the quantitative analysis of logic to identify tractable classes of satisfiability problems. We focus on two classes: problems that can be solved by unit resolution, and problems whose LP relaxations define integral polyhedra.

3.1 Two Properties of Horn Clauses

There is no known necessary and sufficient condition for solubility by unit resolution, but some sufficient conditions are known. We have already seen that Horn and renamable Horn problems, for example, can be solved in this manner (Proposition 6). Two properties of Horn sets account for this, and they are actually possessed by a much larger class of problems. This allows a generalization of Horn problems to *extended Horn* problems that can likewise be solved by unit resolution, as shown by Chandru and Hooker [15].

Unit resolution is adequate to check for satisfiability when we can always find a satisfying solution for the clauses that remain after applying unit resolution to a satisfiable clause set. We can do this in the case of Horn problems because:

- Horn problems are *closed under deletion and contraction*, which ensures that the clauses that remain after unit resolution are still Horn.
- Horn problems have a *rounding property* that allows these remaining clauses to be assigned a solution by rounding a solution of the LP relaxation in a prespecified way; in the case of Horn clauses, by always rounding down.

A class $\mathcal{C}$ of satisfiability problems is closed under *deletion* and *contraction* if, given a clause set $S \in \mathcal{C}$, S remains in $\mathcal{C}$ after (a) any clause is deleted from S and (b) any given literal is removed from every clause of S in which it occurs. Since unit resolution operates by deletion and contraction, it preserves the structure of any class that is closed under

these operations. This is true of Horn sets in particular because removing literals does not increase the number of positive literals in a clause.

Horn clauses can be solved by rounding down because they have an integral least element property. An element $v \in P \subset R^n$ is a *least element* of P if $v \leq x$ for every $x \in P$. It is easy to see that if S is a satisfiable set of Horn clauses, S^{LP} defines a polyhedron that always contains an integral least element. This element is identified by fixing variables as determined by unit resolution and setting all remaining variables to zero. Thus if S is the Horn set that remains after applying unit resolution to a satisfiable Horn set, we can obtain a satisfying solution for S by rounding down any feasible solution of S^{LP}.

Cottle and Veinott [23] state a sufficient condition under which polyhedra in general have a least element.

Proposition 9 *A nonempty polyhedron $P = \{x \mid Ax \geq b,\ x \geq 0\}$ has a least element if each row of A has at most one positive component. There is an integer least element if every positive element of A is 1.*

Proof. If $b \leq 0$ then $x = 0$ is a least element. Otherwise let b_i be the largest positive component of b. Since P is nonempty, row i of A has exactly one positive component A_{ij}. The ith inequality of $Ax \geq b$ can be written

$$x_j \geq \frac{1}{A_{ij}} \left(b_i - \sum_{k \neq j} A_{ik} x_k \right)$$

Since $A_{ik} \leq 0$ for $k \neq j$ and each $x_k \geq 0$, we have the positive lower bound $x_j \geq b_i / A_{ij}$. Thus we can construct a lower bound $\underline{x}$ for x by setting $\underline{x}_j = b_i / A_{ij}$ and $\underline{x}_k = 0$ for $k \neq j$. If we define $\tilde{x} = x - \underline{x}$, we can translate polyhedron P to

$$\tilde{P} = \left\{ \tilde{x} \;\middle|\; A\tilde{x} \geq \tilde{b} = (b - A\underline{x}),\ \tilde{x} \geq 0 \right\}$$

We repeat the process and raise the lower bound $\underline{x}$ until $\tilde{b} \leq 0$. At this point $\underline{x}$ is a least element of P. Clearly $\underline{x}$ is integer if each $A_{ij} = 1$.

Since the inequality set S^{01} for a Horn problem S satisfies the conditions of Proposition 9, Horn problems have the integral least element property.

3.2 Extended Horn Problems

The key to extending the concept of a Horn problem is to find a larger problem class that has the rounding property and is closed under deletion and contraction. Some sets with

the rounding property can be identified through a result of Chandrasekaran [13], which relies on Cottle and Veinott's least element theorem.

Proposition 10 *Let $Ax \geq b$ be a linear system with integral components, where A is an $m \times n$ matrix. Let T be a nonsingular $n \times m$ matrix that satisfies the following conditions:*

(i) T and T^{-1} are integral.

(ii) Each row of T^{-1} contains at most one negative entry, and any such entry is -1.

(iii) Each row of AT^{-1} contains at most one negative entry, and any such entry is -1.

Then if x solves $Ax \geq b$, so does $T^{-1}\lceil Tx \rceil$.

The matrix T in effect gives instructions for how a solution of the LP can be rounded.

Proof. We rely on an immediate corollary of Proposition 9: a polyhedron of the form $P = \{x \mid Ax \geq b,\ x \leq a\}$ has an integral largest element if A, b and a are integral and each row of A has at most one negative entry, namely -1.

Now if $\hat{y}$ solves $AT^{-1}y \geq b$, the polyhedron $\hat{P} = \{y \mid AT^{-1}y \geq b,\ y \leq \hat{y}\}$ has an integral largest element, and $\lceil \hat{y} \rceil$ is therefore in $\hat{P}$. This shows that

$$AT^{-1}y \geq b \text{ implies } AT^{-1}\lceil y \rceil \geq b$$

Setting $x = T^{-1}y$ we have

$$Ax \geq b \text{ implies } AT^{-1}\lceil Tx \rceil \geq b \tag{22}$$

Similarly, if $\tilde{y}$ satisfies $T^{-1}y \geq 0$, the polyhedron $\tilde{P} = \{y \mid T^{-1}y \geq 0,\ y \leq \lceil \tilde{y} \rceil\}$ has an integral largest element and $\lceil \tilde{y} \rceil$ is in $\tilde{P}$. So

$$T^{-1}y \geq 0 \text{ implies } T^{-1}\lceil y \rceil \geq 0$$

and setting $x = T^{-1}y$ we have

$$x \geq 0 \text{ implies } T^{-1}\lceil Tx \rceil \geq 0 \tag{23}$$

Implications (22) and (23) together prove the proposition.

