Logic-Based Benders Decomposition

Article · March 2001

Source: CiteSeer

CITATIONS
181

READS
937

2 authors:

John N. Hooker
Carnegie Mellon University
299 PUBLICATIONS **7,269** CITATIONS
SEE PROFILE

Greger Ottosson
IBM
24 PUBLICATIONS **1,372** CITATIONS
SEE PROFILE

Logic-Based Benders Decomposition *

J. N. Hooker
Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213 USA

November 1995

Abstract

Benders decomposition uses a strategy of "learning from one's mistakes" that has been employed in a more general way by constraint satisfaction methods. The aim of this paper is to achieve some of the generality of the latter methods while exploiting problem structure in much the way that is done by Benders decomposition. This is accomplished by extending Benders' approach to a logic-based context in which the linear programming dual of the subproblem is generalized to become an inference dual. Separation lemmas for duality, such as the Farkas lemma, are reinterpreted as completeness theorems for logical inference methods. The approach is illustrated by working out the details for the propositional satisfiability problem and 0-1 programming problems.

Benders decomposition [3, 6] uses a problem-solving strategy that can be generalized to a larger context. It assigns some of the variables trial values and finds the best solution consistent with these values. In the process it learns something about the quality of other trial solutions. It uses this information to reduce the number of solutions it must enumerate to find an optimal solution.

The problem-solving idea is basically that of "learning from one's mistakes." It has been developed in a general way in the constraint programming literature under the rubric of *nogoods* ([22], Section 5.4; [16]). When in the process of solving a problem one deduces that a certain partial solution cannot be completed to obtain a feasible solution, one can examine the reasons for this. Often the same reasons lead to a constraint that excludes a number of partial solutions. Such a constraint is a nogood.

*This research is partially supported by U.S. Office of Naval Research Grant N00014-95-1-0517 and by the Engineering Design Research Center at Carnegie Mellon University, an Engineering Research Center of the National Science Foundation, under grant EEC-8943164.

Benders decomposition uses a more specific strategy. It begins by partitioning the variables of a problem into two vectors x and y. It fixes y to a trial value so as to define a linear "subproblem" that contains only x. If the solution of the subproblem reveals that the trial value of y is unacceptable, the solution of the subproblem's *dual* is used to identify a number of *other* values of y that are likewise unacceptable. The next trial value must be one that has not been excluded. Eventually only acceptable values remain, and if all goes well, the algorithm terminates after enumerating only a few of the possible values of y. (The method is presented in more detail below.)

Because Benders decomposition searches for an optimal as well as a feasible solution, it actually enumerates trial values of (z, y), where z is the objective function value. Its "nogoods" have the form $z \geq \beta(y)$, where $\beta(y)$ is a bound on the optimal value that depends on y. The constraints $z \geq \beta(y)$ are known as *Benders cuts* and can rule out a large number of values of (z, y).

The specialized context of Benders decomposition enhances the general strategy in two ways.

- The pre-arranged partition of variables can exploit problem structure. When y is fixed, the resulting subproblem may simplify substantially or decouple into a number of small subproblems.

- The linearity of the subproblem allows one to obtain a Benders cut in an easy and systematic way, namely by solving the linear programming dual.

The intent here is to generalize a Benders-like strategy while retaining these two advantages to a large degree. The key to doing so is generalize the notion of a dual. The dual must be definable for any type of subproblem, not just linear ones, and must provide an appropriate bound on the optimal value.

Such a dual can be formulated simply by observing that a bound is obtained by *inferring* it from the constraints. A generalized dual can therefore defined as an *inference dual*, which is the problem of inferring a strongest possible bound from the constraint set. The classical linear programming dual is the inference dual problem for linear optimization.

The inference dual, which has apparently not been explicitly studied, is of interest in its own right. In general it permits one to address an optimization problem by methods of logical inference. In fact, it reinterprets classical separation lemmas as completeness theorems for logical inference. The separation problem for 0-1 programming problem, for instance, is that of finding a complete inference method for 0-1 linear inequalities. Such a method was introduced in [10]. The inference dual also generalizes the notion of sensitivity analysis to the task of identifying the role of constraints as premises in a proof of the dual optimum. The focus here, however, will be on the role of inference duality in generalizing Benders decomposition.

The aim of this paper is not so much to solve any particular problem as to propose a general problem-solving strategy. Nonetheless it is important to

show how the strategy works itself out in particular cases. To this end logic-based Benders decomposition is applied here to the propositional satisfiability problem and to 0-1 programming. The satisfiability problem is chosen because it a) illustrates the ideas in a lucid way, b) provides the tools needed to address 0-1 programming, and c) is an important combinatorial problem in its own right. The 0-1 programming problem is addressed because of the attention it has historically received. The application of logic-based Benders decomposition to these two problems yields algorithms that, to the writer's knowledge, have not hitherto been proposed. It should be emphasized, however, that the generalized Benders approach could be effective for many other problems, including those that are not formulated with 0-1 variables.

The first section below develops an elementary theory of inference duality, and the second shows how linear programming duality is a special case. The next two sections present logic-based Benders decomposition in the abstract, followed by its classical realization. Section 5 and 6 develop a duality theory and a Benders approach to the propositional satisfiability problem. Sections 7 and 8 do the same for 0-1 programming. The final section is reserved for concluding remarks.

1 Inference Duality

Consider a general optimization problem,

$$\begin{array}{ll} \min & f(x) \\ \text{subject to} & x \in S \\ & x \in D. \end{array} \tag{1}$$

The *domain* D is distinguished from the *feasible set* S. The domain might be the set of real vectors, 0-1 vectors, etc. The feasible set is generally defined by a collection of constraints.

To state the inference dual it is necessary to define implication with respect to a domain. Let P and Q be two propositions whose truth or falsehood is a function of x. Then P *implies* Q *with respect to* D (notated $P \overset{D}{\longrightarrow} Q$) if Q is true for any $x \in D$ for which P is true.

The *inference dual* of (1) is

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & x \in S \overset{D}{\longrightarrow} f(x) \geq \beta. \end{array} \tag{2}$$

The dual seeks the largest β for which $f(x) \geq \beta$ can be inferred from the constraint set. In other words, the dual problem is to find a strongest possible bound on the objective function value.

A strong duality theorem is almost true by definition. It is convenient to let the optimal value of a minimization problem be respectively ∞ or $-\infty$ when the problem is infeasible or unbounded, and vice-versa for a maximization problem.

Theorem 1 (Strong Inference Duality) *The optimization problem (1) has the same optimal value as its inference dual (2).*

Proof. If β^* is a finite optimal value of (1), then $x \in S$ implies $f(x) \geq \beta^*$, so that β^* is feasible in the dual. The dual cannot have an optimal value larger than β^* because this would mean that $f(x) = \beta^*$ cannot be achieved in (1). If (1) is infeasible, then any β is feasible in (2), which therefore has optimal value ∞. If (1) is unbounded, then (2) is infeasible with optimal value $-\infty$. □

Because strong duality is trivial, interesting duality theorems can take the form of soundness and completeness theorems for the type of inference used in the dual. An inference method is *sound* when it obtains only valid inferences, and it is *complete* when it obtains all valid inferences. It will be seen that these soundness and completeness theorems generalize the classical separation lemmas for linear systems.

2 Linear Programming Duality

The inference dual of a linear programming problem,

$$\begin{array}{ll} \min & cx \\ \text{s.t.} & Ax \geq a \\ & x \geq 0, \end{array} \tag{3}$$

can be stated,

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & Ax \geq a \xrightarrow{R^n} cx \geq \beta. \end{array} \tag{4}$$

The implication with respect to R^n used here might be called *linear implication*. The classical separation lemma for linear programming is essentially a characterization of linear implication, because it can be stated as follows.

Theorem 2 (Linear inference) *$Ax \geq a$ linearly implies $cx \geq \beta$ if and only if $Ax \geq a$ is infeasible or there is a real vector $u \geq 0$ for which $uA \leq c$ and $ua \geq \beta$.*

That is, a feasible linear system $Ax \geq a$ linearly implies an inequality $cx \geq \beta$ when some positive linear combination of inequalities in the system dominates $cx \geq \beta$ (in the sense that $uA \leq c$ and $ua \geq \beta$). So the separation lemma provides a complete inference method for linear implication. The historical Farkas lemma is a corollary of Theorem 2.

