NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Mixed Logical/Linear Programming

John N. Hooker and Maria A. Osorio

EDRC 7M0-96

Mixed Logical/Linear Programming *

J. N. HOOKER

Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213 USA

M. A. OSORIO

School of Chemical Engineering
University of Puebla, Puebla, Mexico 72550

July 1996

Abstract

Mixed logical/linear programming (MLLP) is an extension of mixed integer/linear programming (MILP). It represents 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 and disadvantages of MLLP 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. Mixed integer/linear programming (MILP), the traditional approach, is effective in many instances. But it unnecessarily restricts the modeling and solution options available. MLLP allows naturally for branching strategies, relaxations and logic processing algorithms that neither fit comfortably into nor are suggested by MILP. In particular it counts the traditional strategies among ii> options and should therefore be seen as an extension of MILP rather than an alternative to it.

Mixed discrete/continuous problems are traditionally conceived as continuous problem* in which some of the variables are restricted to be integers. MLLP takes a completely different view. It does not attempt an often unnatural and contrived embedding of the discrete aspects of the problem within a linear programming model. Instead, it represents the discrete elements by logical formulas and only the continuous element by linear inequalities. It therefore has the option of dispensing with integer variables. Rather than require that a feasible solution satisfy a fixed set of inequalities, MLLP provides several alternative sets of inequalities. The role of the logical formulas is to govern which alternatives are acceptable.

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

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

$$\begin{aligned} &\min \quad ex \\ &\text{s.t.} \quad {}_{Vi} - (A'\dot{x} >\_a'), \quad j \in J \qquad {}_{gi}\{y,h), \quad i \in I. \end{aligned} \qquad (1)$$

The model has a logical part (on the right) and a continuous part (on the left). The logical part consists of formulas $\{y, h)$ that involve *atomic propositions* $y = (y_1, \ldots, y_n)$, which are either true or false. Such a formula might be $y\ \vee\ y<i$, which says that y or $y<i$ (or both) must be true. There may also be some variables $h = (h_1, \ldots, h_m)$ that take several discrete values. The continuous part associates each atomic proposition yj with a system $A^x \geq a^*$ of linear inequalities. The system is enforced when yj is true. So the formula $y\ \vee\ y<i$ in effect requires any solution x to satisfy $A^1x \geq a^1$ or $A^2x \geq a^2$ (or both). In general, $(x,y,/i)$ is feasible if (y, h) makes all the logical formulas true and x satisfies the linear systems corresponding to true j/j's.

The problem (1) can be solved by branching on the truth values of the y/s and the discrete values of the $hj's$. At each node of the search tree, one solves a linear programming problem (LP) containing the constraints that correspond to true y/s, plus any cuts 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 further logical constraints and may fix some additional y/s and $hj's$.

It is easy to show that every MLLP is equivalent to a disjunctive programming problem whose constraint is a disjunction of linear systems (i.e., the solution must satisfy at least one of the systems). Its feasible set is therefore a union of finitely many polyhedra.

1.2 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 framework. The potential advantages of MLLP are articulated, and several are illustrated by computational experiments.

Because MLLP is a general approach to continuous/discrete problem solving, a thoroughgoing 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, cuts, 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., [10, 64]) suggests that MLLP can be usefully extended to mixed logical/nonlinear programming (MLNLP). This possibility is not pursued here.

1.3 Advantages of MLLP

Several potential advantages of MLLP's broader framework, summarized below, should become evident in the ensuing discussion. Some of the advantages are illustrated by the computational experiments, as indicated.

- ***Separation of branching and relaxation.*** **In MILP, integer variables serve as both a branching and relaxation device. The problem is relaxed by removing the integrality constraint on the variables, and one branches on integer variables that have a fractional value in the solution of the relaxation. But there are many other branching and relaxation schemes. MLLP permits one to use any branching scheme in combination with any relaxation, because the relaxation need not involve the discrete variables. The relaxation is generally created by adding cuts to the continuous part of the model.**

- ***Smaller linear programming problems.*** **Some combinatorial problems are formulated only or most conveniently with a large number of discrete variables. MILP forces these variables to be included in the LP problems solved at each node, whereas MLLP puts only continuous variables in the LP problems. This advantage is dramatically illustrated by the party problem. In some cases, however, the advantage of removing integer variables is offset by the necessity of adding cuts to achieve a good relaxation. This is discussed below.**

- ***Faster and/or more effective processing of discrete variables.*** **Logical variables can often be fixed at a node by applying inference procedures to the logical constraints. In many cases the same variables would have been fixed by solving the conventional continuous relaxation. But in these cases logic processing is much faster because it can be achieved by a unit resolution procedure. This is illustrated by the party problem. In other cases, logic processing is more powerful than linear programming, because it fixes variables to true or false when they would receive fractional values in a continuous relaxation.**

- ***Feasible solutions identified sooner.*** **The conventional continuous relaxation of a 0-1 disjunctive formulation can have fractional solution values even when the disjunction is satisfied. This means that MILP can keep branching after a feasible solution is found. MLLP avoids this defect and may therefore produce a smaller search tree. This is illustrated by the flow shop problem and instances of the process synthesis problem that have semicontinuous variables.**

- ***Stronger relaxations.*** **MLLP can provide stronger relaxations in three ways.**

  - **Valid cuts can sometimes be found for a disjunctive constraint that are stronger than the continuous relaxation of any obvious 0-1 formulation.**
  - **Logic processing can generate logical constraints *(logic cuts)* whose linear relaxations can be added to the linear model. The same logic cuts can be used as cutting planes in a conventional branch-and-cut algorithm, but logic processing provides a systematic way of generating them. This is illustrated by the warehouse location and party problems.**
  - **The idea of Benders cuts can be generalized to an MLLP setting to produce logic-based cuts that may be useful when solution of LP relaxations is expensive.**

- *An alternative approach to identifying cuts.* The logical point of view can suggest logic cuts that follow easily from an intuitive understanding of the problem but are not easily revealed by conventional polyhedral analysis. It can also suggest *nonvalid* logic cuts, which do not change the optimal value but are stronger because they exclude feasible solutions. The chemical processing network problems will illustrate both points.

- *More natural modeling.* Integer variables, particularly 0-1 variables, are very often contrived to express what were originally conceived as logical constraints. In such cases MLLP permits a more natural formulation. In other cases a problem is most readily expressed with multivalued discrete variables, as for example problems in which a variable's value may indicate the machine to which a job is assigned or an operation's position in a sequence. Constraints on such variables that are not easily expressed in inequality form may readily be formulated with predicates that are quite natural for logic processing algorithms. A particularly useful predicate is the "all different" predicate, which requires that a set of variables all have different values. The advantages of multivalued variables have been exploited in the constraint satisfaction and constraint programming literature, and the algorithms developed there may be used in the logic processing phase of an MLLP solver.

The last point deserves expansion. A habit of writing models for the convenience of the solver, as is common in operations research, can lead one to forget that the primary role of modeling in science is explanatory. The modeling process should improve one's understanding and should not be befuddled by the necessity of writing a machine-friendly formulation. On the other hand, if the model is totally unsuited for solution, it will require a substantial reformulation either by an automatic procedure or a human expert. It may be difficult to relate the solution to the original formulation. MLLP tries to steer a middle course. It makes discrete/continuous modeling somewhat more natural by eliminating the necessity of integer variables and introducing logical constraints and multivalent variables. But the solution approach is still closely related to the problem statement because it branches on the discrete variables that appear in the original model. The model generally requires additional preparation before solution, but preparation usually involves adding cuts rather than reformulating the problem.

1.4 Mitigating Factors

The advantages of moving to MLLP's broader framework are mitigated by at least three considerations.