To apply Proposition 10 to a clause set S, we note that S^{LP} has the form $Hx \geq h, 0 \leq x \leq e$, where e is a vector of ones. This is an instance of the system $Ax \geq b, x \geq 0$ in Proposition 10 when

$$A = \begin{bmatrix} H \\ -I \end{bmatrix}, b = \begin{bmatrix} h \\ -e \end{bmatrix}$$

From condition (ii) of Proposition 10, each row of T^{-1} contains at most one negative entry (namely, -1), and from (iii) the same is true of $-T^{-1}$. Thus we have:

(i′) T^{-1} is a nonsingular $n \times n$ matrix.

(ii′) Each row of T^{-1} contains exactly two nonzero entries, namely 1 and -1.

(iii′) Each row of HT^{-1} contains at most one negative entry, namely -1.

Condition (ii′) implies that T^{-1} is the edge-vertex incidence matrix of a directed graph. Since T^{-1} is nonsingular, it is the edge-vertex incidence matrix of a directed tree $\mathcal{T}$ on $n+1$ vertices. For an appropriate ordering of the vertices, T^{-1} is lower triangular. The inverse T is the vertex-path incidence matrix of $\mathcal{T}$.

Example 12 *Figure 2 shows the directed tree $\mathcal{T}$ corresponding to the matrices T^{-1}, T below.*

$$T^{-1} = \begin{array}{c} \begin{array}{ccccccc} A & B & C & D & E & F & G \end{array} \\ \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & -1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & -1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 & -1 \end{bmatrix} \end{array} \begin{array}{c} R \\ 1 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ 0 \end{array} \qquad T = \begin{array}{c} A \\ B \\ C \\ D \\ E \\ F \\ G \end{array} \begin{bmatrix} -1 & 0 & 0 & 0 & 0 & 0 & 0 \\ -1 & -1 & 0 & 0 & 0 & 0 & 0 \\ -1 & 0 & -1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & -1 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & 0 \\ 0 & 0 & 0 & -1 & 0 & -1 & -1 \end{bmatrix}$$

The column corresponding to vertex R, the root of $\mathcal{T}$, is shown to the right of T^{-1}. In this example all the arcs are directed away from the root, but this need not be so. An arc directed toward the root would result in reversed signs in the corresponding row of T^{-1} and column of T.

The entries in each row of the propositional matrix H can be interpreted as flows on the edges of $\mathcal{T}$. Thus each variable x_j is associated with an edge in $\mathcal{T}$. Condition (iii′) has the effect of requiring that at most one vertex (other than the root) be a net receiver of flow. To see what this means graphically, let an *extended star* be a rooted tree consisting of one or more arc-disjoint chains extending out from the root. Then (iii′) implies that any row of H has the *extended star-chain property*: it describes flows that can be partitioned into a set of unit flows into the root on some (possibly empty) extended star subtree of $\mathcal{T}$ and a unit flow on one (possibly empty) chain in $\mathcal{T}$.

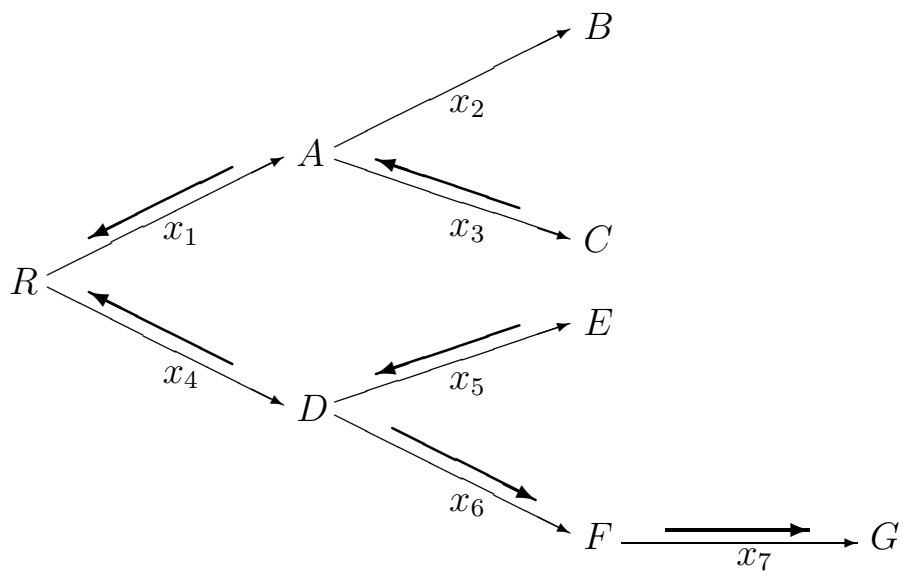

Figure 2: *Extended star-chain flow pattern corresponding to an extended Horn clause.*

Example 13 *Suppose* $H_i = [\ -1 \quad 0 \quad -1 \quad -1 \quad -1 \quad 1 \quad 1\ ]$ *is a row of* H*, corresponding to the clause* $\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5 \vee x_6 \vee x_7$*. It defines the flow depicted by arrows in Fig. 2. Note that a* -1 *in* H_i *indicates flow against the direction of the edge. The extended star consists of the flows* $C \to A \to R$ *and* $D \to R$*, while the chain consists of* $E \to D \to F \to G$*. (Flow* $E \to D$ *could also be regarded as part of the extended star.) In this case* $H_i T^{-1} = [\ 0 \quad 0 \quad 1 \quad 1 \quad 1 \quad 0 \quad -1\ ]$*, which satisfies (iii′).*

A clause set S with the 0-1 formulation $Hx \geq h$ is *renamable extended Horn* if it can be associated with a directed tree $\mathcal{T}$ in which each row of H has the extended star-chain property. S is *extended Horn* if each edge of $\mathcal{T}$ is directed away from the root. If each row of H describes flows into the root on a *star* subtree of $\mathcal{T}$ (i.e., an extended star whose chains have length 1), then H is renamable Horn. Extended Horn is therefore a substantial generalization of Horn. Clause set (9), for example, is not renamable Horn but is renamable extended Horn. This can be seen by associating x_1, x_2, x_3 respectively with arcs $(R, A), (A, B)$ and (B, C) on a tree with arcs directed away from the root R.