Corollary 3 *$Ax \geq a$ is an infeasible system if and only if there exists $u \geq 0$ for which $uA \leq 0$ and $ua > 0$.*

The separation lemma implies that classical linear programming duality is a special case of inference duality. The classical dual of (3) is,

$$\begin{array}{ll} \max & ua \\ \text{s.t.} & uA \leq c \\ & u \geq 0. \end{array} \tag{5}$$

Corollary 4 (Classical linear programming duality) *The optimal value of a linear programming problem (3) is the same as that of its classical dual (5), except when both are infeasible.*

Proof. The proof is elementary, but it is instructive to see precisely how inference and classical dualities relate. By Theorem 1, to show classical duality it suffices to show that (5) has the same value as the inference dual (4), except when both (3) and (5) are infeasible. First suppose that (3) is feasible. If (3) is unbounded, then by Theorem 2 the classical dual (5) is infeasible and likewise has an optimal value of $-\infty$. If (3) has a finite optimal value, then it is the largest β for which $Ax \geq a$ and $x \geq 0$ linearly imply $cx \geq \beta$. By Theorem 2, it is the largest β for which $uA + v \leq c$ and $ua \geq \beta$ for some $(u, v) \geq 0$ (where v is a vector of multipliers associated with $x \geq 0$). But this is also the largest β for which $uA \leq c$ and $ua \geq \beta$ for some $u \geq 0$. The latter problem is just the classical dual (5).

This shows that (3) and (5) have the same value when (3) is feasible. Because the primal problem (3) is the dual of the dual (5), by symmetry (3) and (5) have the same optimal value when (5) is feasible. So they have the same optimal value unless both are infeasible, as claimed. □

Note that the inference dual, unlike the classical dual, requires no regularity condition (i.e., that the primal or dual be feasible).

3 Benders Decomposition in the Abstract

Benders decomposition views elements of the feasible set as pairs (x, y) of objects that belong respectively to domains D_x, D_y. So the optimization problem (1) becomes,

$$\begin{array}{ll} \min & f(x, y) \\ \text{s.t.} & (x, y) \in S \\ & x \in D_x, y \in D_y. \end{array} \tag{6}$$

A general Benders algorithm begins by fixing y at some trial value $\overline{y} \in D_y$. This results in the *subproblem*,

$$\min \quad f(x, \overline{y}) \tag{7}$$

$$\begin{array}{ll} \text{s.t.} & (x, \overline{y}) \in S \\ & x \in D_x. \end{array}$$

Rather than solve this subproblem directly, the algorithm solves the inference dual.

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & (x, \overline{y}) \in S \xrightarrow{D_x} f(x, \overline{y}) \geq \beta. \end{array} \tag{8}$$

The dual problem is to find the best possible lower bound β^* on the optimal cost that can be inferred from the constraints, assuming y is fixed to $\overline{y}$.

The heart of Benders decomposition is somehow to derive a *function* $\beta_{\overline{y}}(y)$ that gives a valid lower bound on the optimal value for any value of y; for $y = \overline{y}$ it simply gives the optimal value β^*, so that $\beta_{\overline{y}}(\overline{y}) = \beta^*$.

Just how this is done varies from one context to another, but essentially the idea is this. A complete inference method for the implication $\xrightarrow{D_x}$ in the dual is identified by proving a generalized separation lemma; in the case of linear implication, it is nonnegative linear combination. This inference method is applied to derive the best lower bound β^* on the optimal value z, given that $y = \overline{y}$. One then notes what valid lower bound *this same line of argument* yields for *other* values of y. This bound is expressed as a function $\beta_{\overline{y}}(y)$ of y, yielding a *Benders cut* $z \geq \beta_{\overline{y}}(y)$ The subscript $\overline{y}$ reflects which value of y gave rise to the bounding function.

The algorithm proceeds as follows. At each iteration the Benders cuts so far generated comprise the constraints of a *master problem*,

$$\begin{array}{ll} \min & z \\ \text{s.t.} & z \geq \beta_{y^k}(y), \quad k = 1, \ldots, K \\ & y \in D_y. \end{array} \tag{9}$$

Here $y^1, \ldots, y^K$ are the trial values of y hitherto obtained. The next trial value $(\overline{z}, \overline{y})$ of (z, y) is obtained by solving the master problem. If the optimal value β^* of the resulting subproblem dual (8) is equal to $\overline{z}$, the algorithm terminates with optimal value $\overline{z}$. Otherwise a new Benders cut $z \geq \beta_{\overline{y}}(y)$ is added to the master problem. The subproblem dual need not be solved to optimality if a suboptimal β is found that is better than $\overline{z}$. At this point the procedure repeats.

A precise statement of the algorithm appears in Fig. 1. Note that if the subproblem is infeasible, the dual is unbounded, so that $\beta^* = \beta_{\overline{y}}(\overline{y}) = \infty$.

Theorem 5 *Suppose that in each iteration of the generic Benders algorithm, the bounding function $\beta_{\overline{y}}$ satisfies the following.*

(B1) *The Benders cut $z \geq \beta_{\overline{y}}(y)$ is valid; i.e., any optimal solution (z^*, x^*, y^*) of (6) satisfies $z^* \geq \beta_{\overline{y}}(y^*)$.*

```
Choose an initial ȳ ∈ D_y in problem (6).
Set z̄ = −∞ and k = 0.
While the subproblem dual (8) has a feasible solution β > z̄:
   Formulate a lower bound function β_ȳ(y) with β_ȳ(ȳ) = β.
   Let k = k + 1, let y^k = ȳ, and add the Benders cut
      z ≥ β_{y^k}(y) to the master problem (9).
   If the master problem (9) is infeasible then
      Stop; (6) is infeasible.
   Else
      Let ȳ be an optimal solution of (9) with
         optimal value z̄.
The optimal value of (6) is z̄.
```

Figure 1: *The generic Benders algorithm.*

Then if the algorithm terminates with a finite optimal solution $(z, y) = (\overline{z}, \overline{y})$ in the master problem, (6) has an optimal solution $(x, \overline{y})$ with value $f(x, \overline{y}) = \overline{z}$. If it terminates with an infeasible master problem, then (6) is infeasible. If it terminates with an infeasible subproblem dual, then (6) is unbounded.

Proof. Suppose first that the algorithm terminates with a finite solution $(z, y) = (\overline{z}, \overline{y})$ that is optimal in the master problem. Due to (B1), any feasible solution of (6) with value $\overline{z}$ is optimal in (6). But the last subproblem dual (8) solved has finite optimal value $\beta^* = \overline{z}$, which means by Theorem 1 that the subproblem (7) has an optimal solution $\overline{x}$. So $(\overline{z}, \overline{x}, \overline{y})$ is feasible and therefore optimal in (6).

Because all Benders cuts are valid, infeasibility of the master problem implies that (6) is infeasible.

Finally, if the subproblem dual is infeasible, then by Theorem 1 the subproblem is unbounded. This implies that (6) is unbounded. □

The algorithm need not terminate in general, even if the subproblem is always solved to optimality. Consider for example the problem,

$$\begin{array}{ll} \min & y \\ \text{s.t.} & y \geq x \\ & x \geq 1 \\ & y \geq 0 \\ & x, y \in R, \end{array}$$

which has the optimal solution $(x, y) = (1, 1)$. It is consistent with the algorithm to set

$$\beta_{\overline{y}}(y) = \begin{cases} \infty & \text{if } y < \frac{1}{2}(1 + \overline{y}), \\ 0 & \text{otherwise.} \end{cases}$$

Then initially $(\overline{z}, \overline{y}) = (0, 0)$, and the solutions y^k of the master problem follow the sequence $1 - (\frac{1}{2})^k$ for $k = 1, 2, \ldots$, so that the algorithm never terminates.

If the domain of y is finite, however, the algorithm must terminate, because only finitely subproblems can be defined. Because $\overline{z}$ must increase in every iteration but the last, the optimal value is reached after finitely many steps.

Theorem 6 *Suppose that in each iteration of the generic Benders algorithm, the bounding function $\beta_{\overline{y}}$ satisfies (B1) and the following.*

(B2) *$\beta_{\overline{y}}(\overline{y}) = \beta$, where β is the solution value obtained in the subproblem dual (8).*

Then if D_y is finite and the subproblem dual is solved to optimalty, the generic Benders algorithm terminates.

