ELSEVIER

Discrete Applied Mathematics 96–97 (1999) 395–442

DISCRETE APPLIED MATHEMATICS

Mixed logical-linear programming ☆

J.N. Hooker[a,*], M.A. Osorio[b]

[a] *Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213, USA*
[b] *School of Computer Science, Autonomous University of Puebla, Puebla, Mexico 72550*

Abstract

Mixed logical/linear programming (MLLP) is an extension of mixed integer/linear programming (MILP). It can represent the discrete elements of a problem with logical propositions and provides a more natural modeling framework than MILP. It can also have computational advantages, partly because it eliminates integer variables when they serve no purpose, provides alternatives to the traditional continuous relaxation, and applies logic processing algorithms. This paper surveys previous work and attempts to organize ideas associated with MLLP, some old and some new, into a coherent framework. It articulates potential advantages of MLLP's wider choice of modeling and solution options and illustrates some of them with computational experiments.

1. Introduction

Mixed logical/linear programming (MLLP) is a general approach to formulating and solving optimization problems that have both discrete and continuous elements. It extends mixed integer/linear programming (MILP) by introducing logic-based modeling and solution options. MLLP in no way rejects integer programming and in fact incorporates all of its techniques. Its expanded modeling framework may, however, allow more natural or succinct formulations without sacrificing solution efficiency. Its larger repertory of solution techniques may accelerate solution or even solve problems that are intractable for MILP alone. These techniques include branching strategies, relaxations and logic processing algorithms that are not ordinarily associated with integer programming.

Mixed discrete/continuous problems are traditionally conceived as continuous problems in which some of the variables are restricted to be integers. MLLP permits one to take a different view. Rather than embed the discrete aspects of the problem within

☆ This research is partially supported by the 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 (USA), under grant EEC-8943164.

* Corresponding author.

0166-218X/99/$ - see front matter
PII: S0166-218X(99)00100-6

a linear programming model, which may not be the most natural approach, one can represent them with logical formulas. MLLP therefore has the option of dispensing with integer variables. Rather than require that a feasible solution satisfy a fixed set of inequalities, an MLLP model can contain several alternative sets of inequalities. The logical formulas govern which sets must be satisfied by a feasible solution.

1.1. General form of an MLLP

An introductory discussion is more meaningful if MLLP is given a brief mathematical description. An MLLP model has the form

$$\begin{aligned} &\min \quad cx \\ &\text{s.t.} \quad p_j(y,h) \rightarrow (A^j x \geqslant a^j),\ j \in J \mid q_i(y,h),\ i \in I. \end{aligned} \tag{1}$$

The constraint set has a logical part (on the right-hand side of the bar) and a continuous part (on the left).

The logical part consists of formulas $q_i(y,h)$ that involve *atomic propositions* $y=(y_1,\ldots,y_n)$, which are either true or false. Such a formula might be $q_1(y,h)=y_1 \vee y_2$, which says that y_1 or y_2 (or both) must be true. There may also be some variables $h=(h_1,\ldots,h_m)$ that take several discrete values. Thus $q_i(y,h)$ could be $(y_1 \vee y_2) \wedge (h_1 \neq h_2)$, where $\wedge$ means 'and'. In general, the formulas p_j and q_i may take any form that is convenient for the purpose at hand, provided that their truth value is a function of the truth values of the propositions y and the values of the discrete variables h.

The continuous part associates logical formulas $p_j(y,h)$ with systems $A^j x \geqslant a^j$ of linear inequalities. A system $A^j x \geqslant a^j$ is enforced when $p_j(y,h)$ is true. So the constraints of the following problem in effect require x to satisfy $A^1 x \geqslant a^1$ or $A^2 x \geqslant a^2$ (or both):

$$\begin{aligned} &\min \quad cx \\ &\text{s.t.} \quad \left.\begin{array}{l} y_1 \rightarrow (A^1 x \geqslant a^1) \\ y_2 \rightarrow (A^2 x \geqslant a^2) \end{array}\right| \ y_1 \vee y_2 \end{aligned}$$

In general, (x,y,h) is feasible if (y,h) makes all the logical formulas $q_i(y,h)$ true, and x satisfies the linear systems corresponding to the formulas $p_j(y,h)$ that (y,h) makes true.

1.2. Solution of an MLLP

Problem (1) can be solved by branching on the truth values of the y_j's and the discrete values of the h_j's. At each node of the search tree, one solves a linear programming problem (LP) containing the constraints that correspond to true p_j's, plus any inequalities added to strengthen the relaxation. A key element of MLLP is to apply a logical inference algorithm to the logical formulas before solving the LP. This

may generate *valid constraints* (constraints satisfied by all feasible solutions) in logical form, and in particular it may fix some additional y_j's and h_j's.

An MLLP can therefore be solved in a manner that is analogous to the traditional branch-and-cut algorithms used in MILP. There are two primary differences, however. First, as one descends into the tree, the LPs solved at the nodes are not necessarily defined by fixing certain variables in them. They may also be defined by adding new constraints corresponding to formulas that fixed variables make true, or by some combination of the two methods.

A second difference is that at each node of the search tree, the logical part of the constraint set can be processed with its own set of algorithms, in order to generate additional constraints or check for feasibility. These include many of the logic programming and constraint satisfaction techniques that appear in the computer science and artificial intelligence literatures (discussed below). MLLP therefore provides one means of uniting mathematical programming with methods have been developed more or less independently in other fields.

1.3. Motivation for MLLP

The primary rationale for MLLP is that it brings to mathematical programming greater modeling power and a wider range of solution options. But MLLP also grows out of a rethinking of the role of integer variables.

Traditionally integer variables have in most cases served a modeling function and a relaxation function simultaneously. It is proposed here that these functions be separated. When integer variables provide the most natural modeling device for certain constraints, e.g. knapsack constraints, they should be used to formulate those constraints. When a certain portion of the constraint set has a useful continuous relaxation when formulated with integer variables, they should be included in that portion of the problem in order to obtain the relaxation.

In other cases, however, inequalities may not provide the most convenient way to formulate the discrete aspect of the problem. Also their continuous relaxation may be weak, or its effect may be duplicated by adding a few valid inequalities that involve only the original continuous variables. Furthermore, it will be seen that integer variables may have fractional values in the continuous relaxation even when a feasible solution of the original problem has been found. Thus if one branches on integer variables with fractional values, branching may continue unnecessarily.

In such cases, integer modeling may not justify the overhead it incurs. The inclusion of integer variables enlarges the linear programming problems that must be solved at nodes of the search tree. This can be particularly costly when there are many discrete variables, because it may be possible to process the discrete elements of the constraint set much more rapidly in logical form. A simple constraint propagation algorithm, for example, may have the same ability to detect infeasibility in logical constraints as solving the linear relaxation of their inequality formulation. But its speed may be two or three orders of magnitude greater, because it need not carry along the data

structures and machinery of a linear solver. Other types of logic processing may obtain valid constraints or fix variables in ways that are not available in MILP.

The primary drawback of MLLP is that it requires more expertise on the part of the user. It provides more options but presupposes that the user knows how to choose the best one. In particular, if integer variables are not used, then the traditional continuous relaxation is unavailable, and it may be necessary to concoct an alternate relaxation.

1.4. Aim of this paper

The aim here is to explore MLLP as a general and practical approach to solving problems with both discrete and continuous elements. Previous work is drawn together, and an attempt is made to order ideas associated with MLLP, some old and some new, in a coherent frame-work. The potential advantages of an expanded repertory of modeling and solution options are articulated, and several are illustrated by computational experiments. The logic processing component of MLLP is explored only deeply enough to convey the flavor of the ideas, but some expository literature is cited.

Because MLLP is a general approach to continuous/discrete problem solving, a thorough-going experimental evaluation would be a massive undertaking, and it is not attempted here. The task would be further complicated, both practically and conceptually, by the fact that MLLP is not a single approach to problem solving but a framework within which several approaches can be used. As in MILP, its effectiveness depends on how carefully one designs relaxations and branching schemes to fit the problem at hand. The intent here is to provide a broader range of options and to show by example that at least some of them can be superior to the conventional ones.

The examples include chemical engineering network synthesis problems, warehouse location problems, flow shop scheduling problems, and the "progressive party problem", which is a scheduling problem posed by a yacht party. The last problem is rather frivolous but has attracted a good deal of attention and illustrates several ideas associated with MLLP.

Experience with engineering design problems (e.g., [11,66]) suggests that MLLP can be usefully extended to mixed logical/nonlinear programming (MLNLP). This possibility is not pursued here.

1.5. Previous work

A logic-based approach to operations research was discussed as early as 1968 in Hammer and Rudeanu's treatise on boolean methods [26]. Granot and Hammer [24] suggested in 1971 the possibility of using boolean methods for integer programming.

The MLLP approach described here was perhaps first clearly articulated by Jeroslow [41,42], who was primarily interested in issues of representability. He viewed discrete variables as artifices for representing a feasible subset of continuous space, which in the case of an MLLP or MILP model is a union of finitely many polyhedra. From this it follows that MLLP and MILP models are essentially disjunctive programming

models. Building on joint work with Lowe [43], Jeroslow proved that an MILP model can represent a union of finitely many polyhedra if and only if they have the same recession cone.

In the meantime, Williams [68–70,72], Blair [9,10] and Hooker [27–32] explored connections between logic and optimization. Beaumont [7] undertook what is apparently the first systematic study of MLLP as a solution technique for optimization problems. Drawing on the seminal work of Balas in disjunctive programming [2–4], he described families of valid inequalities that can be used to create relaxations of disjunctive constraints.

More recently, Hooker argued in [33] that a logic-based approach to optimization, including MLLP, can exploit problem structure in ways that are parallel to traditional polyhedral techniques. Wilson [73–75] studied logic-based formulations.

It is crucial to demonstrate the practical value of MLLP in a problem domain. This was accomplished largely by Grossmann in the area of chemical process design in a series of papers coauthored with Hooker, Turkay, Yan and particularly Raman [38,51–54,66]. These papers developed some of the key MLLP concepts discussed here. Bollapragada, Ghattas and Hooker also obtained encouraging results in structural design [11].

1.6. Other approaches

It is instructive to contrast MLLP with other approaches that combine discrete and continuous elements.

The mixed logical/linear programming approach of McAloon and Tretkoff [45,46], which is implemented in the system 2LP, combines procedural with declarative programming. The discrete element is represented by a user-supplied script that controls the formulation and solution of LP models that represent the continuous element. This contrasts with the approach to MLLP described here, in which both elements are modeled in a declarative fashion. The two approaches are not incompatible, however, and 2LP could in fact provide a framework in which to implement the MLLP techniques presented here.

Even pure 0–1 optimization problems have a continuous element in the sense that the constraints are represented by linear inequalities, and it is not obvious how to apply logic-based methods to them. An approach devised by Barth [6] is to derive formulas from the inequalities that can be processed with logical inference methods. Barth's techniques can enhance the logical processing phase of MLLP algorithms.

The work of McAloon, Tretkoff and Barth is influenced by several streams of research that have historically focused on discrete problems but are experimenting with ways to incorporate continuous variables. Logic programming models, introduced by Colmerauer [18] and Kowalski [44], allow one to formulate a problem in a subset of first-order logic (Horn clause logic). Recent versions of the logic programming language PROLOG [12,63], such as PROLOG III [17] (and soon IV), incorporate linear programming.

The integration of constraint solving with logic programming is formalized in the *constraint logic programming* (CLP) scheme of Jaffar and Lassez [39]. It generalizes the "unification" step of logical inference methods to encompass constraint solving in general [40].

CLP provides a framework for integrating constraint satisfaction methods developed in the artificial intelligence community (and elsewhere) with logic programming ideas [21,65,67]. A number of systems along this line have been developed in addition to Prolog III, including CLP(R) [39], CAL [1], CHIP [20,61], the ILOG solver [48], and other packages [13,55,59]. Linear programming has a place in several of these systems. Unlike MLLP, these methods rely to some extent on procedural modeling. They also lack MLLP's emphasis on exploiting problem structure in the generation of valid constraints and relaxations, although the constraint programming literature has shown some interest in exploiting structure (e.g., [22]).

1.7. Outline of the paper

The remainder of the paper begins with a few simple modeling examples (Section 2). Two long sections (3 and 4) respectively discuss relaxations and logic processing algorithms. Section 5 provides a generic algorithm for solving MLLPs, and Section 6 presents models and computational results for four sets of problems. The concluding section attempts to assemble guidelines for modeling and solving problems in an MLLP framework.

Aside from its survey and development of MLLP generally, the specific contributions of this paper include necessary and sufficient conditions for whether an elementary inequality for a disjunction is supporting (Section 3.4), necessary and sufficient conditions for integrality of a 0–1 disjunctive representation (Section 3.5), a definition of optimal separating inequalities (Section 3.7), a completeness proof for multivalent resolution (Section 4.1), and a unit resolution algorithm for multivalent clauses (Section 4.2).

2. Modeling examples

A few simple examples will illustrate modeling in MLLP.

2.1. Fixed charges and semicontinuous variables

A cost function with a fixed charge is generally given a big-M formulation in integer programming,

$$\begin{aligned} \min \quad & cx + dy \\ \text{s.t.} \quad & x \leqslant My, \\ & x \geqslant 0, \\ & y \in \{0,1\}, \end{aligned}$$

where c is the variable cost and d the fixed cost. (Other constraints and objective function coefficients would normally be present.) An MLLP model is,

$$\begin{array}{lll|l} \min & cx+z & & \\ \text{s.t.} & y \rightarrow (z=d) & & \\ & y' \rightarrow (x=0) & & y \vee y' \\ & x, z \geqslant 0. & & \end{array}$$

The proposition $y \vee y'$ states that either the fixed cost is incurred or it is not. The model can also be written by replacing y' with $\neg y$ (not-y) and deleting $y \vee y'$.

A semicontinuous variable x is one whose value must lie in one of the intervals $[a_t, b_t]$ for $t = 0, \ldots, T$. One MILP representation is,

$$\begin{aligned} & a_t y_t \leqslant x_t \leqslant b_t y_t, \quad t = 0, \ldots, T \\ & \sum_{t=0}^{T} x_t = x, \quad y_t \in \{0,1\},\ t = 0, \ldots, T. \end{aligned} \tag{2}$$

An MLLP representation is

$$y_t \rightarrow (a_t \leqslant x \leqslant b_t) \quad \Big| \quad \bigvee_{t=0}^{T} y_t \ .$$

2.2. Quadratic assignment problem

The quadratic assignment problem is typically formulated as an MILP model in the following way:

$$\begin{aligned} \min \quad & \sum_{ikjl} v_{ij} c_{kl} w_{ijkl} \\ \text{s.t.} \quad & \sum_{i} y_{ik} = 1 \quad \text{all } k, \\ & \sum_{k} y_{ik} = 1 \quad \text{all } i, \\ & z_{ijkl} \geqslant y_{ik} + y_{jl} - 1, \quad \text{all } i, j, k, l, \\ & y_{ik}, w_{ijkl} \in \{0,1\}, \quad \text{all } i, j, k, l. \end{aligned}$$

Here v_{ij} is the volume of traffic between facilities i and j, and c_{kl} is the unit cost of traffic between locations k and l. $y_{ik} = 1$ if facility i is assigned to location k, and $z_{ijkl} = 1$ if facilities i, j are respectively assigned to locations k, l.

An MLLP model can be written with fewer variables.

$$\begin{aligned} \min \quad & \sum_{ij} z_{ij} \\ \text{s.t.} \quad & (y_{ik} \wedge y_{jl}) \rightarrow (z_{ij} = v_{ij} c_{kl}), \text{ all } i,j,k,l \\ & z_{ij} \geqslant 0, \quad \text{all } i,j. \end{aligned} \quad \left| \begin{array}{l} \displaystyle\sum_i y_{ik} = 1, \quad \text{all } k \\ \displaystyle\sum_k y_{ik} = 1, \quad \text{all } i \end{array} \right. \tag{3}$$

The constraints on the right are intended to be read as logical constraints. The first constraint, for example, says that exactly one of the propositions $y_{1k}, \ldots, y_{nk}$ is true, for each k. The symbol $\wedge$ on the left means 'and'.