We can now see why an extended Horn problem can be solved by unit resolution. The extended star-chain structure is clearly preserved by deletion and contraction. Thus if unit resolution does not detect unsatisfiability, the remaining clauses have the extended star-chain property and contain at least two literals each. Their LP relaxation can be solved by

setting all unfixed variables to 1/2, and Chandrasekaran's theorem gives instructions for rounding this solution to obtain an 0-1 solution. Let an edge e of $\mathcal{T}$ be *even* if the number of edges on the path from the root to the closer vertex of e is even, and *odd* otherwise. It is shown in [15] that variables corresponding to even edges are rounded down, and the rest rounded up.

Proposition 11 *Let S be a satisfiable extended Horn set S associated with directed tree $\mathcal{T}$. Apply unit resolution to S and let $\mathcal{T}'$ be the tree that results from contracting the edges of $\mathcal{T}$ that correspond to fixed variables. Then a satisfying solution for S can be found by assigning false to unfixed variables that correspond to even edges of $\mathcal{T}'$, and true to those corresponding to odd edges.*

The result is valid for renamable extended Horn sets if the values of renamed variables are complemented. In the special case of a Horn set, one always assigns false to unfixed variables, since all edges are adjacent to the root and therefore even. The following is a corollary.

Proposition 12 *A renamable extended Horn clause set S is satisfiable if and only if it has no unit refutation, and if and only if S^{LP} is feasible.*

Example 14 *Consider an extended Horn set S consisting of the single clause $\bar{x}_1 \vee \bar{x}_3 \vee \bar{x}_4 \vee \bar{x}_5 \vee x_6 \vee x_7$, discussed in Example 13. Unit resolution has no effect, and S^{LP} has the solution $\hat{x} = (1/2, 0, 1/2, 1/2, 1/2, 1/2, 1/2)$. Thus we obtain a satisfying solution for S when we set $x = T^{-1}\lceil T\hat{x}\rceil = T^{-1}\lceil(-1/2, -1, -1, -1/2, -1, -1, -3/2)\rceil = T^{-1}(0, -1, -1, 0, -1, -1, -1) = (0, 1, 1, 0, 1, 1, 0)$. Note that we round down on even edges and up on odd edges.*

Interestingly, once a numerical interpretation has pointed the way to an extension of the Horn concept, a slightly more general extension becomes evident. Note that the extended star-chain property is preserved by contraction of edges in part because every edge in the extended star is connected by a path to the root. The same is true if the paths to the root are not disjoint. As Schlipf et al. [69] point out, we can therefore generalize the extended star-chain property as follows: a clause has the *arborescence-chain property* with respect to $\mathcal{T}$ when it describes flows that can be partitioned into a set of unit flows into the root on some (possibly empty) subtree of $\mathcal{T}$ and a unit flow on one (possibly empty) chain in $\mathcal{T}$. A clause having this structure is clearly satisfied by the same even-odd truth assignment as before. We can therefore further extend the concept of renamable extended Horn problems as those whose clauses define flows having the arborescence/chain structure on some corresponding directed tree $\mathcal{T}$.

Example 15 *Suppose the clause in Example 14 contains an additional literal* $\bar{x}_2$*. The corresponding flow pattern is that shown in Fig. 2 with an additional flow on arc* (A, B) *directed toward the root. Thus literals* $\bar{x}_1, \bar{x}_2, \bar{x}_3, \bar{x}_4$ *correspond to an arborescence, and* x_5, x_6, x_7 *to a chain. We conclude that the clause set* S *consisting solely of* $\bar{x}_1 \vee \bar{x}_2 \vee \bar{x}_3 \vee \bar{x}_4 \vee x_6 \vee x_7$ *is extended Horn in the expanded sense.*

From here out we understand extended Horn and renamable extended Horn problems in the expanded sense of Schlipf et al. Propositions 11 and 12 continue to hold.

3.3 One-Step Lookahead

Renamable Horn sets have the double advantage that (a) their satisfiability can be checked by unit resolution, and (b) a satisfying solution can be easily identified when it exists. We have (b) because there are linear-time algorithms for checking whether a clause set is renamable Horn and, if so, how to rename variables to make it Horn [1, 14, 56]. Once the renaming scheme is identified, variables that are unfixed by unit resolution can simply be set to false, or to true if they are renamed.

Renamable extended Horn sets do not have this double advantage. Although unit resolution can check for satisfiability, a satisfying solution is evident only when the associated directed tree $\mathcal{T}$ is given. There is no known polynomial-time algorithm for finding $\mathcal{T}$, even in the case of an unrenamed extended Horn problem. Swaminathan and Wagner [71] have shown how to identify $\mathcal{T}$ for a large subclass of extended Horn problems, using a graph realization algorithm that runs in slightly more than linear time. Yet their approach has not been generalized to full extended Horn sets.

Fortunately, as Schlipf et al. point out [69], a simple *one-step lookahead* algorithm can solve a renamable extended Horn problem without knowledge of $\mathcal{T}$. Let a class $\mathcal{C}$ of satisfiability problems have the *unit resolution property* if (a) a clause set in $\mathcal{C}$ is satisfiable if and only if there is no unit refutation for it, and (b) $\mathcal{C}$ is closed under deletion and contraction.

Proposition 13 *If a class of Boolean satisfiability problems has the unit resolution property, a one-step lookahead algorithm can check any clause set in the class for satisfiability and exhibit a satisfying solution if one exists.*

Since renamable extended Horn problems have the unit resolution property, a one-step lookahead algorithm solves their satisfiability problem.

One-step lookahead is applied to a clause set S as follows. Let S_0 be the result of applying unit resolution to S. If S_0 contains the empty clause, stop, because S is unsatisfiable. If S_0 is empty, stop, because the variables fixed by unit resolution already satisfy S. Otherwise perform the following:

1. Let $S_1 = S_0 \cup \{x_j\}$ for some variable x_j occurring in S_0 and apply unit resolution to S_1. If S_1 is empty, fix x_j to true and stop. If S_1 is nonempty and there is no unit refutation, fix x_j to true, let $S_0 = S_1$, and repeat this step.