Proof. Due to (B2), the subproblem value β must increase in every iteration but the last. Because D_y is finite and the subproblem dual is solved to optimality, the latter can generate only finitely many values. So the algorithm must terminate. □

4 Classical Benders Decomposition

The classical Benders technique applies to problems in which the subproblem is linear.

$$\begin{array}{ll} \min & cx + f(y) \\ \text{s.t.} & Ax + g(y) \geq a \\ & x \geq 0 \\ & x \in R^n, \;\; y \in D_y, \end{array} \tag{10}$$

where $g(y)$ is a vector of functions $g_i(y)$. The subproblem (7) becomes,

$$\begin{array}{ll} \min & cx + f(\overline{y}) \\ \text{s.t.} & Ax \geq a - g(\overline{y}) \\ & x \geq 0. \end{array} \tag{11}$$

Due to the classical separation lemma (Theorem 2), the subproblem dual (7) can be written,

$$\begin{array}{ll} \max & u(a - g(\overline{y})) + f(\overline{y}) \\ \text{s.t.} & uA \leq c \\ & u \geq 0. \end{array} \tag{12}$$

If the dual has a finite solution u, it provides an inference procedure (a linear combination) for obtaining a bound

$$z \geq u(a - g(\overline{y})) + f(\overline{y})$$

that is valid when $y = \overline{y}$. The key to obtaining a bound for any y is to observe that this same u remains feasible in the dual for any y. So the same procedure (i.e., the same u) provides a bound

$$z \geq \beta_{\overline{y}}(y) = u(a - g(y)) + f(\overline{y})$$

for any y. This is the classical Benders cut. It is straightforward to show that when the dual is infeasible or unbounded, one can obtain a cut

$$v(a - g(y)) \leq 0,$$

where v solves

$$\begin{array}{ll} \max & v(a - g(\overline{y})) \\ \text{s.t.} & vA \leq 0 \\ & v \geq 0. \end{array}$$

5 The Satisfiability Dual

The propositional satisfiability problem is to find an assignment of truth values to logical variables (*atomic propositions*) x_j for which a given set of logical *clauses* are all true. A clause is a disjunction of *literals*, each of which is an atomic proposition or its negation. For instance, $x_1 \vee \neg x_2 \vee \neg x_3$ is a clause asserting that x_1 is true or x_2 is false or x_3 is false. One clause C implies another D if and only of C *absorbs* D; i.e., C contains all the literals in D.

If set of logical clauses is written $\{g_i(x) \mid i = 1, \ldots, m\}$, the satisfiability problem can be formulated as an optimization problem by adding an artificial variable x_0:

$$\begin{array}{ll} \min & v(x_0) \\ \text{s.t.} & x_0 \vee g_i(x), \quad i = 1, \ldots, m. \end{array} \tag{13}$$

Here $v(x_0)$ is 1 if x_0 is true and 0 if it is false. The minimum value of $v(x_0)$ is zero (i.e., x_0 can be false) if and only if the clauses are satisfiable.

The inference dual is,

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & \{x_0 \vee g_i(x) \mid i = 1, \ldots, m\} \overset{\{T,F\}^{n+1}}{\longrightarrow} v(x_0) \geq \beta. \end{array} \tag{14}$$

A complete inference method for logical clauses, and therefore a separation lemma, is well known and was discovered by W. V. Quine over 40 years ago

```
Let S be a set of logical clauses.
Perform resolve(S).

Procedure resolve(S')
   If S' contains clauses C, D that have a resolvent R
   that no clause in S' absorbs then
      Perform resolve(S' ∪ {R}).
   Else stop.
```

Figure 2: *The resolution algorithm.*

[18, 19]. It is the *resolution* method, known as *consensus* when applied to formulas in disjunctive normal form. The name 'resolution' actually derives from Robinson [20], who developed the method for first-order predicate logic. Given two clauses for which exactly one variable x_j occurs positively on one and negatively in the other, the *resolvent* of the clauses is a clause consisting of all the literals in either clause except x_j and $\neg x_j$. For instance, the third clause below is the resolvent of the first two.

$$\begin{array}{rr} x_1 \vee x_2 \vee \neg x_3 & \\ \neg x_1 \qquad \vee \neg x_3 \vee \neg x_4 & \\ x_2 \vee \neg x_3 \vee \neg x_4 & \end{array}$$

(Resolution has close connections with cutting planes, and a resolvent is in fact a rank 1 cut [8, 9, 13].) The resolution algorithm simply generates nonredundant resolvents as long as possible. It appears in Fig. 2. The separation lemma can be stated,

Theorem 7 (Quine.) *The resolution algorithm is finite, and a satisfiable set S of logical clauses implies a given clause C if and only if the algorithm generates a set S' containing a clause that absorbs C.*

An immediate corollary is parallel to the Farkas lemma.

Corollary 8 *A set S of clauses is unsatisfiable if and only if the resolution algorithm generates a set S' that contains the empty clause.*

The corollary is derived by using an artificial variable, as in the classical case. Given S, let $\overline{S}$ be the set that results from adding the literal x_0 to every clause in S. Then $\overline{S}$ is satisfiable, and S is unsatisfiable if and only if $\overline{S}$ implies x_0. By Theorem 7 the latter is true if and only if there is a resolution proof of x_0 from $\overline{S}$. But this becomes a resolution proof of the empty clause from S by deleting x_0 from every clause in the proof.

The dual (14) can in principle be solved by the resolution algorithm directly, but this tends to be a very inefficient approach [7, 8]. It is generally much more

effective to solve the problem with a "primal" method, such as branching, and then to reconstruct a dual solution from information obtained in the primal solution process—much as is done for the classical linear programming dual. This is readily done for the satisfiability problem. Consider for example the problem,

$$\begin{array}{rrrrl} x_1 \vee & x_2 & & & (a) \\ x_1 \vee \neg x_2 & & & & (b) \\ \neg x_1 & & \vee & x_3 & (c) \\ \vee & x_2 \vee \neg x_3 & & & (d) \\ \vee & \neg x_2 \vee \neg x_3 & & & (e) \\ \neg x_1 \vee & x_2 \vee & & x_3 & (f) \end{array} \tag{15}$$

A possible enumeration tree appears in Fig. 3. At the root node it branches on the two values of x_1, and similarly at other nodes. The clauses that remain at each node are shown. Because clauses are falsified at every leaf, the original clauses are unsatisfiable.

It is straightforward to reconstruct a resolution proof of unsatisfiability from the search tree, as shown in Fig. 4. At each leaf, the clause that is falsified is used as a premise. (At one node, there is a choice between clauses (c) and (f); (c) is chosen.) The clause at every other node is the resolvent of the clauses at its two children. The general algorithm appears in Fig. 5.

Theorem 9 *Given any enumeration tree that proves unsatisfiability of a set S of clauses, the algorithm of Fig. 5 creates a resolution proof of unsatisfiability.*

Proof. It suffices to show that each node P of T has two properties:

a) C_P is falsified by the variables that are fixed between the root and P;

b) P is a leaf node, or C_P is the resolvent of C_Q, C_R for two successor nodes Q, R.

This suffices because (a) implies that at least the root is associated with the empty clause, which means by (b) that a resolution proof of unsatifiability has been constructed.

These properties can be shown by induction on depth in T. First consider a node P at maximum depth in T, so that P is a leaf. Then (a) holds by choice of C_P, and (b) holds by stipulation.