An alternate model uses multivalued discrete variables h_i to indicate which location is assigned to facility i:

$$\begin{aligned} \min \quad & \sum_{ij} z_{ij} \\ \text{s.t.} \quad & (h_i = k \wedge h_j = l) \rightarrow (z_{ij} = v_{ij} c_{kl}), \text{ all } i,j,k,l \\ & z_{ij} \geqslant 0, \quad \text{all } i,j. \end{aligned} \quad \left| \begin{array}{l} \text{alldiff}(h_1, \ldots, h_n) \\ h_i \in \{1, \ldots, n\}, \quad \text{all } i. \end{array} \right.$$

The "alldiff" constraint on the right states that $h_1, \ldots, h_n$ must all take distinct values. All-different constraints are widely used in constraint programming.

3. Relaxations

The linear programming problem solved at each node of an MLLP search tree provides a lower bound on the optimal value at that node. However, the LP contains only those constraints that are enforced by true propositions. Many logical constraints may therefore be unrepresented in the LP relaxation, which may therefore provide a weak bound. When possible it is important to augment the relaxation with additional valid inequalities that represent logical formulas.

This section presents some techniques for obtaining linear relaxations of logical formulas by generating valid inequalities in the continuous variables. We will consider only disjunctive formulas in which each disjunct is an atomic proposition that enforces a linear system:

$$y_j \rightarrow (A^j x \geqslant a^j) \mid y_1 \vee \cdots \vee y_m. \tag{4}$$

An important research question is how relaxations may be written for broader classes of formulas, particularly formulas that contain multivalued discrete variables h_j. This matter is being investigated.

Some of the valid inequalities that will be presented for disjunctions mimic the effect of the traditional continuous relaxation of a 0–1 model. The strength and nature of the traditional relaxation is remarkably ill understood, given the degree to which it is used. An analysis of it will therefore comprise an important part of the discussion.

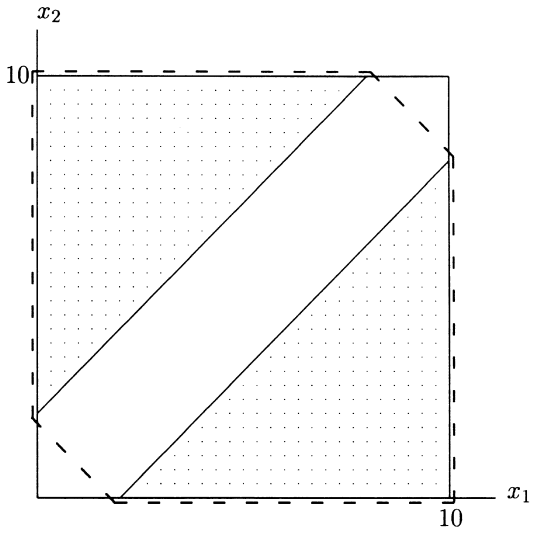

Fig. 1. Convex hull of the feasible set of a scheduling disjunction.

3.1. The convex hull

The task at hand is to generate valid inequalities for (4), which can be written

$$\bigvee_{t\in T} A^t x \geqslant a^t. \tag{5}$$

The feasible set is a union of $|T|$ polyhedra, and a description of the convex hull of this union is the best possible linear relaxation of the formula.

In some cases the convex hull is so large that even the best possible relaxation is poor or useless. If for example x is bounded $0 \leqslant x \leqslant m$, it is not uncommon for the convex hull of (5) to fill most or all of the box described by $0 \leqslant x \leqslant m$. A notorious example of this arises in scheduling problems. If operations 1 and 2 begin at times x_1 and x_2 and last 2 min, one imposes the disjunctive constraint

$$(x_2 \geqslant x_1 + 2) \vee (x_1 \geqslant x_2 + 2)$$

to ensure that one occurs after the other. The upper bounds m represent the latest time at which an operation could be scheduled and therefore may be much larger than 2. The dashed line in Fig. 1 encloses the convex hull when $m = (10, 10)$. In this case the best possible relaxation is given by $x_1 + x_2 \geqslant 2$, $x_1 + x_2 \leqslant 18$ and $0 \leqslant x_j \leqslant 10$. This is not much different than $0 \leqslant x_j \leqslant 10$ and is probably useless in practice.

3.2. Disjunctive and dual inequalities

A relaxation of (5) can be obtained by generating valid inequalities that partially or completely describe the convex hull. Balas [4] characterized valid inequalities for (5)

as follows. First, note that $bx \geqslant \beta$ is a valid inequality for a feasible disjunct $A^t x \geqslant a^t$ if and only if it is dominated by a nonnegative linear combination (or *surrogate*) of $A^t x \geqslant a^t$. A dominating surrogate can be written $uAx \geqslant ua$, where $b \geqslant uA, \beta \leqslant ua$ and $u \geqslant 0$. But $bx \geqslant \beta$ is a valid inequality for the disjunction as a whole if it is valid for each disjunct; i.e., for each disjunct a surrogate can be found that dominates $bx \geqslant \beta$.

Theorem 1 (Balas [4]). *The inequality $bx \geqslant \beta$ is valid for* (5) *if any only if for each feasible system $A^t x \geqslant a^t$ there is a $u^t \geqslant 0$ such that $b \geqslant u^t A^t$ and $\beta \leqslant u^t a^t$.*

Given any set of surrogates $u^t A^t x \geqslant u^t a^t$, if $x \geqslant 0$ one can immediately write the valid disjunctive inequality

$$\left(\max_{t \in T} \{u^t A^t\} \right) x \geqslant \min_{t \in T} \{u^t a^t\} \tag{6}$$

for (5), where the maximum is componentwise. Theorem 1 clearly implies that if $x \geqslant 0$, every valid inequality is dominated by a disjunctive inequality (6). The strength and usefulness of a disjunctive inequality (6) depends radically on the choice of surrogates. One could in principle generate disjunctive inequalities to define every facet of the convex hull, but this is often impractical. The task of obtaining a good relaxation for (5) is in essence the task of choosing multipliers u^t judiciously.

One initially attractive choice for u^t is given by the solution of a dual problem. Each surrogate should ideally give the best possible bound on the objective function cx. That is, u^t should be chosen so that the minimum value of cx subject to $u^t A^t x \geqslant u^t a^t$ is maximized. The desired u^t is easily seen to be the optimal solution of the LP dual of $\min\{cx \mid A^t x \geqslant a^t\}$, where u^t is the vector of dual variables. (To put it differently, the surrogate dual for linear programming is identical to the LP dual [23].)

The difficulty with this approach is that because $A^t x \geqslant a^t$ is only a small part of the original constraint set, it may have no coupling with the objective function. That is, the variables x_j that have nonzero coefficients in cx may have zero coefficients in $A^t x \geqslant a^t$, and vice-versa. This means that cx provides no information to guide the choice of u^t, a situation that is in fact common in practice.

A possible remedy is to include more constraints in the problem whose dual is solved, so as to capture the link between cx and $A^t x \geqslant a^t$. This can be done as follows. At any node of the search tree a system $Ax \geqslant a$ of certain linear constraints are enforced by true formulas $p_i(y,h)$. If $Ax \geqslant a$ is included in each term of the disjunction (5), it becomes

$$\bigvee_{t \in T} \begin{pmatrix} A^t x \geqslant a^t \\ Ax \geqslant a \end{pmatrix}.$$

For each t one solves the dual of

$$\begin{aligned} \min \quad & cx \\ \text{s.t.} \quad & A^t x \geqslant a^t \ (u^t), \\ & Ax \geqslant a \ (u), \end{aligned} \tag{7}$$

where (u^t, u) are the dual variables as shown. An optimal solution of the dual supplies a reasonable set of multipliers u^t for the disjunctive inequality (6).

Unfortunately, this approach appears to be impractical, because (7) is generally a large LP. Computational results reported in Section 6.2 suggest that it is very time-consuming to solve the dual of (7) for each disjunct. The remaining discussion will therefore focus on much faster mechanisms for choosing effective multipliers u^t.

3.3. Elementary inequalities

The most common sort of disjunctive constraint (5) is one in which each disjunct is a single inequality,

$$\bigvee_{t\in T} a^t x \geqslant \alpha^t, \tag{8}$$

where it is assumed that $0 \leqslant x \leqslant m$. Beaumont [7] showed how to generate a valid inequality for (8) that is equivalent to the continuous relaxation of the traditional 0–1 formulation of (8). The latter is

$$\begin{aligned} &a^t x \geqslant \alpha^t - M_t(1 - y_t), \quad t \in T, \\ &\sum_{t\in T} y_t = 1, \\ &0 \leqslant x \leqslant m, \\ &y_t \in \{0,1\}, \quad t \in T. \end{aligned} \tag{9}$$

Each M_t is chosen so that $\alpha_t - M_t$ is a lower bound on the value of $a^t x$:

$$\alpha_t - M_t = \sum_j \min\{0, a_j^t\} m_j. \tag{10}$$

The bounds $0 \leqslant x \leqslant m$ are imposed to ensure that such a lower bound exists. It can be assumed without loss of generality that $M_t > 0$, because otherwise the inequality is vacuous and can be dropped. Beaumont obtains a valid inequality by taking a linear combination of the inequalities in (9), where each inequality t receives weight $1/M_t$. This yields an *elementary inequality* for (8),

$$\left(\sum_{t\in T} \frac{a^t}{M_t}\right) x \geqslant \sum_{t\in T} \frac{\alpha_t}{M_t} - |T| + 1. \tag{11}$$

Theorem 2 (Beaumont [7]). *The elementary inequality* (11) *is equivalent to the continuous relaxation of* (9). *That is, the feasible set of* (11) *and* $0 \leqslant x \leqslant m$ *is equal to the projection of the feasible set of the continuous relaxation of* (9) *onto the x-space.*

One can also prove equivalence by applying Fourier elimination to (9) in order to eliminate y. It is easy to show that (11) and $0 \leqslant x \leqslant m$ are the resulting inequalities.

A similar technique obtains elementary inequalities for all logical formulas that are expressible as knapsack constraints,

$$\begin{aligned} & dy \geqslant \delta \\ & y_t \rightarrow (a^t x \geqslant \alpha_t), \quad t \in T, \\ & 0 \leqslant x \leqslant m, \end{aligned} \tag{12}$$

where $d \geqslant 0$. The 0–1 representation of (12) is

$$\begin{aligned} & a^t x \geqslant \alpha^t - M_t(1 - y_t), \quad t \in T, \\ & 0 \leqslant x \leqslant m, \\ & dy \geqslant \delta, \\ & y_t \in \{0,1\}, \quad t \in T. \end{aligned} \tag{13}$$

A linear combination of the inequalities, using weights d_t/M_t, yields the elementary inequality,

$$\left(\sum_{t \in T} a^t \frac{d_t}{M_t}\right) x \geqslant \sum_{t \in T} \alpha_t \frac{d_t}{M_t} - \sum_{t \in T} d_t + \delta. \tag{14}$$

This is in general weaker than the continuous relaxation of (13), however. If $\sum_t d_t = \delta$, for example, (13) forces all the disjuncts to hold, where (14) only forces a linear combination of them to hold.

In many cases a better lower bound than that in (10) can be obtained for $a^t x$, resulting in a stronger inequality. One method is to minimize $a^t x$ subject to each of the other disjuncts and $0 \leqslant x \leqslant m$ and pick the smallest of the minimum values. M_t is therefore chosen so that

$$\alpha_t - M_t = \min_{t' \neq t} \left\{ \min_x \{ a^t x \mid a^{t'} x \geqslant \alpha_{t'}, \ 0 \leqslant x \leqslant m \} \right\}. \tag{15}$$

The computation involved is negligible.

Consider for example the following constraint set, whose feasible set is the shaded area in Fig. 2.

$$(x_1 + 2x_2 \geqslant 2) \vee (3x_1 + x_2 \geqslant 3), \quad 0 \leqslant x_j \leqslant 2.$$

The 0–1 formulation is

$$\begin{aligned} & x_1 + 2x_2 \geqslant 2 - M_1(1 - y_1), \\ & 3x_1 + x_2 \geqslant 3 - M_2(1 - y_2), \\ & y_1 + y_2 = 1, \\ & 0 \leqslant x_j \leqslant 2, \quad y_j \in \{0,1\}. \end{aligned}$$

Beaumont puts $(M_1, M_2) = (2,3)$ which results in the valid inequality $\frac{3}{2}x_1 + \frac{4}{3}x_2 \geqslant 1$. By contrast, (15) puts $(M_1, M_2) = (1,2)$, which yields the stronger inequality $x_1 + x_2 \geqslant 1$.

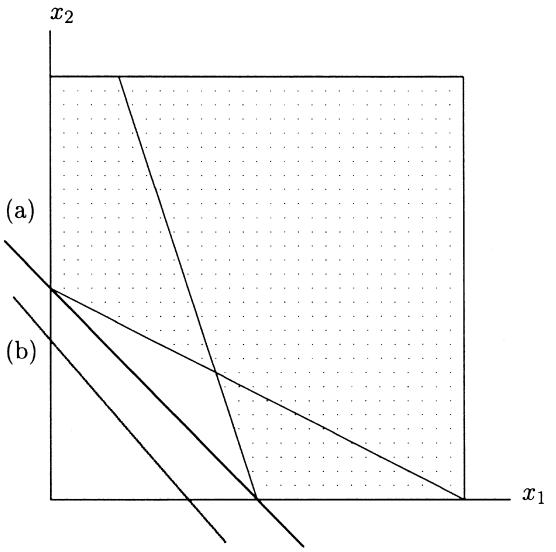

Fig. 2. Illustration of a supporting elementary inequality (a) and a nonsupporting elementary inequality (b).

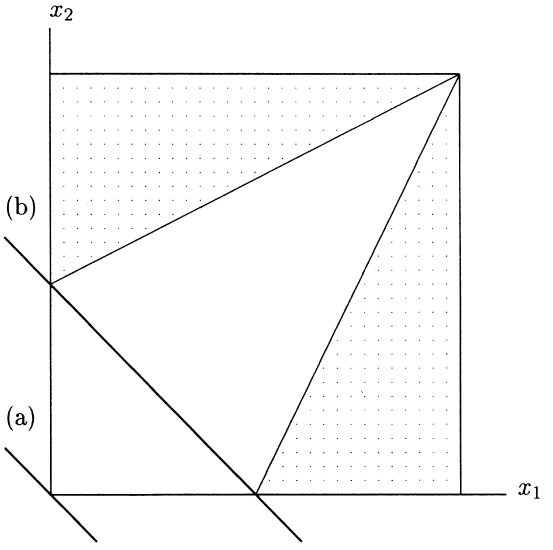

Fig. 3. Illustration of an elementary inequality (a) and a strengthened elementary inequality (b).

This is a *supporting* inequality in the sense that it defines a supporting hyperplane for the feasible set.

Even when (15) is used to compute M_i, the resulting inequality may fail to be supporting. Consider the constraints (Fig. 3),

$$(-x_1 + 2x_2 \geqslant 2) \vee (2x_1 - x_2 \geqslant 2), \quad 0 \leqslant x_j \leqslant 2.$$

Eq. (15) sets $(M_1, M_2) = (4, 4)$, which results in the useless inequality $x_1 + x_2 \geqslant 0$. The inequality can obviously be strengthened to $x_1 + x_2 \geqslant 1$.