- *Relaxations must be explicitly generated.* An MILP model comes with a ready-made linear relaxation, but **MLLP** obliges one to make a conscious choice of relaxation. It is sometimes unobvious how to create a relaxation that is even as strong as the traditional one (without reintroducing the traditional integer variables).

- *The relaxation may be large.* The cuts needed to build a useful relaxation may make the linear constraint set larger than an MILP model.

- *More expertise is needed.* MLLP generally requires greater expertise than MILP because of the greater range of choices it imposes on the user. It may therefore be suited to a smaller circle of practitioners.

One might respond to the first two points as follows. Although a nontraditional relaxation may be large or hard to identify in some cases, there are other cases in which it can solve an otherwise intractable problem. The key is to be able to identify the appropriate relaxation for a given constraint set. The following cases may be distinguished and are more rigorously characterized in the remainer of the paper.

1. ***The constraint(s) to be relaxed can have no good linear relaxation,*** **because the convex** hull of the feasible set occupies most or all of the solution space. In these cases no relaxation should be used. MLLP permits this, whereas MILP forces one to use integer variables, which introduce useless overhead.

2. ***A good relaxation is possible but the MILP relaxation is weak or useless.*** **It may be** possible to replace the MILP relaxation with a reasonable number of cuts that produce a stronger relaxation. Two general methods, described below, that may be helpful are tightening of MILP cuts and generation of "optimal separating cuts."

3. *The MILP relaxation is useful.* In this case it may be easy to mimic the effect of the MILP relaxation with a few cuts. If not, one can always add integer variables to obtain the classical relaxation. Even here, one may wish to relax only a portion of the model in this fashion. The integer variables need not be used for branching purposes, so that the distinction between branching and relaxation is maintained.

The final objection, that MLLP requires more expertise, can be partially overcome by automating as many choices as possible and by installing redundancy. Commercial MILP solvers, for example, automatically apply a number of cuts and preprocessing devices that may or may not be useful for a given problem.

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 tricks 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.

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 [25]. Granot and Hammer [23] 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], 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 [42], 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 [66, 67, 68, 70], Blair [8, 9] and Hooker [28, 28, 29, 30, 31] explored connections betwen logic and optimization. Beaumont [6] 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, 3, 4], he described families of valid cuts that can be used to create relaxations of disjunctive constraints.

More recently, Hooker argued in [32] that a logic-based approach to optimization, including MLLP, can exploit problem structure in ways that are parallel to traditional polyhedral techniques. Wilson [71, 72, 73] studied logic cuts and 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 [37, 49, 50, 51, 52, 64]. These papers developed some of the key MLLP concepts discussed here. Bollapragada, Ghattas and Hooker also obtained encouraging results in structural design [10].

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 [44, 45], 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 [5] 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 [16] and Kowalski [43], allow one to formulate a problem in a subset of first-order logic (Horn clause logic). Recent versions of the logic programming language PROLOG [11, 61], 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 [20, 63, 65]. A number of systems along this line have been developed in addition to Prolog III, including CLP(R) [39], CAL [1], CHIP [19, 59], the ILOG solver [46], and other packages [12, 57, 53]. 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 cuts and relaxations, although the constraint programming literature has shown some interest in exploiting structure (e.g., [21]).

1.7 Outline of the Paper

The remainder of the paper begins with a presentation of four example problems that illustrate MLLP modeling (Section 2). Section 3 presents the disjunctive interpretation of MLLP, and Section 4 summarizes the basic MLLP algorithm. Two long sections (5 and 6) respectively discuss relaxations and logic processing algorithms. Some of these are illustrated in the last section, which presents computational results for the four example problems.

Aside from its survey and development of MLLP generally, the specific contributions of this paper include necessary and sufficient conditions for whether an elementary cut for a disjunction is supporting (Section 5.4), necessary and sufficient conditions for integrality of a 0-1 disjunctive representation (Section 5.5), a definition of optimal separating cuts (Section 5.7), a completeness proof for multivalent resolution (Section 6.6), a unit resolution algorithm for multivalent clauses (Section 6.6), and an application of logic-based Benders decomposition to MLLP (Section 6.8).

2 Some Examples

Examples from four application areas are presented to illustrate some of the concepts, advantages and disadvantages of MLLP. The first two problem classes concern flow shop scheduling with zero-wait transfer and processing network design. They are important in chemical engineering. The third, a warehouse location problem, was chosen partly because it is ill-suited to logic modeling. The fourth problem, the progressive party problem, is chosen to represent problems in which the discrete element dominates. An attempt was made to choose problems with the flavor or complexity of real applications, although the warehouse location problem is somewhat stylized.

2.1 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 $J\{$ be the set of machines on which job i is processed, and $d_{\dot{X}\dot{J}}$ the processing time for job i on machine j. If U is the start time for job t, the job is completed at time

$$u + \sum_{j \in J_i} \wedge$$

It is necessary to make sure that two jobs t, k are not scheduled to be in process at the same time on the same machine j € J,·n J*. The finish time of job i on machine j is $t_x \cdot + D_{\dot{X}\dot{J}}$, where

$$D_{ij} = \sum_{\substack{? \in Ji \\ i' < i}} d_{ij'},$$

and its start time is $t_{\dot{x}} + D_{t\dot{J}} - d_{\dot{x}}j$. 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. The natural

and obvious way to formulate this "or" constraint is with a logical disjunction. For each pair (i, fc), one writes the disjunction,

$$(U + Da_{-} \le t_k + D_{ik} - d_{ik},\ j \in JiD\ J_k)\ \mathbf{V}\ (t_k + D_{kj} \le t_l + Dij - d_{ih}\ j \in Ji\ \mathbf{n}\ J_k).$$

The inequalities in 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. The model to minimize makespan is therefore

$$\begin{aligned} \min\ & T \\ \text{s.t.}\ & T \ge t_i + \sum_{j \in J_i} d_{ij}, \quad \text{all jobs } i, \\ & (t_k - U > r_{ik}) \ \mathbf{V}\ (U - t_k > r_{ki}), \quad \text{all jobs } t, k \text{ with } t \pm \text{fc}, \\ & t \ge 0, \quad \text{allf}, \end{aligned}$$

where

$$r_{ik} = \max_{\in J_i \cap J_k} \{Dij - D_{ik} + d_{ik}\}.$$

This MLLP model is easily put into the form (1) by introducing logical variables. Let *yik* be true when job i is sdieduled before job fc on each machine on which they are both processed. The formal MLLP is therefore

$$\begin{aligned} \mathbf{min}\ & \mathbf{T} \\ \mathbf{s.t.}\ & U \ge 0,\ T \ge U + \sum_{j \in J_i} dij, \ \text{all}\ \% \quad (a) \qquad y_{ik} \ \mathbf{V}\ y_{ki}, \ \text{all}\ i, k,\ i \ne k. \\ & y_{ik} \rightarrow (t_k - t_i \ge r_{ik}) \qquad (b) \end{aligned} \tag{2}$$

There is no need to represent constraints (a) with propositions t/t because they hold categorically.