Now suppose properties (a) and (b) hold for all nodes at depths greater than d, and show that they hold for a node P at depth d. If P is a leaf, the argument is as above. If it is a nonleaf node, then its children Q, R satisfy (a) and (b). There are two cases. i) C_P is C_Q or C_R. Then (a) and (b) are true by the induction hypothesis. ii) C_P is the resolvent on x_j of C_Q and C_R. Then (b) is true by definition of C_P. Also (a) holds because it holds for C_Q and C_R, and resolution on x_j ensures that neither x_j nor $\neg x_j$ occurs in C_P. □

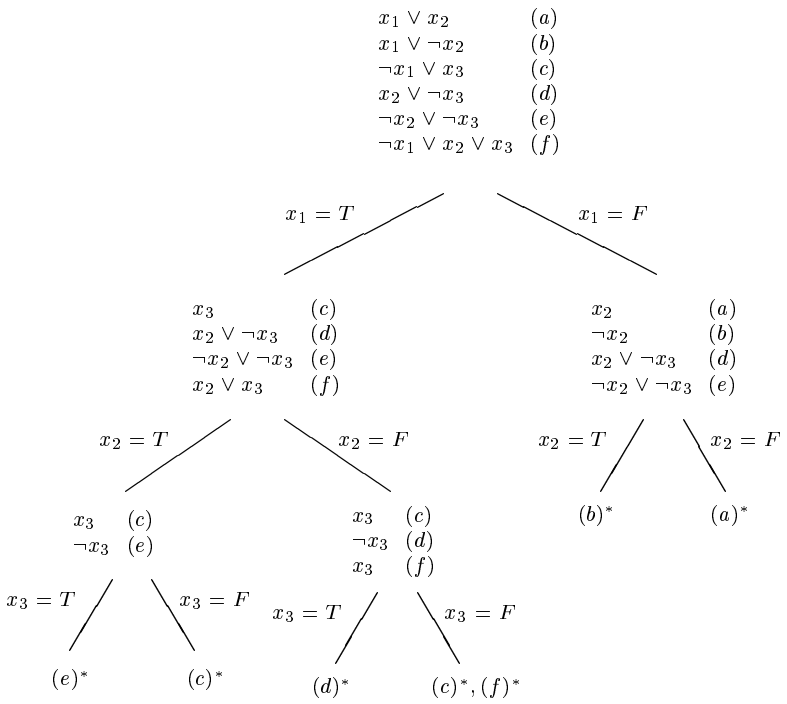

Figure 3: *A solution of the satisfiability subproblem by branching. The clauses remaining at each nonleaf node are indicated. The clauses falsified at each leaf node are indicated with an asterisk.*

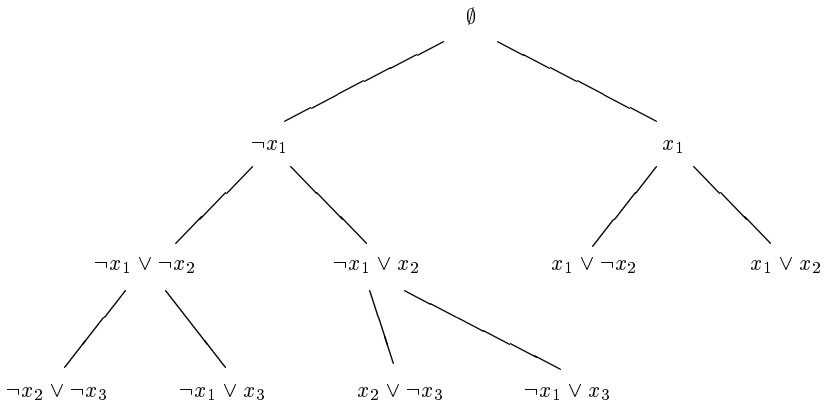

Figure 4: *Construction of a resolution proof of unsatisfiability from the enumeration tree showing unsatisfiability. Each clause at a nonleaf node is the resolvent of the clauses at the two successor nodes.*

```
Let T be the enumeration tree that demonstrates unsatisfiability
   a set S of clauses.
Associate with each leaf node P of T a clause C_P that is
   falsified at P.
Put each leaf node in a list A of active nodes.
While A contains nodes Q, R with the same parent P:
   Let x_j be the variable fixed to true and false in order to
      create nodes Q, R.
   If C_Q and C_R can be resolved on x_j, then
      Deduce the resolvent from C_Q and C_R and call it C_P.
   Else
      Let C_P be either C_Q or C_R.
   Remove Q, R from A and add P to A.
```

Figure 5: *Algorithm for obtaining a dual (resolution) proof of clausal unsatisfability from a primal proof.*

6 Benders Decomposition for Satisfiability

If the propositional variables are partitioned (x, y), the satisfiability problem (13) can be written

$$\begin{array}{ll} \min & v(x_0) \\ \text{s.t.} & x_0 \vee g_i(x) \vee h_i(y), \quad i = 1, \ldots, m, \end{array} \tag{16}$$

where for each i, $g_i(x)$ or $h_i(y)$ may be empty.

For a fixed assignment $\overline{y}$ the subproblem is,

$$\begin{array}{ll} \min & v(x_0) \\ \text{s.t.} & x_0 \vee g_i(x_i), \quad i \in I(\overline{y}), \end{array} \tag{17}$$

where $I(\overline{y})$ is the set of i for which $h_i(\overline{y})$ is false. In practice the variables would be partitioned so that the subproblem has special structure; this is discussed further below. The subproblem dual is,

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & \{x_0 \vee g_i(x) \mid i \in I(\overline{y})\} \overset{\{T,F\}^{n+1}}{\longrightarrow} v(x_0) \geq \beta. \end{array} \tag{18}$$

Rather than solve the dual directly by resolution, one can reconstruct a resolution proof from an enumeration tree, as shown in the previous section. This proof can be used in a straightforward way to generate a Benders cut.

The idea is best explained by way of an example. Consider the satisfiability problem,

$$\begin{array}{lllllllll} \min & v(x_0) & & & & & & & \\ \text{s.t.} & x_0 \vee & x_1 \vee & x_2 & & \vee & y_1 \vee y_2 & & (a) \\ & x_0 \vee & x_1 \vee \neg x_2 & & & \vee & y_1 & & (b) \\ & x_0 \vee & \neg x_1 & & \vee & x_3 & & \vee y_2 \vee y_3 & (c) \\ & x_0 & & \vee & x_2 \vee \neg x_3 \vee & & y_1 \vee y_2 & & (d) \\ & x_0 & & \vee \neg x_2 \vee \neg x_3 & & & \vee y_2 & & (e) \\ & x_0 \vee & \neg x_1 \vee & x_2 \vee & x_3 \vee & & y_1 & & (f) \\ & x_0 \vee & x_1 \vee & x_2 \vee & x_3 \vee \neg y_1 & & & \vee y_3 & (g) \end{array}$$

If initially $\overline{y} = (F, F, F)$, clauses (a)-(f) appear in the subproblem, with the y_j terms removed. The subproblem therefore minimizes $v(x_0)$ subject to these clauses, which is equivalent to the satisfiability problem (15). The problem was shown to be unsatisfiable by the enumeration tree in Fig. 3.

The next step is to obtain a dual solution in the form of a resolution proof of unsatisfiability. Such a solution, shown in Fig. 4, was constructed from the enumeration tree (the primal solution). To obtain a Benders cut, one can carry along the variables y_j in the resolution steps, so as to deduce the clause $y_1 \vee y_2$ at the root, as illustrated in Fig. 6. The deduction of $y_1 \vee y_2$ implies that if y_1 and y_2 were fixed to false, the same clauses (a)-(e) would still appear in the

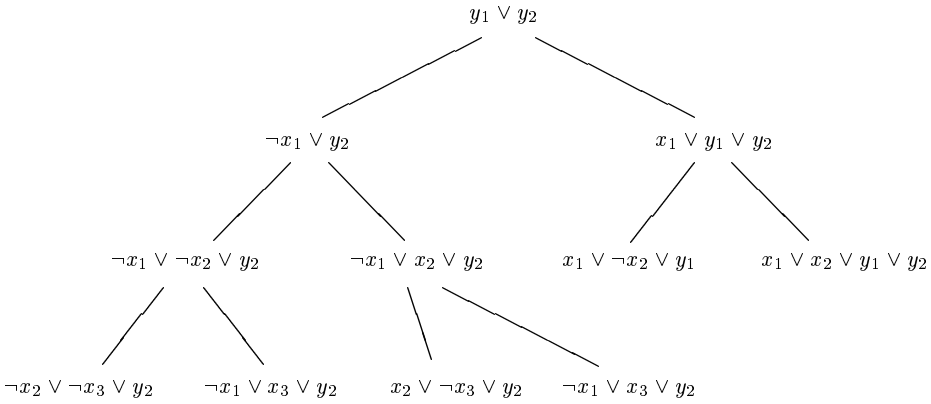

Figure 6: *Construction of a resolution proof of unsatisfiability of the subproblem. Each clause at a nonleaf node is the resolvent of the clauses at the two successor nodes. The variables y_j are carried along in order to obtain a Benders cut.*

subproblem (regardless of the value of y_3), and the same proof would be possible. So any solution in which $(y_1, y_2) = (F, F)$ fails to satisfy all the clauses. The master problem can therefore be stated

$$\begin{aligned} \min \quad & z \\ \text{s.t.} \quad & z \geq 1 - v(y_1 \vee y_2), \end{aligned} \tag{19}$$

or equivalently,

$$\begin{aligned} \min \quad & v(y_0) \\ \text{s.t.} \quad & y_0 \vee y_1 \vee y_2. \end{aligned}$$

The process of "carrying along" the y_j's analogous to a similar process in classical Benders decomposition. A linear combination of the right-hand sides of the subproblem, using the dual solution u as weights, gives the optimal value of the subproblem. To obtain a Benders cut $z \geq u(a - g(y)) + f(y)$, the terms involving y are "carried along" in this linear combination.