2. Let $S_1 = S_0 \cup \{\bar{x}_j\}$ and apply unit resolution to S_1. If S_1 is empty, fix x_j to false and stop. If S_1 is nonempty and there is no unit refutation, fix x_j to false, let $S_0 = S_1$, and return to step 1.

3. Stop without determining whether S is satisfiable.

The algorithm runs in time proportional to nL, where n is the number of variables and L the number of literals in S.

Proof of Proposition 13. Since the one-step lookahead algorithm is clearly correct, it suffices to show that it cannot reach step 3 when S belongs to a problem class with the unit resolution property. Step 3 can be reached only if there is no unit refutation for S_0, but there is a unit refutation for both $S_0 \cup \{x_j\}$ and $S_0 \cup \{\bar{x}_j\}$, which means S_0 is unsatisfiable. Yet S_0 cannot be unsatisfiable because it has no unit refutation and was obtained by unit resolution from a problem S in a class with the unit resolution property.

Example 16 *Consider the clause set S below:*

$$\begin{array}{l} \bar{x}_1 \vee x_2 \\ \bar{x}_1 \vee \bar{x}_2 \\ x_1 \vee x_2 \vee x_3 \\ x_1 \vee x_2 \vee \bar{x}_3 \end{array}$$

S is not renamable Horn but is renamable extended Horn, as can be seen by renaming x_1, x_2 and associating x_1, x_2, x_3 respectively with $(A, B), (R, A), (B, C)$ in a directed tree. However, without knowledge of the tree we can solve the problem with one-step lookahead. Unit resolution has no effect on $S_0 = S$, and we let $S_1 = S_0 \cup \{x_1\}$ in step 1. Unit resolution derives the empty clause from S_1, and we move to step 2 by setting $S_1 = S_0 \cup \{\bar{x}_1\}$. Unit resolution reduces S_1 to $\{x_2 \vee x_3, x_2 \vee \bar{x}_3\}$, and we return to step 1 with $S_0 = \{x_2 \vee x_3, x_2 \vee \bar{x}_3\}$. Setting $S_1 = S_0 \cup \{x_2\}$, unit resolution reduces S_1 to the empty set, and the algorithm terminates with $(x_1, x_2) = (0, 1)$, where x_3 can be set to either value.

At this writing no algorithms or heuristic methods have been proposed for finding a set of variables that, when fixed, leave a renamable extended Horn problem. The same deficiency exists for the integrality classes discussed below.

3.4 Balanced Problems and Integrality

We have identified a class of satisfiability problems that can be quickly solved by a one-step lookahead algorithm that uses unit resolution. We now consider problems that can be solved by linear programming because their LP relaxation describes an integral polytope. Such problems are doubly attractive because unit resolution can solve them even without one-step lookahead, due to the following fact [18].

Let us say that a clause set S is *ideal* if S^{LP} defines an integral polyhedron. If S^{01} is the system $Ax \geq b$, we say that A is an ideal matrix if S is ideal.

Proposition 14 *Let S be an ideal clause set that contains no unit clauses. Then for any variable x_j, S has a solution in which x_j is true and a solution in which x_j is false.*

Proof. Since every clause contains at least two literals, setting each $x_j = 1/2$ satisfies S^{LP}. This solution is a convex combination of the vertices of the polyhedron described by S^{LP}, which by hypothesis are integral. Thus for every j there is a vertex v in the convex combination at which $v_j = 0$ and a vertex at which $v_j = 1$.

To solve an ideal problem S, first apply unit resolution to S to eliminate all unit clauses. The remaining set S is ideal. Pick any atom x_j that occurs in S and add the unit clause x_j or $\bar{x}_j$ to S, arbitrarily. By Proposition 14, S remains satisfiable if it was satisfiable to being with. Repeat the procedure until a unit refutation is obtained or until S is empty (and unit resolution has fixed the variables to a satisfying solution). The algorithm runs in linear time.

One known class of ideal satisfiability problems are *balanced* problems. If S^{01} is the 0-1 system $Ax \geq b$, then S is balanced when A is balanced. A $0, \pm 1$ matrix A is balanced when every square submatrix of A with exactly two nonzeros in each row and column has the property that its entries sum to a multiple of four. The following is proved by Conforti and Cornuéjols [18]:

Proposition 15 *Clause set S is ideal if it is balanced.*

Related results are surveyed in [20, 22]. For instance, balancedness can be checked by examining subsets of S for bicolorability. Let a $\{0, \pm 1\}$-matrix be *bicolorable* if its rows can be partitioned into blue rows and red rows such that every column with two or more nonzeros either contains two entries of the same sign in rows of different colors or two entries of different signs in rows of the same color. A clause set S is bicolorable if the coefficient matrix of S^{01} is bicolorable. Conforti and Cornuéjols [19] prove the following:

Proposition 16 *Clause set S is balanced if and only if every subset of S is bicolorable.*

Example 17 *Consider the following clause set S:*

$$\begin{array}{llll} x_1 & \vee x_3 & & \\ \bar{x}_1 & & & \vee x_4 \\ & x_2 \vee x_3 & & \\ & \bar{x}_2 \vee & & \vee x_4 \end{array} \tag{24}$$

By coloring the first two clauses red and the last two blue, we see from Proposition 16 that all subsets of S are balanced, and by Proposition 15, S is ideal. We can also use Proposition 14 to solve S. Initially, unit resolution has no effect, and we arbitrarily set $x_1 = 0$, which yields the single clause $\bar{x}_2 \vee x_4$ after unit resolution. Now we arbitrarily set $x_2 = 0$, whereupon S becomes empty. The resulting solution is $(x_1, x_2, x_3) = (0, 0, 1)$ with either value for x_4.

3.5 Resolution and Integrality

Since resolvents are cutting planes, one might ask whether applying the resolution algorithm to a clause set S cuts off enough of the polyhedron defined by S^{LP} to produce an integral polyhedron. The answer is that it does so if and only if the monotone subproblems of S already define integral polyhedra.