Formulating (2) as a traditional MILP model requires the additional step of introducing big-M constraints to represent the disjunctions.

$$\begin{aligned} \min\ & T \\ \mathbf{s.t.}\ & U \ge 0,\ T \ge t_l + \sum_{j \in J_i} d_{ji} \ \text{ail}\ i \\ & t_k - t_i \ge r_{ik} - M(1 - y_{ik}) \\ & U - h \ge r_{ki} - My_{ik} \\ & y_{<ik} e \{0,1\}, \quad \text{allt, fc}. \end{aligned}$$

2.2 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. 1. Bach 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. 1 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

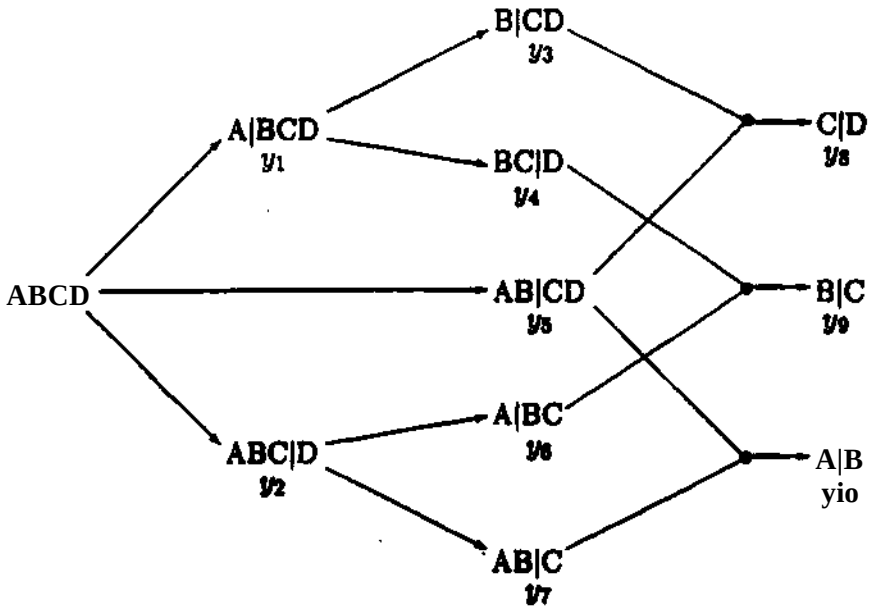

Figure 1: *A ^component separation network.*

often complicated by recycling of streams and waste heat, the latter typically resulting in a nonlinear model that is not discussed here. The volume of some streams into and out of the network can be semicontinuous variables, and this possibility is considered.

Let *E* be the set of directed arcs in the network. The network in general contains a set / of unit nodes, which represent processing units, and a set J of structural nodes, at which no unit is present and flow is simply conserved. The flow on arc (i,*j)* is x_{ij}- and incurs a unit cost of *Cij.* Inputs to the network and outputs from it are represented by structural nodes, and if j is an output node, cy would typically be negative to indicate revenue from sale of the product. The fixed cost of unit *i* is /». The flow on arc (i,*j)* leaving unit *j* is *a%j* times the total input to the unit.

The discrete element of the problem is the decision as to whether to install a unit *i* and its incident arcs. Either fixed cost f_x is incurred, or else there is no fixed cost and no flow leaves the unit. If *Z{* is the fixed cost paid for unit i,

$$(\text{« = k0}\mathbf{V}\mathbf{f}_{\mathbf{r}} \quad \text{*-,°.-0V}$$

If *yi* is true when unit i is installed, the MLLP model can be written,

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_i z_i \\
\text{S.t.} \quad & \sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk}, \quad j \in J \qquad\qquad y_i \vee y(, \quad \text{t 6 /} \\
& x_{ij} = \alpha_{ij} \sum_{(k,i)\in E} x_{ki}, \quad (i,j) \in E, \; i \in I \\
& 0 \le x_{ij} \le k_i, \quad (i,j) \in E \\
& y_i \rightarrow (z_i = f_i), \quad i \in I \\
& y_i' \rightarrow \begin{pmatrix} z_i = 0 \\ \sum_{(i,j)\in E} x_{ij} = 0 \end{pmatrix}, \quad i \in I.
\end{aligned}$$

The MILP model represents the disjunctions in the usual way.

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_i f_i y_i$$

$$\text{s.t.} \sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk}, \quad j \in J$$

$$x_{ij} = a_{ij} \qquad I$$

(W)€£

$$- \sum_{(i,j)\in E} x_{ij} \le k_i y_i, \quad i \in I$$

$$y_i \in \{0,1\}, \quad i \in I$$

Process synthesis problems can involve semicontinuous variables, as described in [52]. These are variables whose values must lie within certain intervals. For example, an input to the processing network may consist of a feedstock obtained from a small set of suppliers, each of which can provide either nothing or a quantity within a certain range. Taking all possible sums of these intervals yields the intervals within which the total input volume must fall.

A semicontinuous flow variable x_{ij} must lie in one of the intervals $[a_t, b_t]$ for $t = 0, \ldots, T$. **The** most straightforward disjunctive representation is,

$$Vt \text{ -* } (\text{at} \le \text{*lj} \le \text{M} \qquad \text{VtLo } Vt$$

The proposition y_t is true when x_{ij} lies in interval t. An alternative formulation is the following.

$$\begin{aligned} & a_0 \le x_{ij} \le b_T \qquad && y_t \vee y'_t, \quad t = 1, \ldots, T \\ & y_t \rightarrow (x_{ij} \ge a_t) && \\ & y'_t \rightarrow (x_{ij} \le b_{t-1}) && \end{aligned} \tag{3}$$

Here yt is true when the flow volume lies in interval t or higher, and $y[$ is true when it lies in interval t - 1 or lower.

An MILP representation is

$$\text{*ij} \ge \text{<*0} + \sum_{\text{tsl}}^{T} o.{}_t y_t$$

$$x_{ij} \le b_0 + \sum_{\text{tssl}}^{T} b_t y_t$$

$$\sum_{t=1}^{T} y_t \le 1$$

$$y_t \in \{0,1\}, \quad t = 1, \ldots, T$$

2.3 A Warehouse Location Problem

A simple warehouse location problem will be useful to illustrate how cuts can be generated from knapsack constraints. The 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

Xij = flow from warehouse *i* to demand point *j*.
fi = fixed cost of warehouse i.
ki = capacity of warehouse i.
dj = demand at point j.
Cij = unit transport cost from *i to j*.

The fixed cost $Z\{$ that is paid for warehouse *i* is either /; or zero, depending on whether the flow out of the warehouse is positive. This poses the disjunctions,

$$(z_i = f_i) \vee \begin{pmatrix} z_i = 0 \\ \sum_j x_{ij} = 0 \end{pmatrix}, \text{ all } i.$$

The MLLP model can therefore be written,

$$\begin{aligned} \min \quad & \sum_i z_i + \sum_{ij} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} \leq \text{fc}_{*>} \wedge * \qquad \text{yi}^{\text{v}}\ y'n \text{ all } i \\ & \sum_i x_{ij} \geq d_j, \text{ all } j \\ & Xij \geq 0, \text{ all i,j} \\ & Vi - \bullet (z\{ = /,\bullet), \text{ all } i \\ & y_i' \rightarrow (\textstyle\sum_j x_{ij} = 0), \text{ all } i \end{aligned}$$

The traditional MILP model is

$$\begin{aligned} \textbf{min} \quad & \wedge 2^k M + \sum_{ij} c_{ij} x_{ij} \\ \textbf{s.t.} \quad & 5Z\text{«tj} \leq him, \text{ all i} \\ & \text{J]x}_{\text{t'i}} \geq dj, \text{ allj} \\ & \text{x}_{\text{ti}} \geq 0, \text{ all i,j}, \\ & \text{KG } \{0,1\}, \text{ allt}. \end{aligned}$$

2.4 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 [60]. It presents insurmountable difficulties for MILP primarily because there seems to be no economical formulation of the problem within the MILP framework. The problem is much more easily formulated in the broader MLLP context and illustrates the advantages of multivalent discrete variables.