In the case of the satisfiability problem, there is actually a simpler way to derive the same Benders cuts. Merely observe that clauses (a)-(e) cover the leaves and therefore suffice as premises for a proof of unsatisfiability in the subproblem. To ensure that each of these clauses occurs in the subproblem it is enough to falsify the literals containing y_j's that occur in then; i.e., to

set $(y_1, y_2) = (F, F)$. So again $y_0 \vee y_1 \vee y_2$ is a Benders cut. This simplified procedure may not yield the same results in other contexts, as will be illustrated in the discussion of 0-1 programming below.

An optimal solution of the master problem (19) is $(\overline{z}, \overline{y}) = (0, F, T, F)$, for which the resulting subproblem is feasible. This proves satisfiability.

In general, when the optimal value of the subproblem (17) is $v(x_0) = 1$, a Benders cut is generated as follows. A set $\{g_i(x) \mid i \in I'(\overline{y})\}$ of clauses from the subproblem is identified, where $I'(\overline{y}) \subset I(\overline{y})$, such that at each leaf of the tree, at least one of the clauses is falsified. The Benders cut is

$$z \geq 1 - v\left(\bigvee_{i \in I'(\overline{y})} h_i(y)\right). \tag{20}$$

After K iterations the master problem is,

$$\begin{array}{ll} \min & z \\ \text{s.t.} & z \geq 1 - v\left(\bigvee_{i \in I'(y^k)} h_i(y)\right), \quad k = 1, \ldots, K \\ & h_i(y), \text{ all } i \text{ for which } g_i(x) \text{ is empty,} \\ & y_j \in \{0,1\}, \text{ all } j, \end{array}$$

or equivalently,

$$\begin{array}{ll} \min & v(y_0) \\ \text{s.t.} & y_0 \vee \bigvee_{i \in I'(y^k)} h_i(y), \quad k = 1, \ldots, K, \\ & h_i(y), \text{ all } i \text{ for which } g_i(x) \text{ is empty,} \\ & y_j \in \{0,1\}, \text{ all } j, \end{array} \tag{21}$$

where $y^1, \ldots, y^K$ are the solutions of previous master problems.

Thus it is not necessary actually to carry out the resolutions in the dual solution in order to obtain the Benders cut; it is enough to find a set of clauses that cover the leaves of the enumeration tree. (The structure of the resolution proof will be useful, however, in the treatment of 0-1 programming below.) The same Benders cuts can be found, then, if the simple branching algorithm used here is replaced with a more sophisticated algorithm (e.g., one of those discussed in [7, 17, 21]), provided the algorithm identifies a subset of premises sufficient to prove unsatisfiability. The Benders algorithm appears in Fig. 7.

Theorem 10 *The algorithm of Fig. 7 terminates, and it indicates satisfiability if and only if (16) is satisfiable.*

```
Set k = 0.
While (21) has an optimal solution (ȳ_0, ȳ) with ȳ_0 = false:
   If the subproblem (17) has no solution with x_0 = false then
      Choose I'(ȳ) ⊂ I(ȳ) so that the clauses g_i(x) for i ∈ I'(ȳ)
         are found during solution of (17) to be unsatisfiable
         (e.g., if (17) is solved by branching after fixing x_0
         to false, choose I'(ȳ) so that at each leaf node,
         there is at least one i ∈ I'(ȳ) for which clause g_i(x)
         is falsified at that node.)
      Let k = k + 1, let y^k = ȳ, and add the k-th Benders cut
         (20) to the master problem (21).
   Else stop; (13) is satisfiable.
(13) is unsatisfiable.
```

Figure 7: *A Benders algorithm for the propositional satisfiability problem.*

Proof. Because D_y is finite, due to Theorems 5 and 6 it suffices to show that $\beta_{\overline{y}}$ has properties (B1) and (B2).

(B1) The cuts (20) are valid for any y. This is true because $\beta_{\overline{y}}(y)$ is nonzero only when y violates all the clauses $h_i(y)$ in $I'(\overline{y})$. When this happens these clauses are in the subproblem, which therefore has value 1.

(B2) $\beta_{\overline{y}}(\overline{y}) = \beta^*$, where β^* is the optimal value of the subproblem. This follows from the fact that $h_i(\overline{y})$ is false for each $i \in I'(\overline{y})$. $\square$.

It is advantageous to find an $I'(\overline{y})$ that minimizes the number of variables y_j in the resulting Benders cut, because a shorter cut is stronger. This is accomplished by solving the following problem, exactly or approximately. Let Y_i be the set of variables that occur in $h_i(y)$, and let L_i be the set of leaf nodes at which $g_i(x)$ is falsified. Find a set $I'(\overline{y}) \in I(\overline{y})$ for which $\bigcup_{i \in I'(\overline{y})} L_i$ contains all leaf nodes and $|\bigcup_{i \in I'(\overline{y})} Y_i|$ is minimized.

As noted earlier, the variables should be partitioned so that the subproblems decouple or have some other special structure. For instance, a subproblem in which the clauses are renamable Horn (if x_0 is ignored) can be checked for satisfiability in linear time [1, 4]. Finding a small subset of variables y for which the subproblem is always renamable Horn is a maximum clique problem [5], which can be solved by various heuristics and exact algorithms.

In practice the master problem might be solved by branching on the variables y_j. Whenever a feasible solution is found, the subproblem is solved, and the Benders cut (if any) is added to the master problem. At this point the master search tree can be updated to reflect the additional clause; an efficient algorithm for doing this is presented in [11]. Note that in the context of a branching algorithm, Benders cuts have a role complementary to that of traditional cutting planes. Whereas the latter contain variables that have not yet

been fixed, Benders cuts contain variables that have already been fixed. They are also global (valid throughout the tree), as are nogoods in general.

Interestingly, when the subproblem is renamable Horn, it is equivalent to a linear programming problem [15], so that the original problem could be solved by traditional Benders decomposition. But the linear-time methods for solving the subproblem are much faster than methods for solving the linear programming equivalent.

Furthermore, it is shown in [14] that if the variables y_j represent inputs to a logic circuit, and the variables x_j represent the outputs of gates in the circuit, then the problem of checking whether the circuit represents a tautology can be solved by Benders decomposition, where the subproblem is renamable Horn. Here again the subproblem is equivalent to a linear programming problem, and the specialized Benders algorithm developed in [14] yields Benders cuts that are in fact equivalent to those obtained by classical Benders—but again much more rapidly than in the classical case.

7 0-1 Programming Duality

A 0-1 programming problem may be stated,

$$\begin{array}{ll} \min & cx + c_0 \\ \text{s.t.} & Ax \geq a \\ & x \in \{0,1\}^n. \end{array} \tag{22}$$

The constant c_0 will be useful in the next section. The inference dual is,

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & Ax \geq a \stackrel{\{0,1\}^n}{\longrightarrow} cx + c_0 \geq \beta. \end{array} \tag{23}$$

A separation lemma for this dual would consist of a complete inference method for linear 0-1 inequalities. Such a method was presented in [10]. It consists of the recursive application of two procedures, resolution and diagonal summation. Only resolution will ultimately be required for present purposes, but the entire procedure is given for completeness.

It is convenient to write an inequality $ax \geq \alpha$ (with integral a, α) in the form $ax \geq \delta + n(a)$, where $n(a)$ is the sum of the negative components of a, and δ is defined to be the *degree* of the inequality. If each $a_j \in \{0, 1, -1\}$ and $\delta = 1$, the inequality is *clausal*, because it represents a clause. For instance, the inequality $x_1 - x_2 - x_3 \geq 1 - 2$ represents the clause $x_1 \vee \neg x_2 \vee \neg x_3$.