When the inequalities $a^t x \geqslant \alpha_t$ in (12) are replaced by systems of inequalities $A^t x \geqslant a^t$, many elementary inequalities are required to achieve the effect of the traditional relaxation. Let each system $A^t x \geqslant a^t$ consist of inequalities $A^{ti} x \geqslant a_i^t$ for $i \in I_t$. The 0–1 formulation is

$$\begin{aligned} &A^t x \geqslant a^t - M^t(1 - y_t), \quad t \in T, \\ &0 \leqslant x \leqslant m, \\ &d y \geqslant \delta, \\ &y_t \in \{0, 1\}, \quad t \in T. \end{aligned} \tag{16}$$

Here M^t is an array such that for each $i \in I_t$, $a_i^t - M_i^t$ is a lower bound on $A^{ti} x$. Repeated applications of Fourier elimination reveal that the projection of the feasible set of (16) onto the x-space is described by the set of inequalities of the form,

$$\left(\sum_{t \in T} A^{ti_t} \frac{d_t}{M_{i_t}^t} \right) x \geqslant \sum_{t \in T} a_{i_t}^t \frac{d_t}{M_{i_t}^t} - \sum_{t \in T} d_t + \delta,$$

for all possible vectors $(i_1, \ldots, i_{|T|}) \in I_1 \times \cdots \times I_{|T|}$.

Elementary inequalities may therefore be impractical when the y_t's correspond to systems of inequalities. In such cases one can use optimal separating inequalities (described below) or the traditional relaxation.

3.4. Supporting elementary inequalities

The example of Fig. 3 shows that an elementary inequality can fail to be supporting. In such cases it is a simple matter to increase its right-hand side until it supports the feasible set, thus obtaining a *strengthened* elementary inequality. In fact, there is a closed-form formula for the best possible right-hand side. The formula allows one to check easily whether a given elementary inequality is supporting, and when it is not, to improve upon the traditional continuous relaxation the inequality represents.

Figures 2 and 3 may suggest that a disjunction $a^1 x \geqslant \alpha_1 \vee a^2 x \geqslant \alpha_2$ produces a supporting elementary inequality if and only if the vectors a^1, a^2 subtend an acute angle, and that a similar relationship might be discovered for more than two disjuncts. A third example reveals that the situation is more complicated than this. Fig. 4 shows the feasible set for

$$(-3x_1 + x_2 \geqslant -3) \vee (-x_2 \geqslant -1), \quad 0 \leqslant x_j \leqslant 3.$$

The elementary inequality (a) is $3x_1 + 2x_2 \leqslant 12$, which is supporting even though $(-3, 1)$ and $(0, -1)$ subtend an obtuse angle.

A more adequate analysis goes as follows. Let $bx \geqslant \beta$ be any valid inequality for the disjunction (8), such as an elementary inequality, such that the inequality defines a

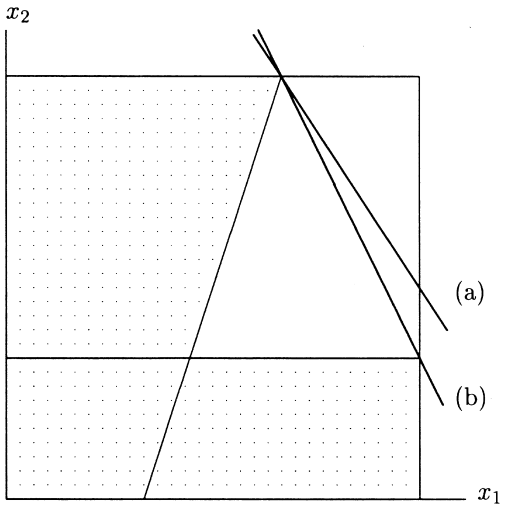

Fig. 4. A supporting elementary inequality (a) and a facet-defining inequality (b).

supporting hyperplane of the feasible set of (8). Then β is the smallest of the minimum values obtained by minimizing bx subject to each of the disjuncts $a^t x \geqslant \alpha_t$. That is,

$$\beta = \min_{t \in T} \beta_t, \tag{17}$$

where

$$\beta_t = \min\{bx \mid a^t x \geqslant \alpha_t,\ 0 \leqslant x \leqslant m\}.$$

The computation of β_t is simplified if $b \geqslant 0$, because in this case the upper bounds $x \leqslant m$ can be ignored. To this end one can introduce the change of variable,

$$\hat{x}_j = \begin{cases} x_j & \text{if } b_j \geqslant 0, \\ m_j - x_j & \text{otherwise.} \end{cases}$$

The strengthened elementary inequality in terms of $\hat{x}$, namely $\hat{b}\hat{x} \geqslant \hat{B}$, can now be computed, where $\hat{b}_j = |b_j|$. The right-hand side of $bx \geqslant \beta$ can then be recovered from (17) by setting

$$\beta_t = \hat{\beta}_t + \sum_{\substack{j \\ b_j < 0}} m_j b_j. \tag{18}$$

It remains to compute

$$\hat{\beta}_t = \min\{\hat{b}\hat{x} \mid \hat{a}^t \hat{x} \geqslant \hat{a},\ \hat{x} \geqslant 0\}, \tag{19}$$

where

$$\hat{a}_j^t = \begin{cases} a_j^t & \text{if } b_j^t \geqslant 0, \\ -a_j^t & \text{otherwise} \end{cases} \tag{20}$$

and

$$\hat{\alpha}_t = \alpha_t - \sum_{\substack{j \\ b_j < 0}} m_j a_j^t. \tag{21}$$

Because $\hat{b} \geqslant 0$, LP duality applied to (19) yields that

$$\hat{\beta}_t = \min_{\substack{j \\ \hat{a}_j^t > 0}} \left\{ \frac{\hat{b}_j}{\hat{a}_j^t} \right\} \max\{\hat{\alpha}_t, 0\}. \tag{22}$$

This proves,

Theorem 3. *A valid inequality $bx \geqslant \beta'$ for the disjunction* (8) *is supporting if and only if $\beta' = \beta$, where β is defined by* (17),(18) and (22).

3.5. Integral 0–1 representations

The traditional continuous relaxation of a disjunctive constraint may permit fractional solutions even when the original disjunction is satisfied. This means that a traditional branch-and-bound method can keep branching even when a feasible solution has been discovered. It is therefore best to check disjunctions (as well as other logical constraints) directly for feasibility, as done in MLLP.

The 0–1 formulation of the disjunction (5) is the following (see Fig. 5):

$$\begin{aligned} &A^t x \geqslant a^t - M_t(1 - y_t), \quad t \in T, \\ &0 \leqslant x \leqslant m, \\ &\sum_{t \in T} y_t = 1, \\ &y_t \in \{0, 1\}, \quad t \in T, \end{aligned} \tag{23}$$

where M_t satisfies

$$a^t - M_t \leqslant \min\{A^t x \mid 0 \leqslant x \leqslant m\}, \tag{24}$$

and $e = (1, \ldots, 1)$. The claim is that when x is fixed to some value $\bar{x}$, an extreme point solution $y = \bar{y}$ of (23) can be nonintegral even when $\bar{x}$ satisfies (5). An example of this is presented by simple semicontinuous variable, $x \in \{0\} \cup [s_1, s_2]$, or

$$(-x \geqslant 0) \vee (x \geqslant s_1), \quad 0 \leqslant x \leqslant s_2.$$

The continuous relaxation of (23) is

$$\begin{aligned} &-x \geqslant -s_2(1 - y), \\ &x \geqslant s_1 - s_1 y, \\ &0 \leqslant x \leqslant s_2, \\ &0 \leqslant y \leqslant 1. \end{aligned} \tag{25}$$

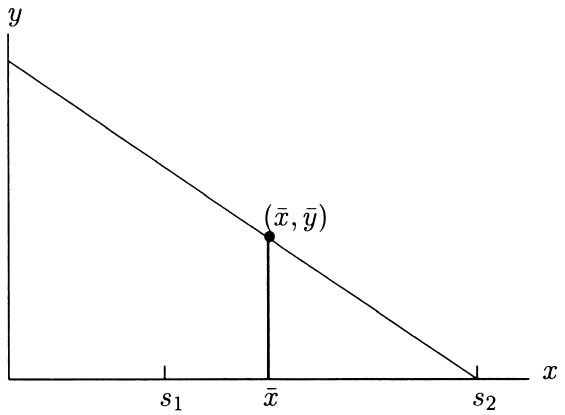

Fig. 5. The line segment from $(\bar{x}, 0)$ to $(\bar{x}, \bar{y})$ is the polytope described by the continuous relaxation of the 0–1 representation of a semicontinuous variable. $(\bar{x}, \bar{y})$ is a fractional extreme point of the polytope even though $\bar{x}$ is a feasible value.

If x is fixed to $\bar{x}$ and (25) is projected onto y, the result is

$$1 - \frac{\bar{x}}{s_1} \leqslant y \leqslant 1 - \frac{\bar{x}}{s_2}, \quad 0 \leqslant y \leqslant 1. \tag{26}$$

If $s_1 \leqslant \bar{x} \leqslant s_2$, $\bar{y} = 1 - \bar{x}/s_2$ is an extreme point solution of (26) and therefore (25), and it is nonintegral whenever $s_1 < \bar{x} < s_2$. So (25) can have extreme point solutions with fractional y even when $\bar{x} \in [s_1, s_2]$, and even though (25) is the best possible (convex hull) relaxation of (23). The extreme point solutions for $\bar{x} \in [s_1, s_2]$ are guaranteed to have integral y only when $s_1 = s_2$, i.e., when x is essentially a rescaled binary variable.

The idea can be defined in general as follows. Let $P_{\bar{x}}$ be the set of points y that satisfy the continuous relaxation of (23) when x is fixed to $\bar{x}$. Let the continuous relaxation of (23) be *integral* if for every $(\bar{x}, \bar{y})$ satisfying (23) such that $\bar{y}$ is an extreme point of $P_{\bar{x}}, \bar{y}$ is integral.

The following characterizes integral relaxations. A disjunct of (5) is *redundant* when its feasible set lies within that of another disjunct. Obviously, redundant disjuncts can be dropped without effect.

Theorem 4. *Suppose that the disjunction* (5) *contains no redundant disjuncts, that* $0 \leqslant x \leqslant M_t$, *and that* M_t *satisfies* (23) *for* $t \in T$. *For* $t, t' \in T$ *with* $t \neq t'$ *define*

$$y_{tt'} = \max\{y_t \mid M_t y_t \leqslant A^t x - a^t + M_t,\ A^{t'} x \geqslant a^{t'},\ 0 \leqslant x \leqslant m,\ y_t \leqslant 1\}.$$

Then the continuous relaxation of (23) *is integral if and only if* $y_{tt'} = 0$ *for every pair* $t, t' \in T$ *with* $t \neq t'$.

Proof. It is clear that $y_{tt'}$ can be written,

$$y_{tt'} = \max\{y_t \mid A^t x \geqslant a^t - M_t(1 - y_t),\ A^{t'} x \geqslant a^{t'},\ 0 \leqslant x \leqslant m,\ y_t \leqslant 1\}. \tag{27}$$

It is convenient to let S_t be the feasible set for disjunct $t \in T$, i.e., $S_t = \{x \mid A^t x \geqslant a^t, 0 \leqslant x \leqslant m\}$. For $\bar{x} \in S_{t'}$ define

$$y_{tt'}(\bar{x}) = \max\{y_t \mid A^t \bar{x} \geqslant a^t - M_t(1 - y_t),\ y_t \leqslant 1\}. \tag{28}$$

Claim. *For any $\bar{x} \in S_{t'}$ and any $t \neq t'$,*

$$y_{tt'}(\bar{x}) = \max_y\{y_t \mid y \in P_{\bar{x}}\}. \tag{29}$$

Proof of Claim. It suffices to show that any y_t that is feasible in (29) is feasible in (28), and vice versa. The former is obvious. To show the latter, let y_t be feasible in (27). To see that if it is feasible in (29), set $y_{t'} = 1 - y_t$ and $y_{t''} = 0$ for $t'' \neq t, t'$. It is enough to show

$$A^{t''}\bar{x} \geqslant a^{t''} - M_{t''}(1 - y_{t''}) \tag{30}$$

for all $t'' \in T$. But (30) holds for $t'' = t$ by stipulation. It holds for $t'' = t'$ because $\bar{x} \in S_{t'}$, and it holds for $t'' \neq t, t'$ by definition of $M_{t''}$. This proves the claim.

Now suppose that $y_{tt'} > 0$ for some t, t' with $t \neq t'$. Because the disjunct t' is not redundant, $S_{t'}$ is nonempty, and one can choose any $\bar{x}^1 \in S_{t'}$ and note that $y_{tt'}(\bar{x}^1) > 0$. Again because disjunct t' is not redundant, one can choose $\bar{x}^2 \in S_{t'} \backslash S_t$ and note that $y_{tt'}(\bar{x}^2) < 1$. There exists a convex combination $\bar{x} \in S_{t'}$ of $\bar{x}^1$ and $\bar{x}^2$ with $0 < y_{tt'}(\bar{x}) < 1$, so that $y_{tt'}(\bar{x})$ is not integral. But (29) implies that some $\bar{y}$ with $\bar{y}_t = y_{tt'}(\bar{x})$ is an extreme point of $P_{\bar{x}}$. It follows that (23) is not integral.

For the converse, suppose that $y_{tt'} = 0$ for all pairs t, t' with $t \neq t'$. It suffices to show that for any $\bar{x}$ satisfying (5), any given extreme point $\bar{y}$ of $P_{\bar{x}}$ is integral. If it is supposed that $\bar{x} \in S_{t'}$, the following can be stated:

$$\begin{aligned} &\max\{y_{t'} \mid y \in P_{\bar{x}}\} = 1 \\ &\max\{y_t \mid y \in P_{\bar{x}}\} = 0, \quad t \neq t'. \end{aligned} \tag{31}$$

The first is due simply to the fact that $\bar{x} \in S_{t'}$. By the above claim, the second is equivalent to $y_{tt'}(\bar{x}) = 0$, which is implied by the fact $y_{tt'} = 0$. Now (31) implies that $P_{\bar{x}}$ is a line segment of unit length extending from the origin in a positive direction along the $y_{t'}$-axis. Thus any extreme point $\bar{y} \in P_{\bar{x}}$ is integral, which means that (23) is integral. □

This specializes to disjunctions with one inequality per disjunct as follows.

Corollary 1. *Consider a disjunction* (8) *with one inequality per disjunction and bounds* $0 \leqslant x \leqslant m$. *If* (8) *contains no redundant disjuncts, then* (23) *is integral if and only if*

$$\max\{a^t x \mid a^{t'} x \geqslant \alpha_{t'},\ 0 \leqslant x \leqslant m\} = \alpha_t - M_t \tag{32}$$

for every $t, t' \in T$ with $t \neq t'$.

The conditions in Theorem 4 and Corollary 1 are quite strict. In fact,

Corollary 2. *The continuous relaxation of* (23) *is integral only if the feasible sets described by the disjuncts of* (5) *are disjoint.*

Proof. Suppose two of the feasible sets intersect, e.g. those corresponding to disjuncts t and t'. Then $y_t^*(t') = 1$, which violates the condition of the theorem. □

Not even disjoint feasible sets are sufficient for integrality, as the above example shows. Furthermore, Corollary 1 and (15) imply that when there are two disjuncts containing one inequality each, (23) is integral only if the feasible sets of the disjuncts are vertices or other faces of the box $0 \leqslant x \leqslant m$. Corollary 2 implies that the faces must also be disjoint.

3.6. Beaumont's inequalities

Beaumont [7] identified a class of facet-defining inequalities for disjunctive constraints in which each disjunct consists of a single inequality, as in (8). They are facet-defining in the sense that, under certain conditions, they define facets of the convex hull of the feasible set of (8). Unfortunately, these conditions are often unsatisfied, which limits the usefulness of the inequalities.