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 / 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 *ha* in each period t, unless it is itself a host, indicated by the truth of proposition $\pounds_i$. In the latter case *ha* = *i* for all *t*. To encourage mingling, no pair of visiting crews are permitted to meet more than once. The proposition *rrtijt* 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 *ha* = *j*. The only propositions that enforce linear inequality constraints are the tf_i's, which force Z_i = 1 when true. The remaining propositions correspond to empty constraint sets.

The problem can be stated as follows. The objective function counts the number of host boats. The predicate alldiff means that all of its arguments have distinct values.

$$
\begin{aligned}
\min \quad & \sum_{i \in I} z_i \\
\text{s.t.} \quad & z_i \geq 0, \quad i \in I && v_{ijt} \leftrightarrow (ha = j), && i,j \in I, t \in T && (a)\\
& \delta_i \rightarrow (z_i \geq 1), \quad i \in I && \delta_i \vee \text{alldiff}(h_{i1}, \ldots, h_{iT}), && i \in I && (b)\\
& && \delta_i \leftrightarrow (ha = i), && i \in I, t \in T && (c)\\
& && \sum_{\substack{i \in I \\ i \neq j}} c_i v_{ijt} \leq K_j - c_j, && j \in I, t \in T && (d)\\
& && \delta_i \vee \delta_j \vee m_{ijt} \vee (h_{it} \neq h_{jt}), && i,j \in I, i < j, t \in T && (e)\\
& && \sum_{t \in T} m_{ijt} \leq 1, && i,j \in I, i < j && (f)\\
& && h_{it} \in \{1, \ldots, |I|\}, && i \in I, t \in T &&
\end{aligned}
\tag{4}
$$

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.

Formula (d) is the boat capacity constraint. It should be interpreted as a logical formula rather than a linear inequality in 0-1 variables. The summation means that c_i is counted in the sum for each true v_{ijt}. The inequality as a whole means that enough v_{ijt}s should be false so that the resulting sum is at most $K_j - C_j$. The interpretation of inequalities as logical propositions is further discussed in Section 6.

Formula (e) says that if crews *i* and *j* are both visiting crews (i.e., $\pounds_i$ and $\pounds_j$ are false), then either m_{ijt} is true or $h_{it} \neq h_{jt}$; i.e., *rrtijt* 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. Here again the inequality should be interpreted as a logical proposition.

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

Formultion of an MILP model is much more difficult. The most challenging constraint is the one that requires visiting crews to meet at most once. The authors of [60] 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 *ha* as numeric variables and enforces the all-different constraints in an awkward manner that is characteristic of MILP. The variables *ha* 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 [60]. Yet it will prove intractable.

The MILP model can be stated as follows.

$$
\begin{aligned}
&\min \quad Yl^{6}* && (^{a})\\
&\text{s.t.} \quad *,\bullet + (1 - t\text{>}y\text{«}) \geq 1, \quad iJeI,t\text{€}T && (6)\\
&\quad (l - *) + t\text{>}\text{'}Mt\text{≥}l, \quad \dot{*} \text{€} I, t \text{€} T && (c)\\
&\quad v_{ijt} + ay, + Pijt \geq 1, \quad i,j\text{€}l,te T && (d)\\
&\quad -hu + 3^{\cdot} \geq 1 - |I|(1 - ay_t), \quad ij \ e \ I,* \ G \ T && (e)\\
&\quad hit\text{-}j \geq 1 - |I|(1 - /3yt), \quad ij \ Zl,teT && (I)\\
&\quad \sum_{\substack{i \in I \\ i \neq j}} avut \leq Kj - CJ, \quad jeI,t\text{£}T && (g)\\
&\quad \sum_{t \in T} v_{ijt} \leq 1, \quad i,j \in I, i \neq j && (h)\\
&\quad Si + Sj + m_{iit} + \text{<}t\text{>}i_{it} + frjt \geq 1, \quad ij \ 6 \ I, \ i < j,t \ G \ T && (i)\\
&\quad -fc\text{<}t + fcj^{*} \geq 1 - |I|(1 \text{"} \text{<}t\text{>}Ht)^{\cdot} \dot{h}i \text{€} I, \dot{t} < \dot{j}, t \text{€} T && (j)\\
&\quad hit - fcjt \geq 1 - |I|(1 - 1\text{>}ijt)^{\cdot} h\dot{3} \ e \ I, \ i < j,t \ 6 \ T && (k)\\
&\quad \sum m_{ijt} \leq 1, \quad i,j \in I, \ i < j && (l)
\end{aligned}
\tag{5}
$$

The objective function (a) again counts the number of host boats. Constraints (b) and (c) require to remain on their own boat if and only if they are a host crew. Constraints (d)-(f) use a disjunctive mechanism to relate $v\text{^}t$ to *ha.* They say that if *ha* = *j* (i.e., *-KXijt* and $-i/3y_y$, which say that $h_{\dot{x}t}$ is neither less than nor greater than j), then v_{ijt} = 1. Constraint (g) is the same capacity constraint as before, interpreted this time as a 0-1 inequality. 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 *ha* = /ijt, then my* = 1. As before, (1) says that a pair of crews must meet at most once.

3 The Disjunctive Interpretation of MLLP

Any instance of the MLLP model (1) can be written in the form of a disjunctive programming problem,

$$
\begin{aligned}
&\min \quad ex && (6)\\
&\text{s.t.} \quad \backslash / \ \bar{A}^{f}z \geq \bar{a}\backslash
\end{aligned}
$$

where *T* is a finite index set. This is done by letting the disjunction in (6) be

$$
\bigvee_{y} \{ A'\dot{x} \geq a^{j} \mid V_j \text{ is true}, \ j = 1, \ldots, n \}. \tag{7}
$$

Here *y* ranges over every value that satisfies the logical constraints; i.e., every value *y* for which there exists an *h* that makes every $0_i(y, h)$ true.

Because every disjunct of (6) represents a polyhedron in the a-space, the feasible set of (6) and therefore any MLLP model is a union of finitely many polyhedra. Jeroslow showed that this union can be represented by an MILP constraint set if all the polyhedra have the same recession cone. The recession cone of a polyhedron *P* is the set of all directions *d* such that for some $\bar{x} \in P$, $\bar{x} + ad \in P$ for all $a \geq 0$. This restriction on recession cones is of little practical import, because if one places laxge upper bounds on all the variables, the recession cone of all the polyhedra is the origin.

Conversely, every MILP model can be written as a disjunctive problem, because it can first be transformed to a 0-1 problem,

$$\begin{aligned} \min \quad & ex \\ \text{s.t.} \quad & Ax + By \geq a \\ & x \in \{0,1\}, \quad \text{all } j, \end{aligned}$$

which is in turn equivalent to a disjunctive model whose constraint set is

$$\bigvee_{y} Ax \geq a - By,$$

where *y* ranges over all 0-1 vectors.

An individual logical formula *gi(y,h)* can also be given a disjunctive interpretation. Its feasible set can be regarded as (7), where *y* ranges over all values that make *gi(y,h)* true for some *h*. The feasible set of $\leq_t(y, h)$ can therefore be regarded as a union of finitely many polyhedra in the x-space.

4 The Basic Algorithm

The basic MLLP algorithm branches on the truth value of propositional variables y_j and on values of the discrete variables *hj*. When branching fixes *yj* to true or false, the formula *yj* or -ij/j becomes one of the logical formulas *gi(y,h)*. When *hj* is fixed to v, the *domain Dj* of *hj* (i.e., the set of its possible values) is reduced to *{v}*.

Next a logic processing algorithm is applied to the formulas. This may generate additional formulas *(logic cuts)*. It may also fix the values of additional j/j's or remove elements from some JDJ'S. If the formulas are unsatisfiable this may or may not be discovered, depending on the strength of the algorithm, but if it is discovered the search backtracks. Section 6 below discusses logic processing algorithms in greater detail.