Given a set of inequalities $\{a^i x \geq \delta + n(a^i) \mid i \in J\}$, where $J \subset N = \{1, \ldots, n\}$, the inequality $ax \geq \delta + 1 + n(a)$ is the *diagonal sum* of the set if

```
Let S be a feasible set of 0-1 linear inequalities.
Perform deduce(S).

Procedure deduce(S)
   If there are clausal inequalities C, D that have a resolvent
   R that no inequality in S implies, such that C and D are
   each implied by some inequality in S, then
      Perform deduce(S ∪ {R}).
   Else if there are inequalities I_1, ..., I_m ∈ T that have a diagonal
   sum I that no inequality in S implies, such that I_1, ..., I_m
   are each implied by some inequality in S, then
      Perform deduce(S ∪ {I}).
   Else stop.
```

Figure 8: *A complete inference algorithm for 0-1 linear inequalities.*

$a_j \neq 0$ for all $j \in J$, $a_j = 0$ for all $j \in N \setminus J$, and

$$a_j^i = \begin{cases} a_j - 1 & \text{if } j = i \text{ and } a_j > 0, \\ a_j + 1 & \text{if } j = i \text{ and } a_j < 0, \\ a_j & \text{otherwise.} \end{cases}$$

For instance, the fourth inequality below is the diagonal sum of the first three.

$$\begin{aligned} x_1 + x_2 - 3x_3 &\geq 2 - 3 \\ 2x_1 \quad - 3x_3 &\geq 2 - 3 \\ 2x_1 + x_2 - 2x_3 &\geq 2 - 2 \\ 2x_1 + x_2 - 3x_3 &\geq 3 - 3. \end{aligned}$$

Whereas resolution eliminates a variable, diagonal summation increases degree.

Figure 8 contains a statement of the inference method. The following is proved in [10] and can be viewed as a separation lemma for 0-1 duality. A class T of inequalities with integral coefficients and right-hand sides is *monotone* if it contains all clausal inequalities, and given any inequality $ax \geq \delta$ in T, T contains all inequalities $a'x \geq \delta' + n(a')$ such that $|a'_j| \leq |a_j|$ for all j, and $0 \leq \beta' \leq \beta$. In particular, the class of all 0-1 inequalities (with integral coefficients and right-hand sides) is monotone.

Theorem 11 *Let S be a feasible set of 0-1 linear inequalities with integral coefficients and right-hand sides, and let I be an inequality in a monotone set T of inequalities. Then the algorithm of Fig. 8 is finite, and if S' is the result of applying the algorithm to S, S implies I if and only if some inequality in S' implies I.*

Refinements of this theorem, particularly as it applies to inequalities with coefficients in $\{0, 1, -1\}$, have been presented by Barth [2].

The following is shown by adding an artificial variable as in the proof of Corollary 8, and by applying Theorem 11 with T equal to the set of clausal inequalities.

Corollary 12 *Let S be a set of 0-1 linear inequalities with integral coefficients and right-hand side. S is unsatisfiable if and only if the algorithm of Fig. 8, modified to use resolution only, generates a set S' that contains the empty clause.*

As in the case of the satisfiability problem, it will be useful to use information obtained while solving the primal problem to construct a solution of the dual. In fact, an enumeration tree permits one to reconstruct quite easily a proof of the optimal value using resolution alone.

Consider for example the problem,

$$\begin{array}{lll} \min & 7x_1 + 5x_2 + 3x_3 + 10 & \\ \text{s.t.} & 2x_1 + 5x_2 - x_3 + \geq 3 & (a) \\ & -x_1 + x_2 + 4x_3 + \geq 4 & (b) \\ & x_1 + x_2 + x_3 + \geq 2 & (c) \\ & x \in \{0,1\}^3 & . \end{array} \tag{24}$$

The enumeration tree of Fig. 9 solves the problem. The optimal solution value is 18.

A resolution proof that $z \geq 18$ can be reconstructed in a way very similar to that presented above for satisfiability problems. At each leaf node, one of the violated inequalities is chosen as a premise; in this case, constraints (a) and (b) suffice. If the leaf node represents a feasible solution, the objective function value $z = \beta$ is replaced with a premise consisting of the constraint $cx + c_0 \leq \beta - 1$. These combined premises must lead to a contradiction, thus proving that $z \geq 18$.

The contradiction can be demonstrated by resolution, as depicted in Fig. 10. The inequalities at nodes 8 and 9, for instance, respectively imply clauses that resolve to obtain $\neg x_1 \vee \neg x_2$ at node 4. The latter resolves with x_2, implied by the inequality at node 5, to obtain $\neg x_1$ at node 2.

Nodes 10 and 11 illustrate an important point. The inequality at node 10 implies a second clause $\neg x_1 \vee \neg x_3$ (not shown in Fig. 6) that resolves with x_3 at node 11. It will be seen below, however, that $\neg x_1 \vee \neg x_3$ can be neglected because it is not falsified by variables fixed between nodes 10 and the root. In fact it is necessary to infer, from the inequality at each leaf node, only one clause that is falsified by the variables fixed between that node and the root. This is generally straightforward to do, particularly because one need only consider clauses containing literals that are fixed to false.

Because the clauses inferred at the leaf nodes are falsified by the fixed variables, the enumeration tree proves they are unsatisfiable. So the algorithm of

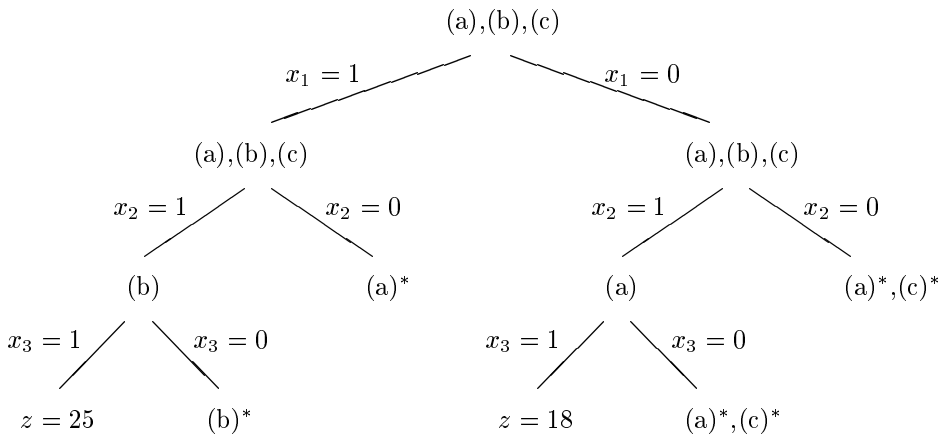

Figure 9: *A solution of the subproblem by branching. The constraints remaining at each nonleaf node are indicated. The constraints violated at each leaf node, if any, are indicated with an asterisk; if no constraints are violated, the objective function value is shown.*

Fig. 5 can be applied to them to generate a resolution proof of unsatisfiability. In this case, the empty clause is generated below the root, at node 3.

Theorem 13 *Given any enumeration tree that proves unsatisfiability of a set of 0-1 inequalities, the algorithm of Fig. 11 creates a resolution proof of unsatisfiability.*

Proof. By Theorem 9 it suffices to show the following, which imply that the enumeration tree in fact proves the unsatisfiability of the clauses C_P associated with the leaf nodes P.

(a) The inequality associated with any leaf node implies at least one clause that is falsified at that node. This is because any set of 0-1 points is the satisfaction set of some set of clauses. A 0-1 inequality is therefore equivalent to the set of clauses it implies, and an inequality that is falsified at a node must imply at least one clause that is falsified.