Beaumont's approach is essentially a reasonable method for choosing multipliers u^t so as to generate a disjunctive inequality (6). He first incorporates the bounds $x \leqslant m$ into the disjunction (8) to obtain

$$\bigvee_{t\in T} \begin{bmatrix} -I \\ a^t \end{bmatrix} x \geqslant \begin{bmatrix} -m \\ \alpha_t \end{bmatrix}, \quad t \in T.$$

The vector of nonnegative multipliers for each disjunct is $u^t = (v^t, w_t)$, where w_t corresponds to the last inequality in the disjunct. The object is to derive an inequality $bx \geqslant \beta$ that satisfies

$$b \geqslant w_t a^t - v^t$$
$$\beta \leqslant w_t \alpha_t - v^t m$$

for all t. For a given w_t (yet undetermined), it is reasonable to make the components of b as small as possible to get a tight constraint. So let

$$b = \min_t \{w_t a^t\}, \tag{33}$$

where the minimum is taken componentwise. One can now set

$$v^t = w_t a^t - b, \quad t \in T,$$

because (33) implies $v^t \geqslant 0$. To make the right-hand side of the inequality as tight as possible, set

$$\beta = \min_{t\in T} \{w_t \alpha_t - v^t m\}. \tag{34}$$

It remains to pick values for the w_t's. Beaumont's choice is equivalent to setting $w_t = M_t$ when M_t is derived from the variable bounds as in (10) and $a^t \leqslant 0$. Thus

$$w_t = \frac{1}{\alpha_t - a^t m}. \tag{35}$$

The approach breaks down when the denominator is nonpositive, whereupon Beaumont suggests letting

$$w_t = \frac{1}{\alpha_t - \min\{a^t, 0\}m}. \tag{36}$$

Theorem 5 (Beaumont [7]). *The inequality $bx \geqslant \beta$ given by* (33)–(35) *is facet-defining for* (8) *if $\alpha_t - a^t m > 0$ for all $t \in T$.*

Beaumont's inequality can therefore be superior to a supporting elementary inequality. This is illustrated in Fig. 4, where Beaumont's inequality is the facet-defining inequality $2x_1 + x_2 \leqslant 7$.

Assuming $\alpha_t - a^t m > 0$ is equivalent to assuming that the point $x = m$ is infeasible, in which case it makes sense to separate this point from the feasible set. However, $x = m$ is often feasible, as in the example of Fig. 2. Here (35) puts $(w_1, w_2) = (-\frac{1}{4}, -\frac{1}{5})$, and one must revert to (36), which yields the useless inequality $3x_1 + 2x_2 \geqslant -2$.

The underlying difficulty is that Beaumont's approach has no mechanism for detecting which corner of the box $0 \leqslant x \leqslant m$ should be cut off from the feasible set.

3.7. Optimal separating inequalities

When valid inequalities are added to the lineaer constraint set, there is always the possibility that most of them will never play a role in the solution process. That is, the relaxations may provide the same bounds even if most of the inequalities are removed.

This is true of the traditional continuous relaxation of an MILP model, for example. The relaxation is nothing other than a set of valid inequalities, most of which are generally inactive in the solution of the relaxation.

This phenomenon can be avoided by generating only *separating inequalities*, which are valid inequalities that are violated by the current solution of the inequality constraints.

It is straightforward to state a small LP problem whose solution identifies an separating inequality for a disjunction if and only if one exists. Thus if no separating inequality is found, the current solution is known to lie within the convex hull of the feasible set. In this case, branching is necessary to obtain a feasible solution, unless of course the current solution is already feasible. The inequality is *optimal* in the sense that it is chosen to maximize the amount by which the current solution violates it. Unlike Beaumont's and elementary inequalities, this sort of inequality can be generated when the disjuncts contain more than one inequality.

Suppose that the solution $\bar{x}$ of the current LP is to be separated from the feasible set of the disjunctive constraint (5). Any upper bounds $x \leqslant m$ should be incorporated into each disjunct of (5). Because any disjunctive inequality is defined by a choice of multipliers u^t, an LP model can be formulated so as to find a set of u^t's that define an inequality $bx \geqslant \beta$ that is maximally violated by $\bar{x}$. Such a model is,

$$\begin{aligned} \max \quad & \beta - b\bar{x} \\ \text{s.t.} \quad & \beta \leqslant u^t a^t, \quad t \in T, \\ & b \geqslant u^t A^t, \quad t \in T, \\ & -e \leqslant b \leqslant e, \\ & u^t \geqslant 0, \quad t \in T, \\ & \beta, b \text{ unrestricted.} \end{aligned} \tag{37}$$

Note that the variables in the model are β, b, u. If the objective function value is zero, there is no separating inequality. The constraint $-e \leqslant b \leqslant e$ ensures that an optimal solution exits.

The model (37) has an interesting dual:

$$\begin{aligned} \min \quad & (s + s')e \\ \text{s.t.} \quad & \bar{x} - \sum_{t \in T} x^t = s - s' \quad (b), \\ & A^t x^t \geqslant a^t y_t, \quad t \in T \quad (u^t), \\ & \sum_{t \in T} y_t = 1 \quad (\beta), \\ & s, s', x^t, y_t \geqslant 0, \quad t \in T. \end{aligned} \tag{38}$$

If $s - s'$ is fixed to zero and $\bar{x}$ is a variable, the constraint set is Balas' convex hull representation for the disjunction (5) [4]. That is, when $s - s' = 0$, the projection of the feasible set of (38) onto the $\bar{x}$-space is the convex hull of the feasible set of (5). (This is related to the fact, observed by Williams [71], that the dual of the dual of a disjunctive programming problem is the convex hull representation of the problem.) The problem (38) therefore seeks a point $\sum_{t \in T} x^t$ in the convex hull that is closest to $\bar{x}$, as measured by the rectilinear distance.

An optimal separating inequality can be superior to a supporting elementary inequality. Consider the example of Fig. 4, which becomes

$$\begin{pmatrix} -3x_1 + x_2 \geqslant -3 \\ -x_1 \geqslant -3 \\ -x_2 \geqslant -3 \end{pmatrix} \vee \begin{pmatrix} -x_2 \geqslant -1 \\ -x_1 \geqslant -3 \\ -x_2 \geqslant -3 \end{pmatrix}.$$

The solution of (37) for $\bar{x}=(1,1)$ is $\beta=-\frac{7}{2}$, $b=(-1,-\frac{1}{2}), u^1=(\frac{1}{3},0,\frac{5}{6})$, $u^2=(\frac{1}{2},1,0)$, which produces the facet-defining inequality $2x_1+x_2\leqslant 7$.

The optimal separating inequality need not be facet-defining, however. If the convex hull of the disjunction is the box defined by $0\leqslant x_j\leqslant m$ for $j=1,2$, the optimal separating inequality for $\bar{x}=(2,2)$ is $x_1+x_2\leqslant 2$.

Optimal separating inequalities are roughly analogous to the optimal disjunctive cuts used in the lift-and-project method of Balas et al. [5]. One difference is that lift-and-project cuts involve integer variables. Another is that they are derived from disjunctions of the form $y_j=0\vee y_j=1$. Optimal separating inequalities may be derived from any disjunction, and they are valid only in those portions of the search tree where the disjunction is valid. Optimal separating inequalities have not been evaluated computationally, but the success of lift-and-project cuts suggests that an evaluation is worthwhile.

4. Logic processing

Logic processing can be understood as the derivation of logical implications from the constraint set. It generates *valid logic constraints*, which are formulas $q(h,y)$ that are implied by the set S of formulas $q_i(y,h)$ in the model; i.e., all truth values of y and discrete values of h that make the formulas in S true also make $q(h,y)$ true.

Valid logic constraints are derived by inference algorithms that may also go by the name of constraint propagation, preprocessing, etc. Feasibility checking is a special case of inference, because a set of formulas is unsatisfiable if and only if they imply a logical contradiction, such as $x_j\wedge\neg x_j$.

Cutting plane algorithms are actually special cases of inference algorithms. An inequality can be viewed as a formula that is true when it is satisfied. A cutting plane for a constraint set S is an inequality that is satisfied by all integer points that are feasible in S, and it is therefore an implication of S. Logic processing algorithms can therefore be viewed as logical analogs of cutting plane algorithms.

The advantage of logic processing is that it can reduce backtracking. It may, for example, determine that the logical constraint set is infeasible and thereby prune the search tree at the current node. It may also generate valid logic constraints that will prune the search tree at a later time. Suppose, for example, that the formulas

$$\begin{aligned} &x_1\vee x_{100}\\ &x_1\vee\neg x_{100}\end{aligned}\tag{39}$$

are among the logical constraints. Obviously x_1 can be false in no feasible solution. Yet if one branches on the variables in the order $x_1,\ldots,x_{100}$ and takes the branch $x_1=$ false first, one could conceivably search all $2^{100}-1$ nodes in the corresponding subtree before discovering that x_1 must be true. However, if the valid logic constraint x_1 had been derived from (39), the subtree could have been eliminated immediately.

A theory to support this view of constraint generation has been developed in the constraint satisfaction literature. Generating the constraint x_1, for example, is viewed as increasing the degree of "consistency" of the constraint set, which in turn reduces backtracking. Consistency is not feasibility, as the word may suggest, but is roughly analogous to integrality in a polyhedral setting, because a totally consistent constraint set can be solved without backtracking. There is no space to present this theory here, but an expository development written for mathematical programmers is available in [36].

In the context of MLLP, generating valid logic constraints has another advantage. It may be possible to define relaxations for the logic constraints in the continuous part of the model, thereby strengthening the overall relaxation.

The discussion here will be limited to three types of inference algorithms that are useful for logic processing: resolution, a simple form of constraint propagation, and the derivation of "l-cuts" for knapsack constraints. In general one does not carry any of these algorithms to completion. It is usually best to generate a few implications that seem most useful for the problem at hand.

Valid (and nonvalid) logic constraints can also be derived from the special structure of a problem, much as is done for polyhedral cuts. These constraints may be valid or nonvalid and are discussed briefly below.

4.1. Resolution

Resolution [49,50,56] was originally defined for logical *clauses*, which are disjunctions of *literals* (atomic propositions or their negations). Resolution can derive valid logic constraints for any set of formulas $q(y)$ in which the variables y are atomic propositions, because any such formula is equivalent to a finite set of clauses.

Clause C_1 implies clause C_2 if and only every literal of C_1 occurs in C_2. Two clauses have a (unique) *resolvent* when exactly one variable y_j occurs positively in one and negatively in the other. The resolvant is a disjunction of all literals that occur in either clause except y_j and $\neg y_j$. For instance, $y_2 \vee \neg y_3$ is the resolvent of $y_1 \vee y_2$ and $\neg y_1 \vee \neg y_3$. Given a set S of clauses, the resolution algorithm picks a pair of clauses in S that have a resolvent that is implied by no clause in S, and adds the resolvent to S. It repeats until there is no such pair, which occurs after finitely many iterations.

Theorem 6 (Quine [49,50]). *A clause set S implies clause C if and only if the resolution algorithm applied to S generates a clause that implies C. In particular, S is unsatisfiable if and only if resolution generates the empty clause.*

Thus, resolution is somewhat analogous to Chvátal's cutting plane procedure, because it generates all valid logic constraints in clausal form. Quine's theorem follows from Theorem 7, proved below. Resolution has exponential complexity in the worst case [25] and can be very slow in the typical case [28]. In practice, however, one would generate a limited number of resolvents, such as those with k or fewer literals, for some small k.

Any formula $q(y,h)$ that contains both atomic propositions y and discrete variables h is equivalent to a finite set of *multivalent clauses.* Logic constraints can be derived for a set of formulas by applying a generalized form of resolution to clauses implied by them. A multivalent clause has the form

$$\bigvee_{j=1}^{n}(h_j \in H_j), \tag{40}$$

where each H_j is a subset of the *domain* D_j of h_j. For notational simplicity, it is assumed that an atomic proposition y_j is written $h_j \in \{T\}$ or $h_j \in \{F\}$, where h_j is a bivalent variable. If H_j is empty, the term $(h_j \in H_j)$ can be omitted from (40), but it is convenient to suppose that (40) contains a term for each h_j. One multivalent clause $\bigvee_j(h_j \in H_{1j})$ implies another $\bigvee_j(h_j \in H_{2j})$ if and only if $H_{1j} \subset H_{2j}$ for each j.

The multivalent resolution algorithm is related to Cooper's algorithm for obtaining k-consistency for a set of constraints [19]. Given a set of multivalent clauses.

$$\left\{ \bigvee_{j=1}^{n}(h_j \in H_{ij}) \,\middle|\, i \in I \right\}, \tag{41}$$

the *resolvent on* h_k of these clauses is

$$\left(h_k \in \bigcap_{i\in I} H_{ik}\right) \vee \bigvee_{j\neq k}\left(h_j \in \bigcup_{i\in I} H_{ij}\right).$$

Ordinary bivalent resolution is a special case.

For example, the first three clauses below resolve on h_1 to produce the fourth. Here each h_j has domain $\{1,2,3,4\}$:

$$(h_1 \in \{1,4\}) \vee (h_2 \in \{1\}),$$
$$(h_1 \in \{2,4\}) \vee (h_2 \in \{1,2,3\}),$$
$$(h_1 \in \{3,4\}) \vee (h_2 \in \{1\}),$$
$$(h_1 \in \{4\}) \vee (h_2 \in \{1,2,3\}).$$

It is pointless to resolve the first three clauses on h_2, because this produces the tautology,

$$(h_1 \in \{1,2,3,4\}) \vee (x_2 \in \{1\}).$$

To apply the resolution algorithm to a set S of multivalent clauses, find a subset of S whose resolvent M is implied by no clause in S, and add M to S. Continue until no further clauses can be added to S.

The multivalent resolution algorithm derives all multivalent clauses that are valid logic constraints for a given set of multivalent clauses. The proof of the theorem uses the idea of Quine's original proof for ordinary resolution.

Theorem 7. *A set S of multivalent clauses implies a multivalent clause M if and only if the multivalent resolution algorithm applied to S generates a clause that implies M.*

Proof. Multivalent resolution derives only implication of S because it is clearly valid. To prove the converse, let S' be the result of applying the algorithm to S. Also define the *length* of a clause (40) be $\sum_j |H_j|$. Suppose the theorem is false, and let (40) be a longest clause implied by S but by no clause in S'.

Claim. *At least one H_j in* (40) *is missing at least two elements, i.e.,* $|D_j \backslash H_j| \geqslant 2$ *for some* j.

First it is clear that no $H_j = D_j$, because otherwise (40) would be implied by a (in fact, every) clause in S'. Suppose contrary to the claim that every H_j is missing exactly one element, say v_j. Then $h = v = (v_1, \ldots, v_n)$ violates (40) and must therefore violate some clause $\bigvee_j (h_j \in H'_j)$ in S', because S' implies (40). This means each $H'_j \subset D_j \backslash \{v_j\}$, so that $\bigvee_j (h_j \in H'_j)$ implies (40), contrary to hypothesis. This proves the claim □

Now suppose v_k, v'_k are missing from H_k, and consider the multivalent clauses

$$(h_k \in H_k \cup \{v_k\}) \vee \bigvee_{j \neq k} (h_j \in H_j), \quad (h_k \in H_k \cup \{v'_k\}) \vee \bigvee_{j \neq k} (h_j \in H_j). \tag{42}$$

They must respectively be implied by clauses M_1, $M_2 \in S'$ because they are longer than (40). This means that the resolvant of M_1, M_2 on h_k implies (40). So by construction of the resolution algorithm, S' contains a clause that implies (40), contrary to hypothesis. □

The proof of the theorem shows that it suffices in principle to generate resolvents only of pairs of clauses.