Linear cuts may now be generated for some of the logical constraints if desired, as discussed in Section 5. A linear programming problem is formulated whose constraints are the inequalities $A^J x \geq a^J$ that correspond to true j/j's, plus any additional cuts. If the LP is infeasible, the algorithm backtracks. Otherwise the solution $\bar{x}$ of the LP will in general satisfy certain constraint sets $A^* x \geq a^J$ and not others. If proposition *yj* is not already fixed to true or false, it is temporarily assumed true if $\bar{x}$ satisfies $A^\wedge x \geq a^J$ and false otherwise. If an unfixed *yj* corresponds to an empty constraint set, it can be given a default value that applies until it is fixed otherwise.

At this point the variables *yj* are true or false and the discrete variables *hj* may have domains of various sizes; a singleton domain fixes the variable. If these values make all formulas true, $\bar{x}$ is a feasible solution. If some *gi{y,h)* is false or has no determinate truth value (because *h*

is not fully determined), one may try to generate a *separating cut;* i.e., a valid inequality for $9x(y, h)$ that $\bar{x}$ violates. The generation of separating cuts is treated in Section 5.7. If no such inequality is generated, then $g_{\bar{x}}(y, h)$ is regarded as an unsatisfied formula. It should be noted that if $\bar{x}$ lies within the convex hull of the feasible set of $gi(y, h)$ but not within the feasible set itself, then no linear inequality can cut it off, and the $flf_t(y, h)$ will inevitably be classified as unsatisfied.

Finally, a variable yj or hj is chosen for branching. This should be a variable that when set to at least one of its values satisfies an unsatisfied formula, or perhaps brings it closer to satisfaction in some sense.

A more precise statement of the algorithm appears in Fig. 2.

5 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 propositional variables. Logical formulas that do not fix any of their variables are not represented in the LP relaxation. The latter is therefore likely to provide a weak bound, and when possible it is important to augment it with additional cuts that represent the logical formulas.

This section presents some techniques for obtaining linear relaxations of logical formulas by generating valid cuts in the continuous variables. As noted earlier, some of these cuts mimic the effect of the traditional continuous relaxation of a 0-1 model. But 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.

5.1 The Convex Hull

It was remarked earlier that any logical formula $g\%\{y^h)$ is equivalent to a disjunction of linear constraint sets,

$$\mathbf{V}_{t\in T}\ A'x^aK \qquad (8)$$

Its feasible set is therefore a union of finitely many 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 $0 \leq x \leq$ m, it is not uncommon for the convex hull of (*) to fill most or all of the box described by $0 \leq x \leq m$. A notorious example of this arises in scheduling problems. If operations 1 and 2 begin at times $x\backslash$ and x_2 and last 2 minutes, one imposes the disjunctive constraint

$$(x_2 \geq x_x + 2)\ \mathrm{V}\ (\mathrm{xi} \geq x_2 + 2)$$

to ensure that one occurs after the other. The upper bounds m represent the latest time at which an operation could be scheduled and are therefore likely to be much larger than 2. The dashed line in Fig. 3 encloses the convex hull when m = (10,10). In this case the best possible relaxation is given by $x\backslash + x_2 \geq 2$, $X\backslash + x_2 \geq 18$ and $0 \leq Xj \leq 10$. This is not much different than $0 \leq Xj \leq 10$ and is probably useless in practice.

NOTICE WARNING CONCERNING COPYRIGHT RESTRICTIONS:
The copyright law of the United States (title 17, U.S. Code) governs the making of photocopies or other reproductions of copyrighted material. Any copying of this document without permission of its author may be prohibited by law.

Mixed Logical/Linear Programming

John N. Hooker and Maria A. Osorio

EDRC 7M0-96

Mixed Logical/Linear Programming *

J. N. HOOKER

Graduate School of Industrial Administration
Carnegie Mellon University, Pittsburgh, PA 15213 USA

M. A. OSORIO

School of Chemical Engineering
University of Puebla, Puebla, Mexico 72550

July 1996

Abstract

Mixed logical/linear programming (MLLP) is an extension of mixed integer/linear programming (MILP). It represents 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 and disadvantages of MLLP 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. Mixed integer/linear programming (MILP), the traditional approach, is effective in many instances. But it unnecessarily restricts the modeling and solution options available. MLLP allows naturally for branching strategies, relaxations and logic processing algorithms that neither fit comfortably into nor are suggested by MILP. In particular it counts the traditional strategies among ii> options and should therefore be seen as an extension of MILP rather than an alternative to it.

Mixed discrete/continuous problems are traditionally conceived as continuous problem* in which some of the variables are restricted to be integers. MLLP takes a completely different view. It does not attempt an often unnatural and contrived embedding of the discrete aspects of the problem within a linear programming model. Instead, it represents the discrete elements by logical formulas and only the continuous element by linear inequalities. It therefore has the option of dispensing with integer variables. Rather than require that a feasible solution satisfy a fixed set of inequalities, MLLP provides several alternative sets of inequalities. The role of the logical formulas is to govern which alternatives are acceptable.

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

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

$$\begin{aligned} &\min \quad ex \\ &\text{s.t.} \quad {}_{Vi} - (A'\dot{x} >\_a'), \quad j \in J \qquad {}_{gi}\{y,h), \quad i \in I. \end{aligned} \qquad (1)$$

The model has a logical part (on the right) and a continuous part (on the left). The logical part consists of formulas $\{y, h)$ that involve *atomic propositions* $y = (y_1, \ldots, y_n)$, which are either true or false. Such a formula might be $y\ \vee\ y<i$, which says that y or $y<i$ (or both) must be true. There may also be some variables $h = (h_1, \ldots, h_m)$ that take several discrete values. The continuous part associates each atomic proposition yj with a system $A^x \geq a^*$ of linear inequalities. The system is enforced when yj is true. So the formula $y\ \vee\ y<i$ in effect requires any solution x to satisfy $A^1x \geq a^1$ or $A^2x \geq a^2$ (or both). In general, $(x,y,/i)$ is feasible if (y, h) makes all the logical formulas true and x satisfies the linear systems corresponding to true j/j's.

The problem (1) can be solved by branching on the truth values of the y/s and the discrete values of the $hj's$. At each node of the search tree, one solves a linear programming problem (LP) containing the constraints that correspond to true y/s, plus any cuts 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 further logical constraints and may fix some additional y/s and $hj's$.

It is easy to show that every MLLP is equivalent to a disjunctive programming problem whose constraint is a disjunction of linear systems (i.e., the solution must satisfy at least one of the systems). Its feasible set is therefore a union of finitely many polyhedra.

1.2 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 framework. The potential advantages of MLLP are articulated, and several are illustrated by computational experiments.

Because MLLP is a general approach to continuous/discrete problem solving, a thoroughgoing 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, cuts, 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., [10, 64]) suggests that MLLP can be usefully extended to mixed logical/nonlinear programming (MLNLP). This possibility is not pursued here.

1.3 Advantages of MLLP

Several potential advantages of MLLP's broader framework, summarized below, should become evident in the ensuing discussion. Some of the advantages are illustrated by the computational experiments, as indicated.