(b) Any falsified clause C_P implied by an inequality associated with a leaf node P is falsified at no predecessor node. If it were, the inequality that implies C_P would have been falsified at the predecessor node, and node P would not have been generated. □

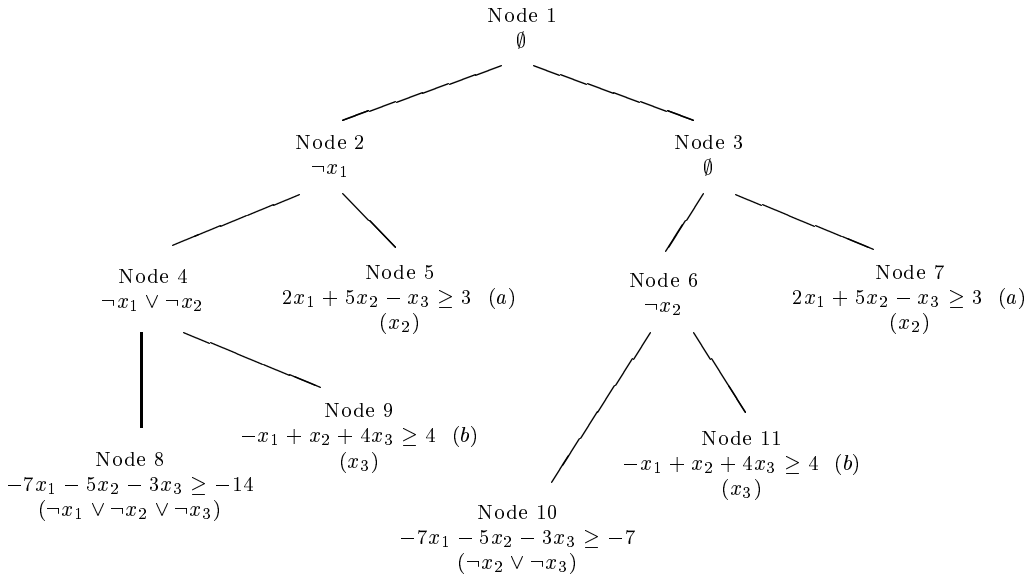

Figure 10: *Construction of a proof of $z \geq 18$. The violated constraint at each leaf node is shown, along with a falsified clause it implies. Resolvents are shown at the nonleaf nodes.*

```
Let T be the enumeration tree that demonstrates unsatisfiability
   a set of 0-1 inequalities.
Associate with each leaf node P of T an inequality that is
   violated at P.  Infer from this inequality a clause C_P
   that is falsified at P.
Let S contain the clauses C_P, so that T is an enumeration
   tree that proves the unsatisfiability of S
Apply the algorithm of Fig. 5 to S.
```

Figure 11: Algorithm for obtaining a dual (resolution) proof of 0-1 unsatisfiability from a primal proof.

8 Benders Decomposition for 0-1 Programming

Benders decomposition has long been applied to 0-1 programming. The difference here is that the subproblem is a 0-1 programming problem, rather than a linear programming problem as in the classical case. This application also illustrates the principle that the Benders cuts $z \geq \beta_{\overline{y}}(y)$ can be weakened to speed computation without sacrificing optimality, provided $\beta_{\overline{y}}(\overline{y})$ is not changed.

If the variables are partitioned (x, y), the problem can be written,

$$\begin{array}{ll} \min & cx + f(y) \\ \text{s.t.} & Ax + By \geq a \\ & x \in \{0,1\}^n,\ y \in \{0,1\}^p. \end{array} \tag{25}$$

For a fixed $\overline{y}$ the subproblem is,

$$\begin{array}{ll} \min & cx + f(\overline{y}) \\ \text{s.t.} & Ax \geq a - B\overline{y} \\ & x \in \{0,1\}^n, \end{array} \tag{26}$$

The subproblem dual is,

$$\begin{array}{ll} \max & \beta \\ \text{s.t.} & Ax \geq a - B\overline{y} \overset{x \in \{0,1\}^n}{\longrightarrow} cx + f(\overline{y}) \geq \beta. \end{array} \tag{27}$$

The dual can in principle be solved directly by the algorithm of the previous section, but again it is probably much faster to extract a dual solution from the enumeration tree that solves the primal. This in turn provides a Benders cut.

Consider the following example.

$$\begin{array}{lrrrrrrrrl} \min & 7x_1 & +5x_2 & +3x_3 & +10y_1 & +20y_2 & & & \\ \text{s.t.} & 2x_1 & +5x_2 & -x_3 & +2y_1 & +3y_2 & \geq 5 & & (a) \\ & -x_1 & +x_2 & +4x_3 & +y_1 & +2y_2 & \geq 5 & & (b) \\ & x_1 & +x_2 & +x_3 & +y_1 & +y_2 & \geq 3 & & (c) \\ & & & & y_1 & +y_2 & \geq 1 & & (d) \\ & x \in \{0,1\}^3,\ y \in \{0,1\}^2 & & & & & & & . \end{array} \tag{28}$$

Initially the master problem can be taken to be

$$\begin{array}{ll} \min & z \\ \text{s.t.} & z \geq 10y_1 + 20y_2 \\ & y_1 + y_2 \geq 1, \end{array} \tag{29}$$

which has optimal solution $(\overline{z}, \overline{y}) = (10, 1, 0)$. The resulting subproblem is (24), where the constant 10 added to the objective function reflects the term $f(\overline{y}) = 10\overline{y}_1 + 20\overline{y}_2$. Recall that Fig. 9 displays a search tree that solves the

problem, and Fig. 10 constructs a resolution proof of the dual solution $z \geq 18$ from this tree using the algorithm of Fig. 11.

The proof in Fig. 11 provides the information necessary to construct a Benders cut. The idea is to determine at each leaf node which values of y are necessary to carry out the deduction shown. Consider for instance the deduction of the resolvent $\neg x_1 \vee \neg x_2$ at node 4. Because one parent of the resolvent is x_3 (node 2), y must have values such that constraint (b) still implies x_3. For this it suffices that $y_1 + 2y_2 \leq 3$, because this ensures that the right-hand side of constraint (b) is at least 2, which in turn implies $x_3 = 1$. The resolution at node 6 requires the same condition on y. To obtain the resolvent $\neg x_1$ at node 2, $2y_1 + 3y_2 \leq 2$ is needed to ensure that the right-hand side of constraint (a) is at least 3, which is necessary if it is to imply x_2. It is the same at node 3.

So a proof of the bound $10y_1 + 20y_2 + 8$ can be carried out as long as $y_1 + 2y_2 \leq 3$ and $2y_1 + 3y_2 \leq 2$. This yields the Benders cut,

$$z \geq \beta_{\overline{y}}(y) = 10y_1 + 20y_2 + 8v(\{y_1 + 2y_2 \leq 3,\ 2y_1 + 3y_2 \leq 2\}) \tag{30}$$

where v evaluates to 1 if both of the inequalities hold. Note that $\beta_{\overline{y}}(\overline{y}) = \beta^* = 18$, as desired. In this case the two inequalities are equivalent to $y_2 = 0$, and the cut can be written

$$z \geq 10y_1 + 20y_2 + 8(1 - y_2). \tag{31}$$

The Benders cuts are in general nonlinear, however. This is no impediment of the master problem is solved purely by branching. The cuts also can be linearized if desired, as is noted below.

The constraint (30) is added to the master problem (29), which can be solved by updating the search tree generated for the original master problem. The solution is $(\overline{z}, \overline{y}) = (18, 1, 0)$, with which the algorithm terminates.

In general a Benders cut is generated as follows. Let

$$ax + b\overline{y} \geq \alpha \tag{32}$$

be the violated inequality and $C_P = \bigvee_{j \in J^+} x_j \vee \bigvee_{j \in J^-} \neg x_j$ the falsified clause associated with each leaf node P in the algorithm of Fig. 11. Associate with P the inequality I_P given by

$$by \leq \alpha - 1 - \sum_{j \in J^+} a_j - \sum_{j \in J'} a_j^+, \tag{33}$$

where $J' = \{1, \ldots, n\} \setminus (J^+ \cup J^-)$, and $a_j^+ = \max\{0, a_j\}$. Then if β^* is the optimal value of the subproblem, the Benders cut is,

$$z \geq \beta_{\overline{y}}(y) = f(y) + \sum_j c_j^- + (\beta^* - f(\overline{y}) - \sum_j c_j^-)v(I), \tag{34}$$

where I is the set of inequalities I_P for all leaf nodes P, and $c_j^- = \min\{0, c_j\}$. The function $v(I)$ evaluates to one if y satisfies all the inequalities in I and to zero otherwise.

Theorem 14 *The algorithm of Fig. 12 terminates with the optimal value of (25) if (25) is feasible, and otherwise it terminates with an unsatisfiable master problem.*

Proof. Because D_y is finite, by Theorems 5 and 6 it suffices to show that $\beta_{\overline{y}}$ has properties (B1) ad (B2).