To make this precise, let a *monotone subproblem* of S be a subset $\hat{S} \subset S$ in which no variable occurs in both a positive and a negative literal. A monotone subproblem is *maximal* if it is a proper subset of no monotone subproblem. We can suppose without loss of generality that all the literals in a given monotone subproblem $\hat{S}$ are positive (by complementing variables as needed). Thus $\hat{S}^{01}$ is a set covering problem, which is a 0-1 problem of the form $Ax \geq e$ in which A is a 0-1 matrix and e a vector of ones. $\hat{S}$ can also be viewed as a satisfiability problem with all positive literals (so that the same definition of an ideal problem applies). The following result, proved in [39], reduces the integrality question for satisfiability problems to that for set covering problems:

Proposition 17 *If S contains all of its prime implicates, then S is ideal if and only if every maximal monotone subproblem $\hat{S} \subset S$ is ideal.*

One can determine whether S contains all of its prime implicates by checking whether the resolution algorithm adds any clauses to S; that is, by checking whether any pair of clauses in S have a resolvent that is not already absorbed by a clause in S.

Guenin [32] pointed out an alternate statement of Proposition 17. Given a clause set S, let A be the coefficient matrix in S^{01}, and define

$$D_S = \begin{bmatrix} P & N \\ I & I \end{bmatrix}$$

where 0-1 matrices P, N are the same size as A and indicate the signs of entries in A. That is, $P_{ij} = 1$ if and only if $A_{ij} = 1$ and $N_{ij} = 1$ if and only if $A_{ij} = -1$.

Proposition 18 *If S contains all of its prime implicates, then S is ideal if and only if D_S is ideal.*

Example 18 *Consider the clause set S below:*

$$\begin{array}{l} x_1 \vee x_2 \vee x_3 \\ x_1 \vee \bar{x}_2 \vee \bar{x}_3 \end{array}$$

Since S is not balanced, Proposition 15 does not show that S is ideal. However, Proposition 17 applies because S consists of its prime implicates, and its two maximal monotone subproblems (i.e., the two individual clauses) obviously define integral polyhedra. S is therefore ideal. Proposition 18 can also be applied if one can verify that the following matrix D_S is ideal:

$$\begin{bmatrix} 1 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 1 & 1 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

D_S is not balanced, since there is no feasible coloring for the submatrix consisting of rows 1,2 and 4. D_S is ideal but must be shown to be so in some fashion other than bicolorability.

Nobili and Sassano [60] strengthened Propositions 17 and 18 by pointing out that a restricted form of resolution suffices. Let a *disjoint* resolvent be the resolvent of two clauses that have no variables in common, except the one variable that appears positively in one clause and negatively in the other. The disjoint resolution algorithm is the same as the ordinary resolution algorithm except that only disjoint resolvents are generated.

Proposition 19 *Let S' be the result of applying the disjoint resolution algorithm to S. Then S is ideal if and only if $D_{S'}$ is ideal.*

Example 19 *Consider the follow clause set S:*

$$\begin{array}{l} x_1 \vee x_2 \\ \bar{x}_1 \qquad \vee x_3 \\ x_1 \vee \bar{x}_2 \vee x_3 \end{array}$$

Disjoint resolution generates one additional clause, so that S' consists of the above and

$$x_2 \vee x_3$$

Further resolutions are possible, but they need not be carried out because the resolvents are not disjoint. The matrix $D_{S'}$ is

$$\begin{bmatrix} 1 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 \\ 0 & 1 & 0 & 0 & 1 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 \end{bmatrix}$$

$D_{S'}$ is not balanced, since the submatrix consisting of rows 1, 3 and 5 is not bicolorable. In fact $D_{S'}$ is not ideal because the corresponding set covering problem $D_{S'}y \geq e$ defines a polyhedron with the fractional vertex $y = (1/2, 1, 1/2, 1/2, 0, 1/2)$. Therefore S is not ideal, as can be verified by noting that S^{LP} defines a polyhedron with the fractional vertex $x = (1/2, 0, 1/2)$.

One can apply Proposition 17 by carrying out the full resolution algorithm on S, which yields $\hat{S} = \{x_1 \vee x_2, x_3\}$. $\hat{S}$ itself is its only maximum monotone subproblem, which means $\hat{S}$ is ideal because it is obviously balanced.

4 Computational Considerations

Computational testing of integer programming methods for satisfiability dates back at least to a 1988 paper of Blair, Jeroslow and Lowe [8]. A 1994 experimental study [35] suggested that, at that time, an integer programming approach was roughly competitive with a pure branching strategy that uses no LP relaxations, such as a Davis-Putnam-Loveland (DPL) method [24, 55]. DPL methods have since been the subject of intense study and have improved dramatically (see [30, 31, 81] for surveys). The Chaff system of Moskewicz et al. [58], for example, solves certain industrial satisfiability problems with a million variables or more. It is well known, however, that satisfiability problems of a given size can vary enormously in difficulty (e.g., [21]).

Integer programming methods for the boolean satisfiability problem have received relatively little attention since 1994, despite substantial improvements in the LP solvers on which they rely. One might argue that the integer programming approach is impractical

for large satisfiability problems, since it requires considerable memory and incurs substantial overhead in the solution of the LP relaxation. This is not the case, however, for the decomposition methods discussed above. They apply integer programming only to a small "core" problem and leave the easier part of the problem to be processed by unit resolution or some other fast method.

The most straightforward form of decomposition, discussed in Section 2.4, is to branch on a few variables that are known to yield renamable Horn problems at the leaf nodes of a shallow tree. (One might also branch so as to produce renamable extended Horn problems at the leaf nodes, but this awaits a practical heuristic method for isolating extended Horn substructure.) The branching portion of the algorithm could be either a branch-and-cut method or a DPL search. In the latter case, decomposition reduces to a DPL method with a sophisticated variable selection rule.

Benders decomposition also takes advantage of renamable Horn substructure (Section 2.5). A branch-and-cut method can be applied to the Benders master problem, which is a general 0-1 problem, while unit resolution solves the renamable Horn subproblems. (The subproblems might also be renamable extended Horn, which would require research into how Benders cuts can be rapidly generated.) Yan and Hooker [80] tested a specialized a Benders approach on circuit verification problems, an important class of satisfiability problems. They found it to be faster than binary decision diagrams at detecting errors, although generally slower at proving correctness. A related Benders method has been applied to machine scheduling problems [40, 41, 42, 43, 47], resulting in computational speedups of several orders of magnitude relative to state-of-the-art constraint programming and mixed integer programming solvers. However, a Benders method has apparently not been tested on satisfiability problems other than circuit verification problems.