- ***Separation of branching and relaxation.*** **In MILP, integer variables serve as both a branching and relaxation device. The problem is relaxed by removing the integrality constraint on the variables, and one branches on integer variables that have a fractional value in the solution of the relaxation. But there are many other branching and relaxation schemes. MLLP permits one to use any branching scheme in combination with any relaxation, because the relaxation need not involve the discrete variables. The relaxation is generally created by adding cuts to the continuous part of the model.**

- ***Smaller linear programming problems.*** **Some combinatorial problems are formulated only or most conveniently with a large number of discrete variables. MILP forces these variables to be included in the LP problems solved at each node, whereas MLLP puts only continuous variables in the LP problems. This advantage is dramatically illustrated by the party problem. In some cases, however, the advantage of removing integer variables is offset by the necessity of adding cuts to achieve a good relaxation. This is discussed below.**

- ***Faster and/or more effective processing of discrete variables.*** **Logical variables can often be fixed at a node by applying inference procedures to the logical constraints. In many cases the same variables would have been fixed by solving the conventional continuous relaxation. But in these cases logic processing is much faster because it can be achieved by a unit resolution procedure. This is illustrated by the party problem. In other cases, logic processing is more powerful than linear programming, because it fixes variables to true or false when they would receive fractional values in a continuous relaxation.**

- ***Feasible solutions identified sooner.*** **The conventional continuous relaxation of a 0-1 disjunctive formulation can have fractional solution values even when the disjunction is satisfied. This means that MILP can keep branching after a feasible solution is found. MLLP avoids this defect and may therefore produce a smaller search tree. This is illustrated by the flow shop problem and instances of the process synthesis problem that have semicontinuous variables.**

- ***Stronger relaxations.*** **MLLP can provide stronger relaxations in three ways.**

  - **Valid cuts can sometimes be found for a disjunctive constraint that are stronger than the continuous relaxation of any obvious 0-1 formulation.**
  - **Logic processing can generate logical constraints *(logic cuts)* whose linear relaxations can be added to the linear model. The same logic cuts can be used as cutting planes in a conventional branch-and-cut algorithm, but logic processing provides a systematic way of generating them. This is illustrated by the warehouse location and party problems.**
  - **The idea of Benders cuts can be generalized to an MLLP setting to produce logic-based cuts that may be useful when solution of LP relaxations is expensive.**

- *An alternative approach to identifying cuts.* The logical point of view can suggest logic cuts that follow easily from an intuitive understanding of the problem but are not easily revealed by conventional polyhedral analysis. It can also suggest *nonvalid* logic cuts, which do not change the optimal value but are stronger because they exclude feasible solutions. The chemical processing network problems will illustrate both points.

- *More natural modeling.* Integer variables, particularly 0-1 variables, are very often contrived to express what were originally conceived as logical constraints. In such cases MLLP permits a more natural formulation. In other cases a problem is most readily expressed with multivalued discrete variables, as for example problems in which a variable's value may indicate the machine to which a job is assigned or an operation's position in a sequence. Constraints on such variables that are not easily expressed in inequality form may readily be formulated with predicates that are quite natural for logic processing algorithms. A particularly useful predicate is the "all different" predicate, which requires that a set of variables all have different values. The advantages of multivalued variables have been exploited in the constraint satisfaction and constraint programming literature, and the algorithms developed there may be used in the logic processing phase of an MLLP solver.

The last point deserves expansion. A habit of writing models for the convenience of the solver, as is common in operations research, can lead one to forget that the primary role of modeling in science is explanatory. The modeling process should improve one's understanding and should not be befuddled by the necessity of writing a machine-friendly formulation. On the other hand, if the model is totally unsuited for solution, it will require a substantial reformulation either by an automatic procedure or a human expert. It may be difficult to relate the solution to the original formulation. MLLP tries to steer a middle course. It makes discrete/continuous modeling somewhat more natural by eliminating the necessity of integer variables and introducing logical constraints and multivalent variables. But the solution approach is still closely related to the problem statement because it branches on the discrete variables that appear in the original model. The model generally requires additional preparation before solution, but preparation usually involves adding cuts rather than reformulating the problem.

1.4 Mitigating Factors

The advantages of moving to MLLP's broader framework are mitigated by at least three considerations.

- *Relaxations must be explicitly generated.* An MILP model comes with a ready-made linear relaxation, but **MLLP** obliges one to make a conscious choice of relaxation. It is sometimes unobvious how to create a relaxation that is even as strong as the traditional one (without reintroducing the traditional integer variables).

- *The relaxation may be large.* The cuts needed to build a useful relaxation may make the linear constraint set larger than an MILP model.

- *More expertise is needed.* MLLP generally requires greater expertise than MILP because of the greater range of choices it imposes on the user. It may therefore be suited to a smaller circle of practitioners.

One might respond to the first two points as follows. Although a nontraditional relaxation may be large or hard to identify in some cases, there are other cases in which it can solve an otherwise intractable problem. The key is to be able to identify the appropriate relaxation for a given constraint set. The following cases may be distinguished and are more rigorously characterized in the remainer of the paper.

1. ***The constraint(s) to be relaxed can have no good linear relaxation,*** **because the convex** hull of the feasible set occupies most or all of the solution space. In these cases no relaxation should be used. MLLP permits this, whereas MILP forces one to use integer variables, which introduce useless overhead.

2. ***A good relaxation is possible but the MILP relaxation is weak or useless.*** **It may be** possible to replace the MILP relaxation with a reasonable number of cuts that produce a stronger relaxation. Two general methods, described below, that may be helpful are tightening of MILP cuts and generation of "optimal separating cuts."

3. *The MILP relaxation is useful.* In this case it may be easy to mimic the effect of the MILP relaxation with a few cuts. If not, one can always add integer variables to obtain the classical relaxation. Even here, one may wish to relax only a portion of the model in this fashion. The integer variables need not be used for branching purposes, so that the distinction between branching and relaxation is maintained.

The final objection, that MLLP requires more expertise, can be partially overcome by automating as many choices as possible and by installing redundancy. Commercial MILP solvers, for example, automatically apply a number of cuts and preprocessing devices that may or may not be useful for a given problem.

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 tricks 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.

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 [25]. Granot and Hammer [23] 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], 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 [42], 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 [66, 67, 68, 70], Blair [8, 9] and Hooker [28, 28, 29, 30, 31] explored connections betwen logic and optimization. Beaumont [6] 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, 3, 4], he described families of valid cuts that can be used to create relaxations of disjunctive constraints.

More recently, Hooker argued in [32] that a logic-based approach to optimization, including MLLP, can exploit problem structure in ways that are parallel to traditional polyhedral techniques. Wilson [71, 72, 73] studied logic cuts and 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 [37, 49, 50, 51, 52, 64]. These papers developed some of the key MLLP concepts discussed here. Bollapragada, Ghattas and Hooker also obtained encouraging results in structural design [10].

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 [44, 45], 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 [5] 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 [16] and Kowalski [43], allow one to formulate a problem in a subset of first-order logic (Horn clause logic). Recent versions of the logic programming language PROLOG [11, 61], 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 [20, 63, 65]. A number of systems along this line have been developed in addition to Prolog III, including CLP(R) [39], CAL [1], CHIP [19, 59], the ILOG solver [46], and other packages [12, 57, 53]. 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 cuts and relaxations, although the constraint programming literature has shown some interest in exploiting structure (e.g., [21]).

1.7 Outline of the Paper

The remainder of the paper begins with a presentation of four example problems that illustrate MLLP modeling (Section 2). Section 3 presents the disjunctive interpretation of MLLP, and Section 4 summarizes the basic MLLP algorithm. Two long sections (5 and 6) respectively discuss relaxations and logic processing algorithms. Some of these are illustrated in the last section, which presents computational results for the four example problems.

Aside from its survey and development of MLLP generally, the specific contributions of this paper include necessary and sufficient conditions for whether an elementary cut for a disjunction is supporting (Section 5.4), necessary and sufficient conditions for integrality of a 0-1 disjunctive representation (Section 5.5), a definition of optimal separating cuts (Section 5.7), a completeness proof for multivalent resolution (Section 6.6), a unit resolution algorithm for multivalent clauses (Section 6.6), and an application of logic-based Benders decomposition to MLLP (Section 6.8).