Resolution can be generalized so as to obtain all valid constraints in the form of 0–1 knapsack constraints (discussed in Section 4.3 below) for a system of such constraints [31]. Barth [6] specialized this approach to obtain constraint generation techniques for *extended clauses* of the form $\sum_{j \in J} x_j \geqslant k$. These inequalities seem to be a useful compromise between 0–1 inequalities and logical clauses, because they retain some of the expressiveness of the former and are yet amenable to logic processing.

4.2. Constraint propagation

Unit resolution, also known as *forward chaining*, provides as fast and very useful constraint propagation algorithm for logical clauses. It is the same as full resolution except that one of the parents of a resolvent is always a unit clause. For example, unit resolution fixes y_1 to true in the following clause set:

$$y_3,$$
$$y_2 \vee \neg y_3,$$
$$y_1 \vee \neg y_2 \vee \neg y_3.$$

```
Let S be a set {∑_{j∈J_i} L_ij ≥ k_i | i ∈ I} of extended clauses,
   where each L_ij is y_j or ¬y_j.
Let U be a stack of unit clauses, initially empty.
For each i ∈ I with |J_i| = k_i {
   For each j ∈ J_i add L_ij to U.
   Let J_i = ∅.
}
While U is nonempty {
   Remove L'_t from U.
   For each i ∈ I with t ∈ J_i {
      If L_it = L'_t then let k_i = k_i − 1, J_i = J_i \ {t}.
      Else {
         If k_i = |J_i| then stop; S is unsatisfiable.
         Else {
            If k_i = |J_i| + 1 then {
               For each j ∈ J_i \ {t} add L_ij to U.
               Let J_i = ∅.
            }
            Else let J_i = J_i \ {t}.
         }
      }
   }
}
```

Fig. 6. A unit resolution algorithm for extended clauses.

Unit resolution is incomplete (i.e., does not derive all valid constraints), as can be seen in the example,

$$y_1 \vee y_2 \vee y_3,$$
$$y_1 \vee \neg y_2 \vee y_3,$$
$$y_1 \vee y_2 \vee \neg y_3,$$
$$y_1 \vee \neg y_2 \vee \neg y_3.$$

Full resolution fixes y_1 to true, but unit resolution does nothing because there are unit clauses to start with. Unit resolution is efficient, however, as it runs in $O(nL)$ time, if there are L literals, and it tends to be very fast in practice.

Unit resolution is easily generalized to broader classes of formulas. It is adapted to extended clauses in Fig. 6. A version for multivalent clauses and all-different constraints appears in Fig. 7.

Unit resolution is a complete inference algorithm for certain classes of clauses, such as Horn clauses, renamable Horn clauses, extended Horn clauses, etc. [14–16,58,64]. No known structural property of a clause set is necessary and sufficient for the completeness of unit resolution.

Unit resolution has the same inferential power as linear programming in the following sense. Suppose that the clauses of S are written as a system $Ay \geqslant a$ of 0–1 inequalities

```
Let S be a set {C_i | i ∈ I} of multivalent clauses, where each C_i
   has the form ⋁_{j=1}^{m}(h_j ∈ H_ij) ∨ ⋁_{t∈T_i} alldiff({h_j | j ∈ J_t})
Let n_i be the number of terms (h_j ∈ H_ij) of C_i with nonempty H_ij.
Let U be a stack of indices representing active domains;
   initially U = {1,...,m}.
Let A be a list of enforced alldiff predicates, initially empty.
For each i ∈ I {
   If n_i = 0 and |T_i| = 1 then:
      Add the alldiff predicate in C_i to A and remove i from I.
   Else if n_i = 1 and |T_i| = 0 then
      Let H_ij be nonempty; let D_j = D_j ∩ H_ij and remove i from I.
}
While U is nonempty {
   Remove an index k from U.
   If D_k is empty then stop; S is unsatisfiable.
   For all i ∈ I {
      If H_ik is nonempty then {
         If D_k ⊂ H_ik then remove i from I.
         Else {
            Let H_ik = H_ik ∩ D_k.
            If H_ik is empty then {
               Let n_i = n_i − 1.
               If n_i = 1 and |T_i| = 0 then {
                  Let H_ij be nonempty and remove i from I.
                  If D_j ⊄ H_ij then
                     Let D_j = D_j ∩ H_ij and add j to U.
               }
               If n_i = 0 and |T_i| = 1 then {
                  Remove i from I.
                  Add the alldiff predicate in C_i to A.
               }
            }
         }
      }
   }
   For each predicate alldiff({h_j | j ∈ J}) in A with k ∈ J
      If |D_k| = 1 then
         For j ∈ J \ {k}
            If D_k ⊂ D_j then Let D_j = D_j \ D_k and add j to U.
}
```

Fig. 7. A unit resolution algorithm for multivalent clauses.

in the usual fashion, i.e., a clause $\bigvee_{j\in J} L_j$ is written $\sum_{j\in J} y_j(L_j)\geqslant 1$, where $y_j(L_j)$ is y_j if $L_j = y_j$ and is $1 - y_j$ if $L_j = \neg y_j$.

Theorem 8 (Blair et al. [10]). *Unit resolution finds a contradiction in the clause set S if and only if the linear relaxation of the corresponding system $Ay\geqslant a$ of* 0–1 *inequalities is infeasible.*

$Ay\geqslant a$ is infeasible when unit resolution finds a contradiction because unit resolution (unlike resolution in general) simply adds the inequality representations of clauses. So deriving the empty clause is equivalent to obtaining $0\geqslant 1$ from a nonnegative linear combination of $Ay\geqslant a$. Conversely, if unit resolution detects no contradiction, then the inequalities that represent the remaining clauses can be satisfied by setting each $y_j = 1/2$.

Although LP duplicates the effect of unit resolution, the latter is preferable for logic processing because it is much faster.

4.3. Knapsack constraints

The familiar 0–1 knapsack constraint $dy\geqslant\delta$, where each $y_j \in \{0,1\}$, can also be regarded as a logical formula that is true when the sum over b_j for which y_j is true is at least β. Boolean functions of this form are called *threshold* functions and are studied in the electrical engineering literature [60]. They are difficult to process logically, but they can be used to generate logic constraints in the form of clauses and extended clauses, which are easily manipulated. For example, the logical clauses implied by a knapsack constraint are identical to the well-known "covering inequalities" for the constraint, and their derivation is straightforward (e.g., [24]).

It may be more effective, however, to infer extended clauses. Although it is hard to derive all the extended clauses that are implied by a constraint, it is easy to derive all 1-*cuts*. Consider a 0–1 inequality $dy\geqslant\delta$ for which it is assumed, without loss of generality, that $d_1\geqslant d_2\geqslant\cdots\geqslant d_n > 0$; if $d_j < 0$, reverse its sign and add d_j to δ. A 1-cut for $dy\geqslant\delta$ is one of the form

$$y_1 + y_2 + \cdots + y_j \geqslant k. \tag{43}$$

The algorithm of Fig. 8, presented in [37], derives all valid 1-cuts. By way of example, the knapsack constraint

$$13y_1 + 9y_2 + 8y_3 + 6y_4 + 5y_5 + 3y_6 \geqslant 30$$

gives rise to the 1-cuts,

$$y_1 + y_2 \geqslant 1,$$
$$y_1 + y_2 + y_3 \geqslant 2,$$
$$y_1 + y_2 + y_3 + y_4 + y_5 \geqslant 3.$$

The first cut could be deleted if desired, because it is redundant of the second. 1-cuts and related cuts are discussed further in [37].

```
Let k = 1, s = Σ_{j=1}^{n} d_j, k_last = 0.
For j = 1,...,n {
   Let s = s − d_j.
   If s < δ then {
      While s + d_k < δ:
         Let s = s + d_k, k = k + 1.
      If k > k_last then {
         Generate the cut y_1 + ... + y_j ≥ k.
         Let k_last = k.
      }
   }
}
```

Fig. 8. An algorithm for generating all 1-cuts for a knapsack constraint $dy \geqslant \delta$ in which $d_1 \geqslant d_2 \geqslant \cdots \geqslant d_n > 0$.

4.4. Structural logic constraints

An intuitive understanding of a problem can suggest logic constraints even when no further valid inequalities are easily identified. Such constraints may be nonvalid as well as valid, as proposed by [38] in connection with the process synthesis example discussed in Section 6.2. Structural constraints have also been derived for truss design problems [11], matching problems [33], and a series of standard 0–1 problems discussed by Wilson [75].

A valid logic constraint was defined above for a set of formulas. It can be defined for an MLLP model (1) as a formula $q(y,h)$ that is true for every (x,y,h) that is feasible in (1). For example, $\neg y_3$ is a valid logic constraint for the problem

$$\begin{array}{ll|l} \min & x_1 + x_2 & \\ \text{s.t.} & y_1 \rightarrow (x_1 \geqslant 1) & \\ & y_2 \rightarrow (x_2 \geqslant 1) & \\ & y_3 \rightarrow (x_1 + x_2 \leqslant 0) & y_1 \vee y_2 \\ & x_1, x_2 \geqslant 0, & \end{array} \tag{44}$$

but is not implied by the formula $y_1 \vee y_2$.

Logic constraints can be defined in a more general sense that permits them to be nonvalid. Let (y,h) be *feasible* in (1) if (x,y,h) is feasible in (1) for some x. Let (y',h') *dominate* (y,h) if for any (x,y,h) that is feasible in (1), there is a feasible (x',y',h') for which $cx' \leqslant cx$. Then $q(y,h)$ might be called a *quasi-valid* logic constraint if any feasible (y,h) that makes $q(y,h)$ false is dominated by a feasible (y',h') that makes $q(y',h')$ true. A quasi-valid constraint may be added to (1) without changing the optimal solution, but it may exclude feasible solutions.

For example, the formulas $\neg y_1$ and $\neg y_2$ are quasi-valid logic constraints for (44). They are nonvalid because they exclude the feasible points (1,0,0), (1,1,0).

5. A generic branching algorithm

A generic branching algorithm for MLLP appears in Fig. 9. For simplicity, it assumes that the propositions p_j in (1) are atomic propositions y_j, which is the case for all the problems solved in the next section. When branching fixes y_j to true or false, the formula y_j or $\neg y_j$ is added to the set Q of logical formulas $q_i(y,h)$. When h_j is fixed to v, the domain D_j of h_j is reduced to $\{v\}$. Again for simplicity, it is assumed that one branches on h_j by setting it to one value at a time, but one could branch by partitioning its domain into subsets containing more than one element.

Logic processing is applied to Q at each node. It may change the content of Q or remove elements from some D_j's. Linear relaxations of formulas in Q are added to the set L of linear inequalities, if desired.

If Q or L is infeasible, the algorithm backtracks. Otherwise, the solution $\bar{x}$ of the LP relaxation will in general satisfy certain constraint sets $A^j x \geqslant a^j$ and not others. If proposition y_j is not already fixed to true or false, it is temporarily assumed true if $\bar{x}$ satisfies $A^j x \geqslant a^j$ and false otherwise. If an unfixed y_j corresponds to an empty constraint set, it can be given a default temporary value that applies until it is fixed otherwise. If the values of the y_j's and h_j's, including the temporary values, make the formulas in Q true, $\bar{x}$ is a feasible solution. Otherwise optimal separating inequalities are added to L if desired. If there are no separating inequalities, a variable is chosen for branching.

Traditional branch-and-cut for mixed 0–1 problems can be seen as a special case of the algorithm of Fig. 9 by formally expressing the problem as follows:

$$
\begin{array}{ll}
\min & cx \\
\text{s.t.} & Ax \geqslant a, \\
& 0 \leqslant x_j \leqslant 1, \quad j = 1,\ldots,r, \\
& y_j \rightarrow (x_j = 1), \quad j = 1,\ldots,r, \\
& \neg\, y_j \rightarrow (x_j = 0), \quad j = 1,\ldots,r.
\end{array}
$$

Here branching on the y_j's is equivalent to branching on the 0–1 variables $x_1,\ldots,x_r$. General MILP problems can be written,

$$
\begin{array}{lll}
\min & cx & \\
& Ax \geqslant a, & \\
& 0 \leqslant h_j \leqslant m_j, \quad j = 1,\ldots,r, & \\
\text{s.t.} & (h_j = k) \rightarrow (x_j \geqslant k),\ k = 0,\ldots,m_j,\ j = 1,\ldots,r, & \displaystyle\bigvee_{k=0}^{m_j} (h_j = k \wedge h'_j = k), \\
& (h'_j = k) \rightarrow (x_j \leqslant k),\ k = 0,\ldots,m_j,\ j = 1,\ldots,r, & j = 1,\ldots,r.
\end{array}
$$

One can branch on the alternatives $x_j \leqslant k-1$, $x_j \geqslant k$ by setting $h'_j = k-1$ and then $h_j = k$.

```
Let Q be a set of logical formulas, initially the formulas
   q_i(y,h) in (1).
Let L be a set of linear inequalities, initially empty,
Let T,F,U indicate true, false and undefined.
Let ȳ be a vector of truth values for y, initially ȳ = (U,...,U).
Let D = (D_1,...,D_m) be the domains of h_1,...,h_m.
Let z̄ be an upper bound on the optimal value, initially ∞.
Let A be the set of active nodes, initially with A = {(Q,L,ȳ,D)}.
While A is nonempty {
   Remove a tuple (Q,L,ȳ,D) from A.
   Apply a logic processing algorithm to Q, possibly changing
      the contents of Q, possibly changing some ȳ_j's from U to T
      or F, and possibly removing elements from some D_j's.
   If no logical contradiction is detected then {
      For each ȳ_j changed to T, add A^j x ≥ a^j to L.
      Generate inequality relaxations for formulas in Q and add them to L.
      Let x̄ minimize cx subject to L.
      If cx̄ < z̄ then {
         For each y_j {
            If ȳ_j ∈ {T,F} then let ŷ_j = ȳ_j.
            Else let ŷ_j = T if A^j x̄ ≥ a^j and ŷ_j = F otherwise.
         }
         Let C, initially empty, be the set of unsatisfied formulas.
         For each q_i(y,h) ∈ Q {
            If q_i(ŷ,h̄) is F or U then {
               If desired, try to generate a separating inequality for
                  q_i(y,h) with respect to (ŷ,h̄).
               If a separating inequality is generated then add it to L.
               Else add q_i(y,h) to C.
            }
         }
         If C is empty then {
            If no separating inequalities were generated then
               x̄ is feasible; let x* = x̄ and z̄ = cx̄.
            Else add (Q,L,ȳ,D) to A.
         }
         Else {
            Choose a variable y_j with ȳ_j = U or a variable h_j with
               |D_j| > 1, such that setting y_j to T or F, or setting h_j
               to one of its discrete values, satisfies or tends
               to satisfy one of the formulas in C.
            If y_j is chosen then
               Add (G ∪ {y_j},L,ȳ,D) and (Q ∪ {¬y_j},L,ȳ,D) to A.
            Else if h_j is chosen then
               For each v ∈ D_j:
                  Set D̄ = D, set D̄_j = {v}, and add (Q,L,ȳ,D̄) to A.
         }
      }
   }
}
If z̄ < ∞ then x* is an optimal solution.
Else the problem is infeasible.
```

Fig. 9. A generic branching algorithm for MLLP.

6. Some examples

Examples from four application areas are formulated and solved. The aim is to illustrate how to choose between a traditional integer programming approach and other MLLP options for a given problem. An attempt was made to choose problems with the flavor or complexity of real applications, although the warehouse location problem is somewhat stylized.

Each problem is formulated as an MLLP without any integer variables and as a traditional MILP. Both are solved with the generic algorithm of Fig. 9, which in the case of an MILP reduces to traditional branch-and-cut. The simplest possible algorithm is used in either case, in order to isolate the effect of the specific MLLP features illustrated by each problem.