A third form of decomposition replaces LP relaxation with Lagrangean relaxation, again in such a way as to isolate special structure (Section 2.6). A full Lagrangean method would solve the Lagrangean dual at some nodes of the search tree, but this strategy has not been tested computationally. The Lagrangean-based method of Bennaceur et al. [7], however, uses a fixed set of Lagrange multipliers and appears to be significantly more effective than DPL. Since such advanced methods as Chaff are highly engineered DPL algorithms, a Lagrangean approach may have the potential to outperform the best methods currently in use.

The integrality results (Sections 3.4, 3.5) remain primarily of theoretical interest, since it is hard to recognize or isolate special structure that ensures integrality. Nonetheless they could become important in applications that involve optimization of an objective function subject to logical clauses, since they indicate when an LP solver may be used rather than a general integer solver. One such application is the maximum satisfiability problem [33, 34].

The ideas presented here are part of a more general strategy of merging logical inference

with optimization methods. There is growing evidence [36, 40, 44, 54, 57, 77] that logic-based methods can enhance optimization, and similar evidence that optimization methods can assist logical inference—not only in boolean logic, but in probabilistic, nonmonotonic, and belief logics as well [17]. The guiding principle in each case is to employ concepts, whether they be logical or numerical, that best reveal the structure of the problem at hand.

5 Exercises

1. Show that the resolvent $x_1 \vee x_4$ obtained in Example 7 corresponds to a rank 1 cut.

2. Formulate the problem of checking whether a clause set is extended Horn as a 2-SAT problem (i.e., a satisfiability problem in CNF with at most two literals per clause). The most straightforward 2-SAT formulation is quadratic in size, but there are linear-size formulations (e.g., [1]).

3. Show that disjoint resolution adds nothing to the inferential power of unit resolution. That is, unit resolution shows a clause set S to be unsatisfiable if and only if a combination of unit and disjoint resolution shows S to be unsatisfiable. Hint: note that a disjoint resolvent is the sum of its parents.

4. Let T be the set of satisfying solutions for a given Horn clause set. Show that if $x^1, x^2 \in T$, then $\min\{x^1, x^2\} \in T$, where the minimum is componentwise. This property of Horn sets is generalized in [49, 68].

5. Exhibit a Horn clause set S for which S^{LP} has a nonintegral extreme point.

6. Suppose that S contains all of its prime implicates. Show that a one-step lookahead algorithm solves the satisfiability problem for S.

7. Exhibit a two-variable satisfiability problem that is not renamable extended Horn.

8. One can show that extended Horn sets have the unit resolution property, based solely on the arborescence-chain structure and without reference to the numerical interpretation. Construct such an argument.

9. Suppose that extended Horn sets were defined so that the arborescence-chain property permitted as many as two chains rather than just one. Show that extended Horn sets, under this definition, would not have the unit resolution property, by exhibiting a counterexample. Where does the argument of the previous question break down?

References

[1] Aspvall, B., Recognizing disguised NR(1) instance of the satisfiability problem, *Journal of Algorithms* **1** (1980) 97–103.

[2] Balas, E., and W. Niehaus, Optimized crossover-based genetic algorithms for the maximum cardinality and maximum weight clique problems, *Journal of Heuristics* **4** (1998) 107–122.

[3] Balas, E., and J. Xue, Minimum weighted coloring of triangulated graphs with application to maximum weight vertex packing and clique finding in arbitrary graphs, *SIAM Journal on Computing* **20** (1991) 209–221.

[4] Balas, E., and C. S. Yu, Finding a maximum clique in an arbitrary graph, *SIAM Journal on Computing* **15** (1986) 1054–1068.

[5] Battiti, R., and M. Protasi, Reactive local search for the maximum clique problem, *Algorithmica* **29** (2001) 610–637.

[6] Benders, J. F. 1962. Partitioning procedures for solving mixed-variables programming problems, *Numerische Mathematik* **4** 238–252.

[7] Bennaceur, H., I. Gouachi and G. Plateau, An incremental branch-and-bound method for the satisfiability problem, *INFORMS Journal on Computing* **10** (1998) 301-308.

[8] Blair, C., R. G. Jeroslow, and J. K. Lowe, Some results and experiments in programming techniques for propositional logic, *Computers and Operations Research* **13** (1988) 633–645.

[9] Bomze, I. M, M. Budinich, M. Pellilo, and C. Rossi, Annealed replication: A new heuristic for the maximum clique problem, *Discrete Applied Mathematics* **121** (2002) 27–49.

[10] Boole, G. *The Mathematical Analysis of Logic: Being a Essay Toward a Calculus of Deductive Reasoning*, Blackwell (Oxford, 1951), original work published 1847.

[11] Busygin, S., S. Butenko, P. M. Pardalos, A heuristic for the maximum independent set problem based on optimization of a quadratic over a sphere, *Journal of Combinatorial Optimization* **6** (2002) 287–297.

[12] Carraghan, R., and P. Pardalos, An exact algorithm for the maximum clique problem, *Operations Research Letters* **9** (1990) 375–382.

[13] Chandrasekaran, R., Integer programming problems for which a simple rounding type of algorithm works, in W. R. Pulleyblank, *Progress in Combinatorial Optimization*, Academic Press Canada (1984) 101–106.

[14] Chandru, V., C. R. Coullard, P. L. Hammer, M. Montañez, and X. Sun, Horn, renamable Horn and generalized Horn functions, *Annals of Mathematics and Artificial Intelligence* **1** (1990) 333–347.

[15] Chandru, V., and J. N. Hooker, Extended Horn clauses in propositional logic, *Journal of the ACM* **38** (1991) 203–221.

[16] Chandru, V., and J. N. Hooker, Detecting embedded Horn structure in propositional logic, *Information Processing Letters* **42** (1992) 109–111.