(B2) $\beta_{\overline{y}}(\overline{y}) = \beta^*$. This is true if $v(I) = 1$ when $y = \overline{y}$; i.e., if $y = \overline{y}$ satisfies all the inequalities (33) in I. Because (32) implies C_P, any x that falsifies C_P must violate (32). This is true in particular of x' given by

$$x'_j = \begin{cases} 0 & \text{if } j \in J^+, \text{ or } j \in J' \text{ and } a_j \geq 0 \\ 1 & \text{if } j \in J^-, \text{ or } j \in J' \text{ and } a_j < 0. \end{cases} \tag{35}$$

By substituting $x = x'$ in $ax + b\overline{y} \leq \alpha - 1$, one obtains (33).

(B1) The cut (34) is valid for all y. It is clearly valid when $v(I) = 0$. It therefore suffices to show that $z \geq \beta^*$ when $v(I) = 1$; i.e., when y satisfies all the inequalities in I. But $z \geq \beta^*$ if $ax + by \geq \alpha$ implies C_P at each leaf P, which is equivalent to saying that any x that falsifies C_P violates $ax + by \geq \alpha$. Because y satisfies the inequalities in $v(I)$, in particular it satisfies I_P, given by (33). (33) can be written $by \leq \alpha - 1 - ax'$, where x' is defined in (35). So (x', y) violates $ax + by \geq \alpha$. Clearly any x that falsifies C_P has the property that $ax \leq ax'$. So any x that falsifies C_P violates $ax + by \geq \alpha$. □

In practice it may be advantageous to replace the set I of inequalities in the Benders cut (34) with a simpler set that implies I. The resulting cut is still a Benders cut provided $\beta_{\overline{y}}(\overline{y}) = \beta^*$, because the cut clearly remains valid. For instance, if the subproblem is solved in a way that does not readily yield a set of resolutions but still identifies an infeasible subset of inequalities, a weaker Benders cut can be constructed. To return to the example (28), suppose that solution of the subproblem reveals only that inequalities (a) and (b) are jointly infeasible but does not produce a tree such as Fig. 9. These inequalities are

$$\begin{aligned} 2x_1 + 5x_2 - x_3 \geq 5 - 2\overline{y}_1 - 3\overline{y}_2 = 3 \quad (a) \\ -x_1 + x_2 + 4x_3 \geq 5 - \overline{y}_1 - 2\overline{y}_2 = 4 \quad (b). \end{aligned}$$

They remain valid as long as $2y_1 - 3y_2 \leq 2$ and $y_1 - 2y_2 \leq 3$. This produces the Benders cut,

$$z \geq \beta_{\overline{y}}(y) = 10y_1 + 20y_2 + 8v(\{2y_1 + 3y_2 \leq 2,\ y_1 + 2y_2 \leq 1\}).$$

Note that the inequalities in the argument of v differ from those in the previous cut (30), although in this case the two cuts are satisfied by the same 0-1 points. Normally the use of resolvents, as in (30), obtains stronger cuts.

Choose an initial $\overline{y} \in \{0,1\}^n$ in problem (25).
Set $\overline{z} = -\infty$ and $k = 0$.
While the subproblem (26) has optimal value $\beta^* > \overline{z}$:
  For each leaf P of the search tree used to solve (26):
    If $Ax \geq a - B\overline{y}$ is satisfied at P **then**
      Let (32) be the inequality $cx + f(\overline{y}) \leq \beta - 1$, where
        β is the value of the objective function at P.
    Else
      Let (32) be an inequality among $Ax \geq a - B\overline{y}$ that is
        violated at P.
      Identify a clause C_P that is implied by (32) and
        false at P.
      Let I_P be the inequality (33).
  Let I be $\{I_P \mid \text{all leaves } P\}$.
  Let $k = k + 1$, let $y^k = \overline{y}$, and add the Benders cut (34)
    to the master problem (9).
  If (35) is infeasible **then**
    Stop; (25) is infeasible.
  Else
    Let $(\overline{z}, \overline{y})$ be an optimal solution of (9).
The optimal value of (25) is $\overline{z}$.

Figure 12: *A Benders algorithm for 0-1 programming.*

In general, the solution of the subproblem reveals an infeasible subset I' of inequalities and an optimal value β^*. For each inequality $ax + b\overline{y} \geq \alpha$ in I', let I contain the inequality $by \leq b\overline{y}$. Because the resulting cut (34) is clearly valid and satisfies $\beta_{\overline{y}}(\overline{y}) = \beta^*$, it is a Benders cut.

The Benders cut (30) can also be linearized, which again tends to weaken it. This is done simply by identifying a subset $J(\overline{y}) \subset \{1, \ldots, n\}$ for which setting $y_j = \overline{y}_j$ for each $j \in J(\overline{y})$ satisfies all the inequalities in I. The linearized Benders cut is,

$$z \geq \beta_{\overline{y}}(y) = f(y) + \sum_j c_j^- + (\beta^* - f(\overline{y}) - \sum_j c_j^-) \left(1 - \sum_{j \in J(\overline{y})} [(1 - \overline{y}_j) y_j + \overline{y}_j (1 - y_j)] \right).$$

The linear cut 31 derives from $J(\overline{y}) = \{2\}$.

9 Conclusion

An elementary theory of logic-based Benders decomposition has been developed and applied to two specific problems, namely the propositional satisfiability problem and the 0-1 programming problem, producing what appear to be new algorithms for both.

A similar approach can in principle be applied to any optimization or feasibility problem, because inference duality is defined for any such problem. Its success depends on the extent to which

- a tractable subproblem can be obtained by a judicious partitioning of the variables, and
- a proof of a lower bound in the subproblem dual for $y = \overline{y}$ yields a useful Benders cut for other values of y.

Logic-based Benders decomposition may also have potential in a branch-and-cut context, as mentioned in connection with the satisfiability problem above. Cuts presently used typically involve variables that have not yet been fixed in the enumeration tree and serve primarily to strengthen a linear or some other relaxation. They may or may not be global (i.e., apply throughout the tree). Logic-based Benders cuts, by contrast, would involve variables that are already fixed and would always be global. In fact their function would be precisely to prune other branches of the tree, and their ability to do so would be unrelated to the use of a relaxation.

Benders decomposition is only one area in which solution strategies in highly structured contexts can be generalized by finding analogous structure in a logic-based approach. Some other areas are surveyed in [12].

References

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

[2] Barth, P., Logic-based 0-1 constraint solving in constraint logic programming, dissertation, Universität des Saarlandes, Saarbrücken, Germany, 1995.

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

[4] Chandru, V., C. R. Coullard, P. L. Hammer, M. Montañez, and X. Sun, On renamable Horn and generalized Horn functions, *Annals of Mathematics and AI* **1** (1990) 33-48.

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

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

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

[8] Hooker, J. N., Resolution vs. cutting plane solution of inference problems: Some computational experience, *Operations Research Letters* **7** (1988) 1-7.

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

[10] Hooker, J. N., Generalized resolution for 0-1 linear inequalities, *Annals of Mathematics and Artificial Intelligence* **6** (1992) 271-286.

[11] Hooker, J. N., Solving the incremental satisfiability problem, *Journal of Logic Programming* **15** (1993) 177-186.

[12] Hooker, J. N., Logic-based methods for optimization, in A. Borning, ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science* **874** (1994) 336-349.

[13] Hooker, J. N. and C. Fedjki, Branch-and-cut solution of inference problems in propositional logic, *Annals of Mathematics and AI* **1** (1990) 123-139.

[14] Hooker, J. N., and Hong Yan, 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.

[15] Jeroslow, R. G. and J. Wang, Dynamic programming, integral polyhedra and Horn clause knowledge base, *ORSA Journal on Computing* **4** (1989) 7-19.

[16] Jiang, Y., T. Richards and B. Richards, No-good backmarking with min-conflict repair in constraint satisfaction and optimization, in A. Borning, ed., *Principles and Practice of Constraint Programming*, *Lecture Notes in Computer Science* **874** (1994) 21-39.

[17] Mitterreiter, I., and F. J. Radermacher, Experiments on the running time behavior of some algorithms solving propositional logic problems, manuscript, Forschungsinstitut für anwendungsorientierte Wissensverarbeitung, Ulm, Germany, 1991

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

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

[20] Robinson, J. A., A machine-oriented logic based on the resolution principle, *Journal of the ACM* **12** (1965) 23-41.

[21] Trick, M., and D. S. Johnson, eds., *Second DIMACS Challenge: Cliques, Coloring and Satisfiability*, Series in Discrete Mathematics and Theoretical Computer Science, American Mathematical Society (1995).

[22] Tsang, E., *Foundations of Constraint Satisfaction* (London, Academic Press) 1993.