2 Some Examples

Examples from four application areas are presented to illustrate some of the concepts, advantages and disadvantages of MLLP. The first two problem classes concern flow shop scheduling with zero-wait transfer and processing network design. They are important in chemical engineering. The third, a warehouse location problem, was chosen partly because it is ill-suited to logic modeling. The fourth problem, the progressive party problem, is chosen to represent problems in which the discrete element dominates. An attempt was made to choose problems with the flavor or complexity of real applications, although the warehouse location problem is somewhat stylized.

2.1 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 $J\{$ be the set of machines on which job i is processed, and $d_{\dot{X}\dot{J}}$ the processing time for job i on machine j. If U is the start time for job t, the job is completed at time

$$u + \sum_{j \in J_i} \wedge$$

It is necessary to make sure that two jobs t, k are not scheduled to be in process at the same time on the same machine j € J,·n J*. The finish time of job i on machine j is $t_x \cdot + D_{\dot{X}\dot{J}}$, where

$$D_{ij} = \sum_{\substack{? \in Ji \\ i' < i}} d_{ij'},$$

and its start time is $t_{\dot{x}} + D_{t\dot{J}} - d_{\dot{x}}j$. 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. The natural

and obvious way to formulate this "or" constraint is with a logical disjunction. For each pair (i, fc), one writes the disjunction,

$$(U + Da_{-} \le t_k + D_{ik} - d_{ik},\ j \in JiD\ J_k)\ \mathbf{V}\ (t_k + D_{kj} \le t_l + Dij - d_{ih}\ j \in Ji\ \mathbf{n}\ J_k).$$

The inequalities in 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. The model to minimize makespan is therefore

$$\begin{aligned} \min\ & T \\ \text{s.t.}\ & T \ge t_i + \sum_{j \in J_i} d_{ij}, \quad \text{all jobs } i, \\ & (t_k - U > r_{ik}) \ \mathbf{V}\ (U - t_k > r_{ki}), \quad \text{all jobs } t, k \text{ with } t \pm \text{fc}, \\ & t \ge 0, \quad \text{allf}, \end{aligned}$$

where

$$r_{ik} = \max_{\in J_i \cap J_k} \{Dij - D_{ik} + d_{ik}\}.$$

This MLLP model is easily put into the form (1) by introducing logical variables. Let *yik* be true when job i is sdieduled before job fc on each machine on which they are both processed. The formal MLLP is therefore

$$\begin{aligned} \mathbf{min}\ & \mathbf{T} \\ \mathbf{s.t.}\ & U \ge 0,\ T \ge U + \sum_{j \in J_i} dij, \ \text{all}\ \% \quad (a) \qquad y_{ik} \ \mathbf{V}\ y_{ki}, \ \text{all}\ i, k,\ i \ne k. \\ & y_{ik} \rightarrow (t_k - t_i \ge r_{ik}) \qquad (b) \end{aligned} \tag{2}$$

There is no need to represent constraints (a) with propositions t/t because they hold categorically.

Formulating (2) as a traditional MILP model requires the additional step of introducing big-M constraints to represent the disjunctions.

$$\begin{aligned} \min\ & T \\ \mathbf{s.t.}\ & U \ge 0,\ T \ge t_l + \sum_{j \in J_i} d_{ji} \ \text{ail}\ i \\ & t_k - t_i \ge r_{ik} - M(1 - y_{ik}) \\ & U - h \ge r_{ki} - My_{ik} \\ & y_{<ik} e \{0,1\}, \quad \text{allt, fc}. \end{aligned}$$

2.2 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. 1. Bach 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. 1 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

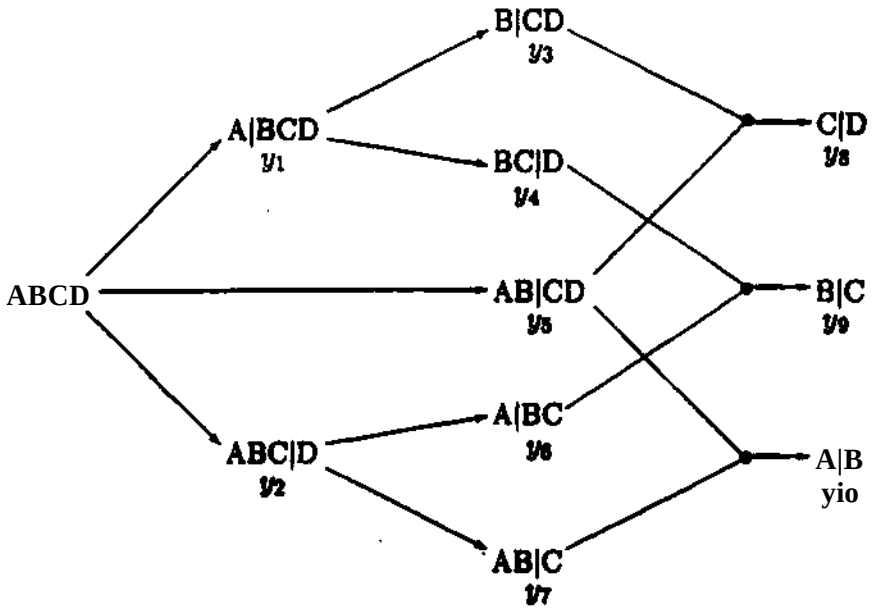

Figure 1: *A ^component separation network.*

often complicated by recycling of streams and waste heat, the latter typically resulting in a nonlinear model that is not discussed here. The volume of some streams into and out of the network can be semicontinuous variables, and this possibility is considered.

Let *E* be the set of directed arcs in the network. The network in general contains a set / of unit nodes, which represent processing units, and a set J of structural nodes, at which no unit is present and flow is simply conserved. The flow on arc (i,*j)* is x_{ij}- and incurs a unit cost of *Cij.* Inputs to the network and outputs from it are represented by structural nodes, and if j is an output node, cy would typically be negative to indicate revenue from sale of the product. The fixed cost of unit *i* is /». The flow on arc (i,*j)* leaving unit *j* is *a%j* times the total input to the unit.

The discrete element of the problem is the decision as to whether to install a unit *i* and its incident arcs. Either fixed cost f_x is incurred, or else there is no fixed cost and no flow leaves the unit. If *Z{* is the fixed cost paid for unit i,

$$(\text{« = k0}\mathbf{V}\mathbf{f}_{\mathbf{r}} \quad \text{*-,°.-0V}$$

If *yi* is true when unit i is installed, the MLLP model can be written,

$$\begin{aligned}
\min \quad & \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_i z_i \\
\text{S.t.} \quad & \sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk}, \quad j \in J \qquad\qquad y_i \vee y(, \quad \text{t 6 /} \\
& x_{ij} = \alpha_{ij} \sum_{(k,i)\in E} x_{ki}, \quad (i,j) \in E, \; i \in I \\
& 0 \le x_{ij} \le k_i, \quad (i,j) \in E \\
& y_i \rightarrow (z_i = f_i), \quad i \in I \\
& y_i' \rightarrow \begin{pmatrix} z_i = 0 \\ \sum_{(i,j)\in E} x_{ij} = 0 \end{pmatrix}, \quad i \in I.
\end{aligned}$$

The MILP model represents the disjunctions in the usual way.

$$\min \sum_{(i,j)\in E} c_{ij}x_{ij} + \sum_i f_i y_i$$

$$\text{s.t.} \sum_{(i,j)\in E} x_{ij} = \sum_{(j,k)\in E} x_{jk}, \quad j \in J$$

$$x_{ij} = a_{ij} \qquad I$$

(W)€£

$$- \sum_{(i,j)\in E} x_{ij} \le k_i y_i, \quad i \in I$$

$$y_i \in \{0,1\}, \quad i \in I$$

Process synthesis problems can involve semicontinuous variables, as described in [52]. These are variables whose values must lie within certain intervals. For example, an input to the processing network may consist of a feedstock obtained from a small set of suppliers, each of which can provide either nothing or a quantity within a certain range. Taking all possible sums of these intervals yields the intervals within which the total input volume must fall.