[17] Chandru, V., and J. N. Hooker, *Optimization Methods for Logical Inference*, John Wiley & Sons (New York, 1999).

[18] Conforti, M., and G. Cornuéjols, A class of logic programs solvable by linear programming, *Journal of the ACM* **42** (1995) 1107–1113.

[19] Conforti, M., and G. Cornuéjols, Balanced 0,±1 matrices, bicoloring and total dual integrality, *Mathematical Programming* **71** (1995) 249–258.

[20] Conforti, M., G. Cornuéjols, A. Kapoor, and K. Vušković, Perfect, ideal and balanced matrices, *European Journal of Operational Research* **133** (2001) 455–461.

[21] Cook, S. A., and D. G. Mitchell, Finding hard instances of the satisfiability problem: A survey, in: D. Du, J. Gu, P. M. Pardalos, *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **35**, American Mathematical Society (1997) 1–17.

[22] Cornuéjols, G., and B. Guenin, Ideal clutters, *Discrete Applied Mathematics* **123** (2002) 303–338.

[23] Cottle, R. W., and A. F. Veinott, Polyhedral sets having a least element, *Mathematical Programming* **3** (1972) 238–249.

[24] Davis, M., and H. Putnam, A computing procedure for quantification theory, *Journal of the ACM* **7** (1960) 201–215.

[25] Dilkina, B., C. Gomes, and A. Sabharwal, Tradeoffs in the complexity of backdoor detection, in C. Bessiere, ed., Principles and Practice of Constraint Programming (CP 2007), LNCS 4741, 256–270.

[26] Feo, T. A., M. G. C. Resende, S. H. Smith, A greedy randomized adaptive search procedure for maximum independent set, *Operations Research* **42** (1994) 860–878.

[27] Funabiki, N., Y. Takefuji and Kuo-Chun Lee, A neural network model for finding a near-maximum clique, *Journal of Parallel and Distributed Computing* **14** (1992) 340–344.

[28] Gendreau, M., P. Soriano, and L. Salvail, Solving the maximum clique problem using a tabu search approach, *Annals of Operations Research* **41** (1993) 385–403.

[29] Geoffrion, A. M., Generalized Benders decomposition, *Journal of Optimization Theory and Applications* **10** (1972) 237–260.

[30] Giunchiglia, E., M. Maratea, A. Tacchella, and D. Zambonin, Evaluating search heuristics and optimization techniques in propositional satisfiability, R. Gore, A. Leitsch, and T. Nipkow, *Automated Reasoning: First International Joint Conference* (IJCAR2001), *Lecture Notes in Artificial Intelligence* **2083**, Springer (2001) 347–363.

[31] Gu, Jun, P. W. Purdom, J. Franco, and B. W. Wah, Algorithms for the satisfiability (SAT) problem: A survey, in D. Du, J. Gu, P. M. Pardalos, *Satisfiability Problem: Theory and Applications*, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **35**, American Mathematical Society (1997) 19–51.

[32] Guenin, B., Perfect and ideal 0, ± 1 matrices, *Mathematics of Operations Research* **23** (1998) 322–338.

[33] Hansen, P., and B. Jaumard, Algorithms for the maximum satisfiability problem, *Computing* **44** (1990) 279–303.

[34] Hansen, P., B. Jaumard, and M. P. De Aragao, Mixed-integer column generation algorithms and the probabilistic maximum satisfiability problem, *European Journal of Operational Research* **108** (1998) 671–683.

[35] Harche, F., J. N. Hooker, and G. Thompson, A computational study of satisfiability algorithms for propositional logic, *ORSA Journal on Computing* **6** (1994) 423–435.

[36] Heipcke, S., *Combined Modelling and Problem Solving in Mathematical Programming and Constraint Programming*, PhD thesis, University of Buckingham (1999).

[37] Hifi, M., A genetic algorithm-based heuristic for solving the weighted maximum independent set and some equivalent problems, *Journal of the Operational Research Society* **48** (1997) 612–622.

[38] Hooker, J. N., Input proofs and rank one cutting planes, *ORSA Journal on Computing* **1** (1989) 137–145.

[39] Hooker, J. N., Resolution and the integrality of satisfiability polytopes, *Mathematical Programming* **4** (1996) 1–10.

[40] Hooker, J. N., *Logic-Based Methods for Optimization: Combining Optimization and Constraint Satisfaction*, John Wiley & Sons (New York, 2000).

[41] Hooker, J. N., A hybrid method for planning and scheduling, *Constraints* **10** (2005) 385–401.

[42] Hooker, J. N., An integrated method for planning and scheduling to minimize tardiness, *Constraints* **11** (2006) 139–157.

[43] Hooker, J. N., Planning and scheduling by logic-based Benders decomposition, *Operations Research* **55** (2007) 588–602.

[44] Hooker, J. N., *Integrated Methods for Optimization*, Springer (2007).

[45] Hooker, J. N., and C. Fedjki, Branch-and-cut solution of inference problems in propositional logic, *Annals of Mathematics and Artificial Intelligence* **1** (1990) 123–140.

[46] Hooker, J. N. and G. Ottosson, Logic-based Benders decomposition, *Mathematical Programming* **96** (2003) 33–60.

[47] Jain, V., and I. E. Grossmann, Algorithms for hybrid MILP/CP models for a class of optimization problems, *INFORMS Journal on Computing* **13** (2001) 258–276.

[48] Jagota, A., and L. A. Sanchis, Adaptive, restart, randomized greedy heuristics for maximum clique, *Journal of Heuristics* **7** (2001) 565–585.

[49] Jeavons, P., D. Cohen, and M. Gyssens, A test for tractability in E. C. Freuder, *Principles and Practice of Constraint Programming* (CP96), *Lecture Notes in Computer Science* **1118**, Springer (1996) 267–281.

[50] Jeroslow, R. E., and J. Wang, Solving propositional satisfiability problems, *Annals of Mathematics and Artificial Intelligence* **1** (1990) 167–188.

[51] Kilby, P., J. K. Slaney, S. Thibaux, and T. Walsh, Backbones and backdoors in satisfiability, *AAAI Proceedings* (2005) 1368-1373.