For logic-based models, the generic algorithm of Fig. 9 is fleshed out as follows. The search tree is traversed in depth-search manner, so that memory requirements for the tree are modest. The branching rule is to branch on the first propositional variable in the first unsatisfied logical formula. Logic processing consists of the unit resolution algorithms of Figs. 6 and 7. The logical formulas were represented in the same data structure used to provide inequality constraints to CPLEX. The relaxation of logical formulas varies from case to case, as described below. The code is written in C and compiled with the Sun C compiler version 1.1 with optimization. The tests were conducted on a SPARC Station 330 running SUN OS version 4.1.1 and with xx megabytes memory. The LP relaxations were solved by CPLEX version 3.0.

The MILP algorithm is a straightforward branch-and-bound procedure. The branching rule is to branch on a variable whose value in the relaxation is nearest 1/2. The LP relaxations were solved with the same CPLEX routine.

Run times and node counts for version 2.1 of the CPLEX MILP code are also reported. It is argued in [35], however, that comparison with a commercial code may provide limited insight. The many features of a commercial code make it difficult to isolate which are responsible for performance differences.

6.1. A flow shop problem

A scheduling problem that frequently occurs in chemical processing is a flow shop problem with zero-wait transfer. There are several jobs, each representing a batch of some reagent. Each job is processed on several machines (reactors). The machines are always visited in the same order, but a given job may skip some of the machines. When a job's processing is completed on one machine, it must move immediately to the next machine in its sequence. The objective is to minimize makespan.

Let J_i be the set of machines on which job i is processed, and d_{ij} the processing time for job i on machine j. If t_i is the start time for job i, the job is completed at time

$$t_i + \sum_{j \in J_i} d_{ij}.$$

It is necessary to make sure that two jobs i,k are not scheduled to be in process at the same time on the same machine $j \in J_i \cap J_k$. The finish time of job i on machine j is $t_i + D_{ij}$, where

$$D_{ij} = \sum_{\substack{j' \in J_i \\ j' \leqslant j}} d_{ij'},$$

and its start time is $t_i + D_{ij} - d_{ij}$. To avoid clashes one must say that for each machine j on which jobs i,k are processed, job k starts after job i has finished, or vice versa. Thus for each pair (i,k),

$$(t_i + D_{ij} \leqslant t_k + D_{kj} - d_{kj},\ j \in J_i \cap J_k) \vee (t_k + D_{kj} \leqslant t_i + D_{ij} - d_{ij},\ j \in J_i \cap J_k).$$

The inequalities is either disjunct are the same except for the right-hand side. It is therefore necessary to write only one disjunction in each disjunct, using the tightest right-hand side. An MLLP model is

$$\begin{array}{ll} \min & T \\ \text{s.t.} & t_i \geqslant 0,\ T \geqslant t_i + \sum_{j \in J_i} d_{ij},\ \text{all } i \\ & y_{ik} \rightarrow (t_k - t_i \geqslant r_{ik}) \end{array} \left| \begin{array}{l} \\ \\ \\ y_{ik} \vee y_{ki},\ \text{all } i,k,\ i \neq k, \end{array} \right. \tag{45}$$

where

$$r_{ik} = \max_{j \in J_i \cap J_k} \{D_{ij} - D_{jk} + d_{kj}\}.$$

A traditional MILP model can be formulated with big-M constraints.

$$\begin{array}{ll} \min & T \\ \text{s.t.} & t_i \geqslant 0,\ T \geqslant t_i + \sum_{j \in J_i} d_{ij}, \quad \text{all } i, \\ & t_k - t_i \geqslant r_{ik} - M(1 - y_{ik}), \quad \text{all } i,k,\ i \neq k, \\ & t_i - t_k \geqslant r_{ki} - M y_{ik} \quad \text{all } i,k,\ i \neq k, \\ & y_{ik} \in \{0,1\}, \quad \text{all } i,k. \end{array} \tag{46}$$

The problem can also be solved by solving m traveling salesman problems, where m is the number of jobs [47].

In this case one can anticipate that the logic-based formulation (45) is best, for two reasons: (a) the MILP representation of the disjunctions is not integral, and (b) the linear relaxation of 0–1 scheduling constraints is weak (as discussed in Section 3.1), so that there is little to be lost in forfeiting it. If there are m jobs and n machines, eliminating integer variables reduces the number of variables in the LP relaxation from $2m + mn$ to $2m$.

The nonintegrality of the MILP representation follows from Corollary 1, which implies that it is integral if and only if

$$\begin{aligned} &\max\{t_k - t_i \mid t_i - t_k \geqslant r_{ki},\ (0,0) \leqslant (t_i,t_k) \leqslant (m_i,m_k)\} = r_{ki} - M_{ki}, \\ &\max\{t_i - t_k \mid t_k - t_i \geqslant r_{ik},\ (0,0) \leqslant (t_i,t_k) \leqslant (m_i,m_k)\} = r_{ik} - M_{ik}. \end{aligned} \tag{47}$$

Table 1
Computational results for flow shop problems with zero-time transfer, showing number of nodes in the search tree, time in seconds, and seconds per node

Number of		MLLP			MILP			CPLEX		
Jobs	Machines	Nodes	Time	Per node	Nodes	Time	Per node	Nodes	Time	Per node
6	5	407	2.7	0.0066	689	10.1	0.0147	527	8.1	0.0154
7	5	1951	15.7	0.0080	3171	52.2	0.0165	2647	51.0	0.0193
8	5	14573	129.0	0.0089	24181	546.4	0.0226	16591	413.9	0.0249

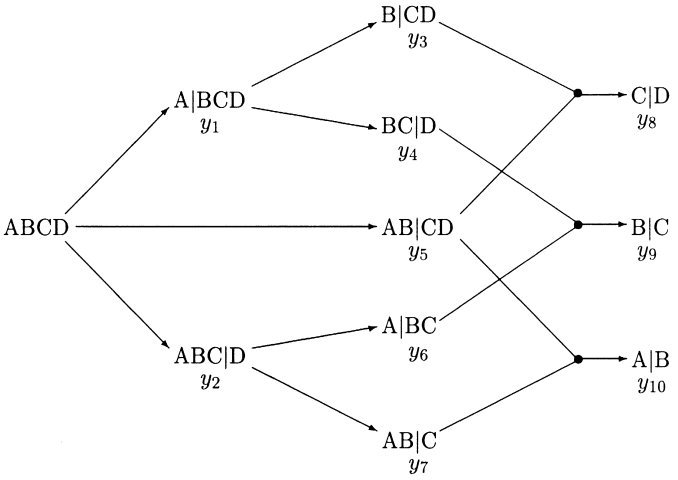

Fig. 10. A 4-component separation network.

Defining M_{ki}, M_{ik} by (15) yields $(M_{ki}, M_{ik}) = (r_{ki} + m_k, r_{ik} + m_i)$. Also it is easy to see that the two maxima in (47) are respectively equal to $-r_{ki}$ and $-r_{ik}$. So (47) implies that the MILP representation is integral if and only if $(r_{ki}, r_{ik}) = (m_k, m_i)$, which does not occur in practice.

Three flow shop problems that represent process scheduling problems in a chemical plant [54] were solved, and the results appear in Table 1. The logic-based approach generated about 60% as many nodes as MILP and used less than half as much time per node. It ran 3–4 times as fast as MILP on these problems.

6.2. A processing network design problem

Another common problem in chemical engineering is the design ("synthesis") of processing networks. For instance, one may wish to separate the components (A,B,C,D) of a mixture by passing it through various distillation units, as illustrated in Fig. 10. Each unit separates the input mixture into two streams as indicated. The volumes of the outputs are fixed proportions of the input. Clearly, some of the units in the network

of Fig. 10 are redundant. The problem is to choose units and flow volumes so as to minimize fixed and variable costs, subject to capacity and volume constraints. Such problems can involve processes other than distillation and are often complicated by recycling of streams and waste heat, the latter typically resulting in a nonlinear model that is not discussed here. In some problems the volume of streams into and out of the network are semicontinuous variables.

Let E be the set of directed arcs in the network. The network contains a set I of unit nodes, which represent processing units, and a set J of structural nodes, at which no unit is present and flow is conserved. The flow on arc (i,j) is x_{ij} and incurs a unit cost of c_{ij}, typically negative on output flows to indicate revenue. The fixed cost of unit i is f_i and its capacity is k_i. Flow x_{ij} on arc (i,j) is α_{ij} times the total input to unit i.

If proposition y_i is true when unit i is installed, an MLLP model can be written as

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_i z_i \\
\text{s.t.} \quad & \left.\begin{array}{l}
\displaystyle\sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk}, \quad j \in J, \\
x_{ij} = \alpha_{ij} \displaystyle\sum_{(k,i)\in E} x_{ki}, \quad (i,j) \in E,\ i \in I, \\
0 \leqslant x_{ij} \leqslant k_i, \quad (i,j) \in E, \\
y_i \to (z_i = f_i), \quad i \in I, \\
y_i' \to \left(\begin{array}{l} z_i = 0 \\ \displaystyle\sum_{(i,j)\in E} x_{ij} = 0 \end{array} \right), \quad i \in I.
\end{array}\right| \quad y_i \vee y_i', \quad i \in I,
\end{aligned}$$

An MILP model is

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_i f_i y_i \\
\text{s.t.} \quad & \sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk}, \quad j \in J, \\
& x_{ij} = \alpha_{ij} \sum_{(k,i)\in E} x_{ki}, \quad (i,j) \in E,\ i \in I, \\
& 0 \leqslant \sum_{(i,j)\in E} x_{ij} \leqslant k_i y_i, \quad i \in I, \\
& y_i \in \{0,1\}, \quad i \in I.
\end{aligned} \tag{48}$$

Semicontinuous variables x_{ij} are given the logical representation,

$$\left.\begin{array}{l}
a_0 \leqslant x_{ij} \leqslant b_T, \\
y_t \to (x_{ij} \geqslant a_t), \\
y_t' \to (x_{ij} \leqslant b_{t-1}),
\end{array}\right| \quad y_t \vee y_t', \quad t = 1, \ldots, T, \tag{49}$$

and the MILP representation (2).

Elementary inequalities can be generated for the disjunctions $y_i \vee y_i'$ in (48). Because of upper and lower bounds on the variables, the corresponding constraint sets can be written as

$$y_i \rightarrow (z_i \geqslant f_i),$$
$$y_i' \rightarrow \begin{pmatrix} -z_i \geqslant 0 \\ -\sum_{(i,j)\in E} x_{ij} \geqslant 0 \end{pmatrix}.$$

This expands into two disjunctions that can be relaxed:

$$(z_i \geqslant f_i) \vee (-z_i \geqslant 0), \tag{50}$$

$$(z_i \geqslant f_i) \vee \left(-\sum_{(i,j)\in E} x_{ij} \geqslant 0 \right). \tag{51}$$

Because f_i is an upper bound on z_i, the elementary inequality (11) for (50) is simply $0 \geqslant 0$, which is useless. But the elementary inequality for (51) is

$$\frac{z_i}{f_i} \geqslant \frac{1}{M_i} \sum_{(i,j)\in E} x_{ij}, \tag{52}$$

where M_i is an upper bound on the flow out of unit i. This inequality is easily seen to define a facet of the convex hull of the disjunction.

There are also some useful quasi-valid logic constraints. Note in Fig. 10 that one should not install a distillation unit unless at least one adjacent upstream unit is installed, and all adjacent downstream units are installed. For example, unit 3 should not be installed unless unit 1 is installed, nor should unit 5 be installed unless both units 8 and 10 are present. This produces the logic constraints

$$y_3 \rightarrow y_1, \quad y_5 \rightarrow (y_8, y_{10}),$$

which can be written as three clauses,

$$y_1 \vee \neg y_3, \qquad \neg y_5 \vee y_8, \qquad \neg y_5 \vee y_{10}. \tag{53}$$

These constraints are nonvalid because there is nothing infeasible about installing a unit that carries no flow. One might suspect that a branch-and-bound search would not consider such spurious solutions, so that the constraints (53) would have no effect. Experience reported in [38,52], however, shows that they can be very effective, a fact that is confirmed here.

Although the linear relaxation of the MILP model can be duplicated with elementary inequalities, and quasi-valid logic constraints are available, there is reason to believe a logic-based approach is slightly worse than MILP. Once valid inequalities are added, the logic-based LP is actually slightly larger than the MILP model. The nonvalid logic constraints, although logically inspired, can be added to an MILP model. Furthermore, Theorem 4 implies that the 0–1 formulation of the disjunction $y_i \vee y_i'$ is integral. It is

Table 2
Node counts and computation times in seconds for separation network synthesis problems

Problem	MLLP	MLLP +dual ineq.	MLLP +elem ineq.	MILP	CPLEX	MLLP +elem. ineq. +logic constr.	MLLP +elem. ineq. +logic constr. +logic relax.	MILP +logic constr.	CPLEX +logic constr.
Node count									
5-component sep.	61	21	15	17	11	9	3	3	7
+ 4 unit restr.			15	49	29	13	3	3	4
6-component sep.	1659	105	97	191	94	63	97	33	40
+ 5 unit restr.			9	163	56	5	3	3	15
Seconds									
5-component sep.	0.91	3.39	0.41	0.31	0.33	0.35	0.40	0.18	0.40
+ 4 unit restr.			0.45	1.01	0.82	0.52	0.42	0.23	0.28
6-component sep.	33.3	26.5	2.3	5.6	3.5	2.6	8.1	3.3	3.5
+ 5 unit restr.			0.8	5.9	2.0	0.6	0.9	0.4	1.4

easily checked that if 0–1 variables y_1, y_2 correspond to the two disjuncts y_i, y_i', then $y_{12}(y_2) = y_{21}(y_1) = 0$.

Some of the synthesis problems are modified by fixing the number of units to be installed:

$$\sum_i y_i = k.$$

To generate elementary inequalities, the formula is written as two inequalities:

$$\sum_i y_i \geqslant k, \qquad \sum_i y_i' \geqslant k.$$

Elementary inequalities of the form (12) for these are respectively,

$$\sum_i z_i / f_i \geqslant k, \qquad \sum_{ij} x_{ij} / M_i \leqslant n - k,$$

where n is the number of potential units.

Experimental results for two 5-component and two 6-component problems studied in [52] are displayed in Table 2. The second 5-component problem fixes the total number of units to 4, and the second 6-component problem fixes it to 5. The solution methods are grouped by the strength of the formulation. The problems are first solved with pure MLLP branching, without any relaxation of the disjunctive constraints. The very poor results in the first column of the table indicate the importance of using relaxations. The next column illustrates the expense of generating dual inequalities, as discussed in Section 3.2.

Table 3
Node counts and computation times in seconds for 10-process and 38-process network synthesis problems

Problem	Nodes			Seconds		
	MLLP	MILP	CPLEX	MLLP	MILP	CPLEX
10 processes, version 1	5	29	24	0.24	0.82	0.65
10 processes, version 2	13	35	52	0.41	0.88	1.47
38 processes, version 1	729	1083	677	199	376	178
38 processes, version 2	1907	3237	868	559	1173	271
38 processes, version 3	1161	1999	345	306	836	104
38 processes, version 4	1901	2861	747	514	1093	229
38 processes, version 5	1081	1561	296	287	551	89

The next three columns of the table compare MLLP, MILP and CPLEX using relaxations that have the strength of the traditional continuous relaxation of the original problem; in the MLLP case, this requires the elementary inequalities (52). The next column adds the logic constraints described above to the MLLP model but not their relaxations. The last three columns add logic constraints to the MILP and CPLEX models and elementary relaxations of them to the MLLP model.

The results suggest that adding nonvalid logic constraints can bring a substantial improvement in an MILP context. They also reduce the number of nodes generated by the CPLEX MILP routine, which indicates that their employment does not merely duplicate the action of the CPLEX preprocessor. Experiments reported in [52] provide a similar indication for the OSL preprocessor. As predicted, MILP is slightly better than a logic-based approach.

The use of propositional variables is highly advantageous, however, for representing semi-continuous variables are added to the problem. As noted earlier, the 0–1 representation is nonintegral, and any continuous relaxation of it is useless.

The 10-process and a 38-process problem described in [57] were solved. All the valid constraints described above were used, except that relaxations for the nonvalid logic constraints were omitted from the logic-based model. The results appear in Table 3. The 10-process problem has 3 semicontinuous variables, and the 38-process problem has 7. Different versions of the problem were obtained by varying the time horizon and the placement of intervals.

The results show that a logical representation of semicontinuity roughly halves the computation time, even though semicontinuity accounts for only about half the discrete variables. A reasonable approach for these problems would therefore be to use the traditional approach for everything except the semicontinuous variables. The MLLP framework provides this kind of flexibility.

The CPLEX preprocessor eliminated most of the rows and columns of the 38-process problems (but not the 10-process problems) and therefore obtained superior performance on these problems. It is impossible to analyze this result without detailed knowledge of the preprocessor. Perhaps the operation that proved so effective could be added to the MLLP algorithm.

6.3. A warehouse location problem

A simple warehouse location problem is to choose a set of warehouses of limited capacity so as to serve a set of demand points while minimizing fixed and transport costs. Let

x_{ij}= flow from warehouse i to demand point j.
f_i= fixed cost of warehouse i.
k_i= capacity of warehouse i.
d_j= demand at point j.
c_{ij}= unit transport cost from i to j.

An MLLP model can be written as

$$\begin{aligned} \min \quad & \sum_i z_i + \sum_{ij} c_{ij}x_{ij} \\ \text{s.t.} \quad & \left.\begin{array}{l} \sum_j x_{ij} \leqslant k_i, \quad \text{all } i, \\ \sum_i x_{ij} \geqslant d_j, \quad \text{all } j, \\ z_i, x_{ij} \geqslant 0, \quad \text{all } i,j, \\ y_i \rightarrow (z_i = f_i), \quad \text{all } i, \\ y_i' \rightarrow \left(\sum_j x_{ij} = 0\right), \quad \text{all } i. \end{array}\right| \quad y_i \vee y_i', \quad \text{all } i, \end{aligned}$$

The traditional MILP model is

$$\begin{aligned} \min \quad & \sum_i f_i y_i + \sum_{ij} c_{ij}x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} \leqslant k_i y_i, \quad \text{all } i, \\ & \sum_i x_{ij} \geqslant d_j, \quad \text{all } j, \\ & x_{ij} \geqslant 0, \quad \text{all } i,j, \\ & y_i \in \{0,1\}, \quad \text{all } i. \end{aligned}$$

The formulation of elementary inequalities for the disjunctive constraints $y_i \vee y_i'$ is the same as in the network synthesis problems. The fact that total installed warehouse capacity must accommodate total demand gives rise to the valid knapsack constraint,

$$\sum_i k_i y_i \geqslant \sum_j d_j. \tag{54}$$

It can be viewed as a logical formula whose elementary relaxation can be added to the LP model:

$$\sum_i k_i(z_i/f_i) \geqslant \sum_j d_j.$$

Table 4
Node counts, computation times in seconds, and seconds per node for warehouse location problems

Problem	No.	Cap.	Nodes			Seconds			Seconds per node		
	whse	ratio	MLLP	MILP	CPLEX	MLLP	MILP	CPLEX	MLLP	MILP	CPLEX
CAP41	16	1.37	57	81	62	8.6	8.8	5.5	0.15	0.11	0.09
CAP41	16	1.29	59	81	57	8.9	8.6	5.5	0.15	0.11	0.10
CAP43	16	1.29	61	83	42	9.1	8.9	4.4	0.15	0.11	0.10
CAP44	16	1.37	43	61	40	7.1	6.8	4.3	0.17	0.11	0.11
CAP51	16	2.75	1239	1429	1134	172	135	92	0.14	0.09	0.08
CAP61	16	3.86	2147	2631	3017	266	237	235	0.12	0.09	0.08
CAP71	16	16.00	3481	4495	8830	409	398	658	0.12	0.09	0.07
1	10	6.12	61	147	31	1.21	0.70	0.57	0.020	0.015	0.018
2	10	5.10	63	45	25	1.34	0.67	0.52	0.021	0.015	0.021
3	10	4.08	71	73	59	1.54	1.14	1.17	0.022	0.016	0.020
4	10	3.06	49	173	138	1.11	2.61	2.50	0.023	0.015	0.018
5	10	2.04	19	31	27	0.45	0.45	0.55	0.024	0.015	0.020
6	10	1.02	3	21	20	0.16	0.30	0.32	0.053	0.014	0.016

1-cuts can be derived from (54) as described in Section 4.3, and their elementary relaxations added to the LP.

The 0–1 representation is again integral. The MLLP is also a little larger than the MILP model, because it contains elementary inequalities for the disjunctions, and furthermore because the MILP model combines the capacity constraints with the big-M constraints. The 1-cuts can be used in the MILP as well as the logic model. One would therefore expect an MILP formulation to have a small advantage.

Seven warehouse location problems from [8] were solved, and the results appear in Table 4. Each problem has 50 demand points with a total demand of 58,268. The number of warehouses is shown. Each warehouse has the same capacity, and the ratio of total warehouse capacity to total demand is shown as "Cap. ratio."

The 1-cuts were used in the MLLP model but not the MILP model. They result in a 20–30% reduction in the number of nodes but contributed to a 30–50% increase in the amount of time per node, because of they enlarge the LP model. The net result is that MLLP is slightly slower than MILP. The 1-cuts are therefore useful, but as predicted, one should use them in a traditional MILP relaxation.

Problems 1–6 in Table 4 were solved to test the hypothesis that 1-cuts have greater effect when the problem is more tightly constrained, as roughly indicated by the ratio of total warehouse capacity to total demand. The problems are identical except for the warehouse capacity. There are 7 demand points with demands 4,5,6,7,8,9,10. The data tend to confirm the hypothesis.

6.4. The progressive party problem

The final problem to be considered is a scheduling problem posed by a "progressive party" that was organized at a yachting rally in England. The problem gained some notoriety when a group of mathematical programmers and constraint programmers

found it to be intractable for the former and soluble by the latter, albeit with some manual intervention [62].

In a progressive party, the object is for the crews of a fleet of yachts to visit a subset of yachts and mingle with the other crews. The visiting crews move to different boats at the end of each phase of the party. Presumably to simplify the provision of refreshments and so forth, the number of host yachts should be small.

The problem can be more precisely defined as follows. A set I of boats is given. Each boat i occupied by a crew of c_i persons and has space for K_i persons on board. The problem is to minimize the number of host boats. Each crew i visits a different host boat h_{it} in each period t, unless it is itself a host, indicated by the truth of proposition δ_i. In the latter case $h_{it} = i$ for all t. To encourage mingling, no pair of visiting crews are permitted to meet more than once. The proposition m_{ijt} is true when non-host crews i and j visit the same boat in period t.

For checking capacity constraints it is convenient to define a proposition v_{ijt} that is true when $h_{it} = j$. The only propositions that enforce linear inequality constraints are the δ_i's, which force $z_i = 1$ when true.

An MLLP model can be written as follows. The objective function counts the number of host boats:

$$
\begin{array}{lll}
\min & \displaystyle\sum_{i\in I} z_i & \\
\text{s.t.} & z_i \geqslant 0, \quad i \in I, & \left|\begin{array}{ll}
v_{ijt} \equiv (h_{it} = j), & i,j \in I,\ t \in T, \quad \text{(a)} \\
\delta_i \vee \ \text{alldiff}(h_{i1},\ldots,h_{i|T|}), & i \in I, \quad \text{(b)} \\
\delta_i \equiv (h_{it} = i), & i \in I,\ t \in T, \quad \text{(c)} \\
\displaystyle\sum_{\substack{i\in I\\ i\neq j}} c_i v_{ijt} \leqslant K_j - c_j, & i \in I,\ t \in T, \quad \text{(d)} \\
\delta_i \vee \delta_j \vee m_{ijt} \vee (h_{it} \neq h_{jt}), & \\
\quad i,j \in I,\ i < j,\ t \in T, & \quad \text{(e)} \\
\displaystyle\sum_{t\in T} m_{ijt} \leqslant 1, & i,j \in I,\ i < j, \quad \text{(f)} \\
h_{it} \in \{1,\ldots,|I|\}, & i \in I,\ t \in T.
\end{array}\right. \qquad (55) \\
 & \delta_i \to (z_i \geqslant 1), \quad i \in I, &
\end{array}
$$

Formula (a) defines v_{ijt}. Formula (b) says that crew i should visit a different boat in each period unless it is a host crew. Formula (c) causes a crew to remain on its own boat if and only if it are a host crew. Knapsack constraint (d) is the boat capacity constraint. Formula (e) says that if crews i and j are both visiting crews (i.e., δ_i and δ_j are false), then either m_{ijt} is true or $h_{it} \neq h_{jt}$, i.e., m_{ijt} is true if the two crews visit the same boat in period t. The next formula (f) says that a pair of visiting crews should not meet more than once.

The entire model has $O(|I|^2|T|)$ variables and constraints. The LP is trivial, as it consists only of an objective function and constraints of the form $z_i \geqslant 1$. The LP will become more interesting when inequalities are added to strengthen the relaxation.

Formulation of an MILP model is more difficult. The most challenging constraint is the one that requires visiting crews to meet at most once. Smith et al. [62] remark that if this is formulated using the variables v_{ijt}, $O(|I|^4|T|^2)$ constraints are generated. Because this is impractical, they introduce $O(|I|^3|T|)$ variables y_{ijkt}, which take the value 1 when crews j,k meet on boat i in period t. But because there are 29 boats in the problem, this results in an enormous number of binary variables.

A more compact MILP model is suggested here. It reinterprets the multivalent variables h_{it} as numeric variables and enforces the all-different constraints with big-M constraints. The variables h_{it} need not be explicitly constrained to be integral, because the remaining constraints enforce integrality. The model has $O(|I|^2|T|)$ integer variables and constraints, many fewer than the model of [62].

The objective function (a) in the model below again counts the number of host boats. Constraints (b) and (c) require a crew to remain on their own boat if and only if they are a host crew. Constraints (d)–(f) use a disjunctive mechanism to relate v_{ijt} to h_{it}. They say that if $h_{it}=j$ (i.e., $\neg\alpha_{ijt}$ and $\neg\beta_{ijt}$, which say that h_{it} is neither less than nor greater than j), then $v_{ijt}=1$. Constraint (h) plays the role of the all-different constraint. Constraints (i)–(k) again use a disjunctive mechanism to say that if i and j are visiting crews and $h_{it}=h_{jt}$, then $m_{ijt}=1$:

$$
\begin{aligned}
\min\quad & \sum_{i\in I}\delta_i && \text{(a)}\\
\text{s.t.}\quad & \delta_j+(1-v_{ijt})\geqslant 1, \quad i,j\in I,\ t\in T, && \text{(b)}\\
& (1-\delta_i)+v_{iit}\geqslant 1, \quad i\in I,\ t\in T, && \text{(c)}\\
& v_{ijt}+\alpha_{ijt}+\beta_{ijt}\geqslant 1, \quad i,j\in I,\ t\in T, && \text{(d)}\\
& -h_{it}+j\geqslant 1-|I|(1-\alpha_{ijt}), \quad i,j\in I,\ t\in T, && \text{(e)}\\
& h_{it}-j\geqslant 1-|I|(1-\beta_{ijt}), \quad i,j\in I,\ t\in T, && \text{(f)}\\
& \sum_{\substack{i\in I\\ i\neq J}} c_i v_{ijt}\leqslant K_j-c_j, \quad j\in I,\ t\in T, && \text{(g)}\\
& \sum_{t\in T} v_{ijt}\leqslant 1, \quad i,j\in I,\ i\neq j, && \text{(h)}\\
& \delta_i+\delta_j+m_{ijt}+\phi_{ijt}+\psi_{ijt}\geqslant 1, \quad i,j\in I,\ i<j,\ t\in T, && \text{(i)}\\
& -h_{it}+h_{jt}\geqslant 1-|I|(1-\phi_{ijt}) \quad i,j\in I,\ i<j,\ t\in T, && \text{(j)}\\
& h_{it}-h_{jt}\geqslant 1-|I|(1-\psi_{ijt}) \quad i,j\in I,\ i<j,\ t\in T, && \text{(k)}\\
& \sum_{t\in T} m_{ijt}\leqslant 1, \quad i,j\in I,\ i<j, && \text{(l)}\\
& 1\leqslant h_{it}\leqslant |I|, \quad \delta_i,v_{ijt},\alpha_{ijt},\beta_{ijt},m_{ijt},\phi_{ijt},\psi_{ijt}\in\{0,1\},\ \text{all } i,j,t. &&
\end{aligned}
\tag{56}
$$

An alternate form of this model replaces constraints (d)–(f) with the constraints

$$h_{it}=\sum_j jv_{ijt},$$

Table 5
Node counts, computation times in seconds, and seconds per node for the progressive party problem

Boats	Periods	Nodes		Seconds		Seconds per mode	
		MLLP	CPLEX	MLLP	CPLEX	MLLP	CPLEX
5	2	171	1136	0.98	197.0	0.006	0.173
6	2	239	5433	1.41	1708	0.006	0.314
6	3	37	366	0.25	162.8	0.007	0.445
7	3	71	44	0.52	44.6	0.007	1.014
8	3	209	2307	1.63	3113	0.008	1.349
8	4	167	582	1.35	887.5	0.008	1.525
10	3	1143,973	20,000[a]	12,259	54,150[a]	0.011	2.708
10	4	28,923	20,000[a]	319	43,223[a]	0.011	2.161

[a]Computation was terminated after 20,000 nodes, without finding an integer solution.

resulting in a somewhat smaller formulation. In preliminary computational tests, this sometimes improved and sometimes worsened solution time.

The logic-based formulation was augmented with a simple logic constraint that requires the number of host boats to be no less than the number of periods:

$$\sum_{i\in I} \delta_i \geqslant |T|. \tag{57}$$

This was represented by an elementary inequality in the LP relaxation at each node. As in the warehouse location problem, there is a valid knapsack constraint that ensures there is enough capacity to meet total demand:

$$\sum_{i\in I} K_i\delta_i \geqslant \sum_{i\in I} c_i. \tag{58}$$

An elementary inequality for this was added to the LP relaxation. 1-cuts were also generated for (58) and their relaxations added to the LP. Elementary inequalities were not generated for the knapsack constraints (55d). The logic processing was achieved by a section of code that in effect implements the unit resolution algorithms of Figs. 6 and 7.

The MILP model was also augmented with the logic constraints (57). There was no need to add (58) because it is a linear combination of the other constraints.

The logic-based model (55) is not only simpler but has a substantial computational advantage. This is primarily because of the huge number of discrete variables, which are more efficiently processed in the logical part of the problem.

The computational results appear in Table 5.Due to the difficulty of the problem, only the CPLEX implementation of MILP was used. It was run with a feature that identifies specially ordered sets (sosscan), because MLLP's processing of propositional variables that are not associated with linear constraint sets can be viewed as incorporating the advantage of using type 1 specially ordered sets.

The original problem described in [62] had 29 boats and 6 periods and was solved by the ILOG Solver, but only after specifying exactly which boats were to serve

as hosts, and even then only after manual intervention. Smith et al. [62] report that XPRESSMP solved an MILP model of the problem with 15 boats and 4 periods, but only after specifying that only boats 1–8 (in descending order of $K_i - c_i$) could serve as hosts and only crews 5–15 could visit other boats (the optimal solution uses 5 hosts). The problems were solved here without any manual intervention. When the problem contains $|I|$ boats, they are the $|I|$ largest boats as measured by $K_i - c_i$.