[52] Kottler, S., M. Kaufmann, and C. Sinz, Computation of renameable Horn backdoors, *Theory and Applications of Satisfiability Testing, Eleventh International Conference* (SAT 2008), *Lecture Notes in Computer Science* **4996** (2008) 154–160.

[53] Lau, H. Y., and H. F. Ting, The greedier the better: An efficient algorithm for approximating maximum independent set, in T. Asano, H. Imai, D. T. Lee, S. Nakano, and T. Tokuyama, *Computing and Combinatorics: 5th Annual International Conference* (COCOON'99), *Lecture Notes in Computer Science* **1627**, Springer (1999) 483–492.

[54] Little, J., and K. Darby-Dowman. The significance of constraint logic programming to operational research, in M. Lawrence and C. Wilson, eds., *Operational Research Tutorial Papers 1995*, Operational Research Society (1995) 20–45.

[55] Loveland, D. W., *Automated Theorem Proving: A Logical Basis*, North-Holland (Amsterdam, 1978).

[56] Mannila, H., and K. Mehlhorn, A fast algorithm for renaming a set of clauses as a Horn set, *Information Processing Letters* **21** (1985) 261–272.

[57] Milano, M., *Constraint and Integer Programming: Toward a Unified Methodology*, Operations Research/Computer Science Interfaces Series, Springer (2003).

[58] Moskewicz, M. W., C. F. Madigan, Y. Zhao, L. Zhang, S. Malik, Chaff: Engineering an efficient SAT solver, in *Proceedings of the 38th Design Automation Conference*, ACM (New York, 2001) 530–535.

[59] Nishimura, N., P. Ragde, and S. Szeider, Detecting backdoor sets with respect to Horn and binary clauses, *Theory and Applications of Satisfiability Testing, Seventh International Conference* (SAT 2004).

[60] Nobili, P., and A. Sassano, $(0, \pm 1)$ ideal matrices, *Mathematical Programming* **80** (1998) 265–281.

[61] Ostergard, P. R. J., A fast algorithm for the maximum clique problem, *Discrete Applied Mathematics* **120** (2002) 197–207.

[62] Pardalos, P., and G. Rogers, A branch and bound algorithm for the maximum clique problem, *Computers and Operations Research* **19** (1992) 363–375.

[63] Pardalos, P. M., N. Desai, An algorithm for finding a maximum weighted independent set in an arbitrary graph, *International Journal of Computer Mathematics* **38** (1991) 163–175.

[64] Paris, L., R. Ostrowski, P. Siegel, and L. Sais, Computing Horn strong backdoor sets thanks to local search, *18th IEEE International Conference on Tools with Artificial Intellignece* (ICTAI 2006) 139–143.

[65] Paris, L., R. Ostrowski, P. Siegel, and L. Sais, From Horn strong backdoor sets to ordered strong backdoor sets, *Advances in Artificial Intelligence* (MICAI 2007), *Lecture Notes in Computer Science* **4827**, Springer (2007) 105–117.

[66] Quine, W. V., The problem of simplifying truth functions, *American Mathematical Monthly* **59** (1952) 521–531.

[67] Quine, W. V., A way to simplify truth functions, *American Mathematical Monthly* **62** (1955) 627–631.

[68] Schaeffer, T. J., The complexity of satisfiability problems, *Proceedings, 10th Annual ACM Symposium on Theory of Computing* (STOC'78), ACM (New York, 1978) 216–226.

[69] Schlipf, J. S., F. S. Annexstein, J. V. Franco, and R. P. Swaminathan, On finding solutions for extended Horn formulas, *Information Processing Letters* **54** (1995) 133–137.

[70] Shindo, M., and E. Tomita, A simple algorithm for finding a maximum clique and its worst-case time complexity, *Systems and Computers in Japan* **21** (1990) 1–13.

[71] Swaminathan, R. P., and D. K. Wagner, The arborescence-realization problem, *Discrete Applied Mathematics* **59** (1995) 267–283.

[72] Szeider, S., Backdoor sets for DLL subsolvers, *Journal of Automated Reasoning* **35** (2005) 73–88.

[73] Tarjan, R. E., and A. E. Trojanowski, Finding a maximum independent set, *SIAM Journal on Computing* **6** (1977) 537–546.

[74] Thorsteinsson, E. S., Branch-and-check: a hybrid framework integrating mixed integer programming and constraint logic programming, in T. Walsh, *Principles and Practice of Constraint Programming* (CP2001), *Lecture Notes in Computer Science* **2239** Springer (2001) 16-30.

[75] Williams, R., C. Gomes, and B. Selman, Backdoors to typical case complexity, *Proceedings, International Joint Conference on Airtificial Intelligence* (IJCAI 2003) 11731178.

[76] R. Williams, C. Gomes, and B. Selman. On the connections between heavy-tails, backdoors, and restarts in combinatorial search, *Theory and Applications of Satisfiability Testing, 6th International Conference* (SAT 2003), *Lecture Notes in Computer Science* **2919**, Springer (2004) 222-230.

[77] Williams, H. P., and J. M. Wilson. Connections between integer linear programming and constraint logic programming–An overview and introduction to the cluster of articles, *INFORMS Journal on Computing* **10** (1998) 261–264.

[78] Wolsey, L. A., *Integer Programming*, John Wiley & Sons (New York, 1998).

[79] Yamada, Y., E. Tomita, and H. Takahashi, A randomized algorithm for finding a near-maximum clique and its experimental evaluations, *Systems and Computers in Japan* **25** (1994) 1–7.

[80] Yan, H., and J. N. Hooker, Logic circuit verification by Benders decomposition, in V. Saraswat and P. Van Hentenryck, eds., *Principles and Practice of Constraint Programming: The Newport Papers*, MIT Press (Cambridge, MA, 1995) 267–288.

[81] Zhang, Lintao, and S. Malik, The quest for efficient Boolean satisfiability solvers, in A. Voronkov, *Automated Deduction* (CADE-18), *18th International Conference on Automated Deduction*, *Lecture Notes in Artificial Intelligence* **2392**, Springer (2002) 295–313.