Both solution methods could no doubt be improved with more intelligent branching and other devices. But the underlying computational advantage of a logical representation is clear and is due primarily to a much smaller LP relaxation and the speed of logic processing.

7. Conclusions

We conclude that the larger repertory of modeling and solution options in MLLP can, if judiciously chosen, provide a more flexible modeling environment than MILP alone, without sacrificing solution efficiency and in some cases substantially improving it.

We attempt here to collect some rough guidelines for choosing the options, based on computational experience to date. The basic issue addressed is what part of the constraint set should be given a logical formulation, and what part should be embedded in the linear model with the help of integer variables. It is assumed throughout that constraints with purely continuous variables appear in the continuous portion of the model.

Because the formulas p_j of (1) all have the form y_j in the problems solved, the discussion here assumes that they have this form. In general some propositions y_j enforce one or more linear inequality constraints when they are true, and others do not. It is convenient to refer to the former as *linked* and the latter as *unlinked.*

- As a general rule, constraints should receive the most convenient formulation, unless one of the considerations to follow indicates otherwise. For example, a 0–1 knapsack constraint $ax \geqslant \alpha$ (where some coefficients a_j are other than $0, 1, -1$) should be written as a 0–1 inequality. Constraints with a logical flavor, however, should normally be appear in logical form. These might include disjunctions, implications, etc.
- Constraints that contain primarily unlinked propositions in their logical form should normally be written in that form. It is likely that logic processing is as useful as solving the linear relaxation of the 0–1 formulation; it is equivalent when the logical formulas are clauses. The advantage of a logical formulation can be substantial when there are a large number of unlinked propositions, as illustrated by the progressive party problem.
- For constraints that contain primarily linked propositions in their logical formulation, the best treatment depends on the nature of the relaxation.

- If there is no good linear relaxation, as in the case of the flow job scheduling disjunctions and the semicontinuous variables discussed earlier, then the constraints should receive a logical formulation. In this case the overhead of using integer variables is unjustified.
- If a good linear relaxation exists, the following considerations apply. If the continuous relaxation of the 0–1 formulation can be duplicated or improved upon using a small number of (strengthened) elementary cuts or other cuts, then the logical formulation should be used, and the cuts added to the LP. Examples of this were given in Sections 3.3 and 3.4. If the cuts required to duplicate the continuous relaxation are too numerous or unavailable, then the constraints should receive a traditional 0–1 formulation. This was the situation in the warehouse location and process network design problems (aside from the semicontinuous variables in the latter).
- It may be advantageous to use both a logical and a 0–1 formulation, so as to apply logic processing to the former and obtain a relaxation from the latter.
- If the 0–1 representation is nonintegral, this argues against it, because branching may continue unnecessarily. The flow shop problems and semicontinuous variables serve as examples.

- Any logic cuts (valid or nonvalid) that can be identified are probably useful. Examples of these include the nonvalid cuts generated for the network design problems and the 1-cuts used in the warehouse and party problems. Logic cuts can be represented in logical form, 0–1 form, or both, depending on factors already discussed.
- Optimal separating cuts have not been tested computationally, but the success of separating cuts and lift-and-project cuts (to which they are analogous) suggests that they could be useful. It may also be beneficial to use Benders cuts, which can be generalized to a logic-based setting [34].
- It may be possible to construct useful linear relaxations of common logical formulas that contain multivalued discrete variables (other than integer variables), such as all-different constraints. This issue is now under investigation.

A software package based on MLLP would probably require more expertise that existing ones based on MILP. Ultimately, however, a large class of combinatorial problems may always require a certain amount of expertise for their solution. The issue is how much user intervention is appropriate. It seems unreasonable to restrict oneself to automatic routines in general-purpose solvers when some simple additional devices may obtain solutions that are otherwise out of reach. At the other extreme, it is impractical to invest in every new problem the years of research effort that have been lavished on traveling salesman and job shop scheduling problems. MLLP is designed to present a compromise between these two extremes.

References

[1] A. Aiba, K. Sakai, Y. Sato, D.J. Hawley, R. Hasegawa, Constraint logic programming language CAL, Fifth Generation Computer Systems, Springer, Tokyo, 1988.

[2] E. Balas, Disjunctive programming: cutting planes from logical conditions, in: O.L. Mangasarian, R.R. Meyer, S.M. Robinson (Eds.), Nonlinear Programming, vol 2, Academic Press, New York, 1975, pp. 279–312.
[3] E. Balas, A note on duality in disjunctive programming, J. Optim. Theory Appl. 21 (1977) 523–527.
[4] E. Balas, Disjunctive programming, Ann. Discrete Math. 5 (1979) 3–51.
[5] E. Balas, S. Ceria, G. Cornuéjols, Mixed 0–1 programming by lift-and-project in a branch and cut framework, Management Sci. 42 (1996) 1229–1246.
[6] P. Barth, Logic-Based 0–1 Constraint Programming, Kluwer Academic Publishers, Boston, 1995
[7] N. Beaumont, An algorithm for disjunctive programs, European J. Oper. Res. 48 (1990) 362–371.
[8] J.E. Beasley, An algorithm for solving large capacitated warehouse location problems, European J. Oper. Res. 3 (1988) 314–325.
[9] C. Blair, Two rules for deducing valid inequalities for 0–1 problems, SIAM J. Appl. Math. 31 (1976) 614–617.
[10] C. Blair, R.G. Jeroslow, J.K. Lowe, Some results and experiments in programming techniques for propositional logic, Comput. Oper. Res. 13 (1988) 633–645.
[11] S. Bollapragada, O. Ghattas, J.N. Hooker, Optimal design of truss structures by mixed logical and linear programming, to appear in Operations Research.
[12] I. Bratko, PROLOG Programming for Artificial Intelligence, International Computer Science, Addison-Wesley, Reading, MA, 1986.
[13] BULL Corporation, CHARME VI User's Guide and Reference Manual, Artificial Intelligence Development Centre, BULL S.A., France, 1990.
[14] V. Chandru, C.R. Coullard, P.L. Hammer, M. Montañez, X. Sun, On renamable Horn and generalized Horn functions, Ann. Math. AI 1 (1990) 33–48.
[15] V. Chandru, J.N. Hooker, Extended Horn clauses in propositional logic, J. ACM 38 (1991) 205–221.
[16] V. Chandru, J.N. Hooker, Detecting embedded Horn structure in propositional logic, Inform. Process. Lett. 42 (1992) 109–111.
[17] A. Colmerauer, An introduction to Prolog III, Comm. ACM 33 (1990) 52–68.
[18] A. Colmerauer, H. Kanouia, R. Pasero, P. Roussel, Un système de communication homme-machine en français, Technical Report, Université d'Aix-Marseilles II, Groupe intelligence artificielle, 1973.
[19] M.C. Cooper, An optimal k-consistency algorithm, Artif. Intell. 41 (1989) 89–95.
[20] M. Dincbas, P. Van Hentenryck, H. Simonis, A. Aggoun, T. Graf, F. Bertier, The constraint programming language CHIP, Proceedings on the International Conference on Fifth Generation Computer Systems FGCS-88, Tokyo, December 1988.
[21] A. Drexl, C. Jordan, A comparison of logic and mixed-integer programming solvers for batch sequencing with sequence-dependent setups, INFORMS J. Comput., to appear.
[22] E.C. Freuder, Exploiting structure in constraint satisfaction problems, in: B. Mayoh, E. Tyugu, J. Penjam (Eds.), Constraint Programming, Springer, Berlin, 1993, pp. 50–74.
[23] F. Glover, Surrogate constraint duality in mathematical programming, Oper. Res. 23 434–451.
[24] F. Granot, P.L. Hammer, On the use of boolean functions in 0–1 linear programming, Methods Oper. Res. (1971) 154–184.
[25] A. Haken, The intractability of resolution, Theoret. Comput. Sci. 39 (1985) 297–308.
[26] P.L. Hammer, S. Rudeanu, Boolean Methods in Operations Research and Related Areas, Springer, Berlin, 1968.
[27] J.N. Hooker, Resolution vs. cutting plane solution of inference problems: some computational experience, Oper. Res. Lett. 7 (1988) 1–7.
[28] J.N. Hooker, Generalized resolution and cutting planes, Ann. Oper. Res. 12 (1988) 217–239.
[29] J.N. Hooker, A quantitative approach to logical inference, Decision Support Systems 4 (1988) 45–69.
[30] J.N. Hooker, Input proofs and rank one cutting planes, ORSA J. Comput. 1 (1989) 137–145.
[31] J.N. Hooker, Generalized resolution for 0–1 linear inequalities, Ann. Math. AI 6 (1992) 271–286.
[32] J.N. Hooker, Logical inference and polyhedral projection, Proceedings, Computer Science Logic Workshop (CSL'91), Lecture Notes in Computer Science, vol. 626, Springer, Berlin, 1992, pp. 184–200.
[33] J.N. Hooker, Logic-based methods for optimization, in: A. Borning (Ed.), Principles and Practice of Constraint Programming, Lecture Notes in Computer Science, vol. 874, Springer, Berlin, 1994, pp. 336–349.

[34] J.N. Hooker, Logic-based Benders decomposition, available at http://www.gsia.cmu.edu/afs/andrew/gsia/jh38/jnh.html, 1995.
[35] J.N. Hooker, Testing heuristics: we have it all wrong, J. Heuristics 1 (1995) 33–42.
[36] J.N. Hooker, Constraint satisfaction methods for generating valid cuts, in: D.L. Woodruff, ed., Advances in Computational and Stochastic Optimization, Logic Programming and Heuristic Search, Kluwer (1997) pp. 1–30. edu/afs/andrew/gsia/jh38/jnh.html.
[37] J.N. Hooker, N.R. Natraj, Solving 0–1 optimization problems with k-tree relaxation, in preparation.
[38] J.N. Hooker, H. Yan, I. Grossmann, R. Raman, Logic cuts for processing networks with fixed charges, Comput. Oper. Res. 21 (1994) 265–279.
[39] J. Jaffar, J.-L. Lassez, Constraint logic programming, Proceedings of the 14th ACM Symposium on Principles of Programming Languages, München, ACM, New York, 1987, pp. 111–119.
[40] J. Jaffar, J.-L. Lassez, From unification to constraints, Logic programming 87, Proceedings of the Sixth Conference, Springer, Berlin, 1987, pp. 1–18.
[41] R.E. Jeroslow, Representability in mixed integer programming, I: characterization results, Discrete Appl. Math. 17 (1987) 223–243.
[42] R.E. Jeroslow, Logic-Based Decision Support: Mixed Integer Model Formulation, Annals of Discrete Mathematics, vol. 40, North-Holland, Amsterdam, 1989.
[43] R.E. Jeroslow, J.K. Lowe, Modeling with integer variables, Math. Programm. Stud. 22 (1984) 167–184.
[44] R.A. Kowalski, Predicate logic as programming language, Proceedings of the IFIP Congress, North-Holland, Amsterdam, 1974, pp. 569–574.
[45] K. McAloon, C. Tretkoff, 2LP: linear programming and logic programming, in: P. van Hentenryck, V. Saraswat (Eds.), Principles and Practice of Constraint Programming, MIT Press, Cambridge, MA, 1995, pp. 99–114.
[46] K. McAloon, C. Tretkoff, Optimization and Computational Logic, Wiley, New York, in preparation.
[47] J. Piehler, Ein Beitrag zum Reihenfolgeproblem, Unternehmenforschung 4 (1960) 138–142.
[48] J.-F. Puget, A C++ implementation of CLP, Technical Report 94-01, ILOG S.A., Gentilly, France, 1994.
[49] W.V. Quine, The problem of simplifying truth functions, Amer. Math. Mon. 59 (1952) 521–531.
[50] W.V. Quine, A way to simplify truth functions, Amer. Math. Mon. 62 (1955) 627–631.
[51] R. Raman, I.E. Grossmann, Relation between MILP modeling and logical inference for chemical process synthesis, Comput. Chem. Eng. 15 (1991) 73–84.
[52] R. Raman, I.E. Grossmann, Symbolic integration of logic in MILP branch and bound methods for the synthesis of process networks, Ann. Oper. Res. 42 (1993) 169–191.
[53] R. Raman, I.E. Grossmann, Symbolic integration of logic in mixed-integer linear programming techniques for process synthesis, Comput. Chem. Eng. 17 (1993) 909–927.
[54] R. Raman, I.E. Grossmann, Modeling and computational techniques for logic based integer programming, Comput. Chem. Eng. 18 (1994) 563–578.
[55] C. Remy, Programming by constraints, Micro Systemes 104 (1990) 147–150.
[56] J.A. Robinson, A machine-oriented logic based on the resolution principle, J. ACM 12 (1965) 23–41.
[57] N.V. Sahinidis, I.E. Grossmann, R.E. Fornari, M. Chathrathi, Optimization model for long range planning in the chemical industry, Comput. Chem. Eng. 13 (1989) 1049–1063.
[58] J.S. Schlipf, F.S. Annexstein, J.V. Franco, R.P. Swamination, On finding solutions for extended Horn formulas, Inform. Process. Lett. 54 (1995) 133–137.
[59] D. Sciamma, J. Gay, A. Guillard, CHARME: a constraint oriented approach to scheduling and resource allocation, Artificial Intelligence in the Pacific Rim, Proceedings of the Pacific Rim International Conference on Artificial Intelligent, Nagoya, Japan, 1990, pp. 71–76.
[60] C.-L. Sheng, Threshold Logic, Academic Press, New York, 1969.
[61] H. Simonis, M. Dincbas, Propositional calculus problems in CHIP, in: F. Benhamou, A. Colmerauer (Eds.), Constraint Logic Programming: Selected Research, MIT Press, Cambridge, MA, 1993, pp. 269–285.
[62] B.M. Smith, S.C. Brailsford, P.M. Hubbard, H.P. Williams, The progressive party problem: Integer linear programming and constraint programming compared, in: U. Montanari, F. Rossi (Eds.), Proceedings of Principles and Practice of Constraint Programming, Cassis, France, Springer, Berlin, 1995, pp. 36–52.
[63] L. Sterling, E. Shapiro, The Art of Prolog: Advanced Programming Techniques, MIT Press, Cambridge, MA, 1986.

[64] R.P. Swaminathan, D.K. Wagner, The arborescence-realization problem, Discrete Appl. Math. 59 (1995) 267–283.
[65] E. Tsang, Foundations of Constraint Satisfaction, Academic Press, London, 1993.
[66] M. Turkay, I.E. Grossmann, Logic-based MINLP algorithms for the optimal synthesis of process networks, Comput. Chem. Eng. 20 (1996) 959–978.
[67] P. Van Hentenryck, Constraint Satisfaction in Logic Programming, MIT Press, Cambridge, MA, 1988.
[68] H.P. Williams, Fourier-Motzkin elimination extension to integer programming problems, J. Combin. Theory 21 (1976) 118–123.
[69] H.P. Williams, Logical problems and integer programming, Bull. Inst. Math. Implications 13 (1977) 18–20.
[70] H.P. Williams, Linear and integer programming applied to the propositional calculus, Int. J. Systems Res. Inform. Sci. 2 (1987) 81–100.
[71] H.P. Williams, An alternative explanation of disjunctive formulations, European J. Oper. Res. 72 (1994) 200–203.
[72] H.P. Williams, Logic applied to integer programming and integer programming applied to logic, European J. Oper. Res. 81 (1995) 605–616.
[73] J.M. Wilson, Compact normal forms in propositional logic and integer programming formulations, Comput. Oper. Res. 90 (1990) 309–314.
[74] J.M. Wilson, Generating cuts in integer programming with families of specially ordered sets, European J. Oper. Res. 46 (1990) 101–108.
[75] J.M. Wilson, A note on logic cuts and valid inequalities for certain standard (0–1) integer programs, manuscript, Loughborough University Business School, Loughborough, Leicestershire LZE11 3TU, UK, 1995.