A semicontinuous flow variable x_{ij} must lie in one of the intervals $[a_t, b_t]$ for $t = 0, \ldots, T$. **The** most straightforward disjunctive representation is,

$$Vt \text{ -* } (\text{at} \le \text{*lj} \le \text{M} \qquad \text{VtLo } Vt$$

The proposition y_t is true when x_{ij} lies in interval t. An alternative formulation is the following.

$$\begin{aligned} & a_0 \le x_{ij} \le b_T \qquad && y_t \vee y'_t, \quad t = 1, \ldots, T \\ & y_t \rightarrow (x_{ij} \ge a_t) && \\ & y'_t \rightarrow (x_{ij} \le b_{t-1}) && \end{aligned} \tag{3}$$

Here yt is true when the flow volume lies in interval t or higher, and $y[$ is true when it lies in interval t - 1 or lower.

An MILP representation is

$$\text{*ij} \ge \text{<*0} + \sum_{\text{tsl}}^{T} o.{}_t y_t$$

$$x_{ij} \le b_0 + \sum_{\text{tssl}}^{T} b_t y_t$$

$$\sum_{t=1}^{T} y_t \le 1$$

$$y_t \in \{0,1\}, \quad t = 1, \ldots, T$$

2.3 A Warehouse Location Problem

A simple warehouse location problem will be useful to illustrate how cuts can be generated from knapsack constraints. The 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

Xij = flow from warehouse *i* to demand point *j*.
fi = fixed cost of warehouse i.
ki = capacity of warehouse i.
dj = demand at point j.
Cij = unit transport cost from *i to j*.

The fixed cost $Z\{$ that is paid for warehouse *i* is either /; or zero, depending on whether the flow out of the warehouse is positive. This poses the disjunctions,

$$(z_i = f_i) \vee \begin{pmatrix} z_i = 0 \\ \sum_j x_{ij} = 0 \end{pmatrix}, \text{ all } i.$$

The MLLP model can therefore be written,

$$\begin{aligned} \min \quad & \sum_i z_i + \sum_{ij} c_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_j x_{ij} \leq \text{fc}_{*>} \wedge * \qquad \text{yi}^{\text{v}}\ y'n \text{ all } i \\ & \sum_i x_{ij} \geq d_j, \text{ all } j \\ & Xij \geq 0, \text{ all i,j} \\ & Vi - \bullet (z\{ = /,\bullet), \text{ all } i \\ & y_i' \rightarrow (\textstyle\sum_j x_{ij} = 0), \text{ all } i \end{aligned}$$

The traditional MILP model is

$$\begin{aligned} \textbf{min} \quad & \wedge 2^k M + \sum_{ij} c_{ij} x_{ij} \\ \textbf{s.t.} \quad & 5Z\text{«tj} \leq him, \text{ all i} \\ & \text{J]x}_{\text{t'i}} \geq dj, \text{ allj} \\ & \text{x}_{\text{ti}} \geq 0, \text{ all i,j}, \\ & \text{KG } \{0,1\}, \text{ allt}. \end{aligned}$$

2.4 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 [60]. It presents insurmountable difficulties for MILP primarily because there seems to be no economical formulation of the problem within the MILP framework. The problem is much more easily formulated in the broader MLLP context and illustrates the advantages of multivalent discrete variables.

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 / 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 *ha* in each period t, unless it is itself a host, indicated by the truth of proposition $\pounds_i$. In the latter case *ha* = *i* for all *t*. To encourage mingling, no pair of visiting crews are permitted to meet more than once. The proposition *rrtijt* 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 *ha* = *j*. The only propositions that enforce linear inequality constraints are the tf_i's, which force Z_i = 1 when true. The remaining propositions correspond to empty constraint sets.

The problem can be stated as follows. The objective function counts the number of host boats. The predicate alldiff means that all of its arguments have distinct values.

$$
\begin{aligned}
\min \quad & \sum_{i \in I} z_i \\
\text{s.t.} \quad & z_i \geq 0, \quad i \in I && v_{ijt} \leftrightarrow (ha = j), && i,j \in I, t \in T && (a)\\
& \delta_i \rightarrow (z_i \geq 1), \quad i \in I && \delta_i \vee \text{alldiff}(h_{i1}, \ldots, h_{iT}), && i \in I && (b)\\
& && \delta_i \leftrightarrow (ha = i), && i \in I, t \in T && (c)\\
& && \sum_{\substack{i \in I \\ i \neq j}} c_i v_{ijt} \leq K_j - c_j, && j \in I, t \in T && (d)\\
& && \delta_i \vee \delta_j \vee m_{ijt} \vee (h_{it} \neq h_{jt}), && i,j \in I, i < j, t \in T && (e)\\
& && \sum_{t \in T} m_{ijt} \leq 1, && i,j \in I, i < j && (f)\\
& && h_{it} \in \{1, \ldots, |I|\}, && i \in I, t \in T &&
\end{aligned}
\tag{4}
$$

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.

Formula (d) is the boat capacity constraint. It should be interpreted as a logical formula rather than a linear inequality in 0-1 variables. The summation means that c_i is counted in the sum for each true v_{ijt}. The inequality as a whole means that enough v_{ijt}s should be false so that the resulting sum is at most $K_j - C_j$. The interpretation of inequalities as logical propositions is further discussed in Section 6.

Formula (e) says that if crews *i* and *j* are both visiting crews (i.e., $\pounds_i$ and $\pounds_j$ are false), then either m_{ijt} is true or $h_{it} \neq h_{jt}$; i.e., *rrtijt* 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. Here again the inequality should be interpreted as a logical proposition.

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

Formultion of an MILP model is much more difficult. The most challenging constraint is the one that requires visiting crews to meet at most once. The authors of [60] 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 *ha* as numeric variables and enforces the all-different constraints in an awkward manner that is characteristic of MILP. The variables *ha* 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 [60]. Yet it will prove intractable.

The MILP model can be stated as follows.

$$
\begin{aligned}
&\min \quad Yl^{6}* && (^{a})\\
&\text{s.t.} \quad *,\bullet + (1 - t\text{>}y\text{«}) \geq 1, \quad iJeI,t\text{€}T && (6)\\
&\quad (l - *) + t\text{>}\text{'}Mt\text{≥}l, \quad \dot{*} \text{€} I, t \text{€} T && (c)\\
&\quad v_{ijt} + ay, + Pijt \geq 1, \quad i,j\text{€}l,te T && (d)\\
&\quad -hu + 3^{\cdot} \geq 1 - |I|(1 - ay_t), \quad ij \ e \ I,* \ G \ T && (e)\\
&\quad hit\text{-}j \geq 1 - |I|(1 - /3yt), \quad ij \ Zl,teT && (I)\\
&\quad \sum_{\substack{i \in I \\ i \neq j}} avut \leq Kj - CJ, \quad jeI,t\text{£}T && (g)\\
&\quad \sum_{t \in T} v_{ijt} \leq 1, \quad i,j \in I, i \neq j && (h)\\
&\quad Si + Sj + m_{iit} + \text{<}t\text{>}i_{it} + frjt \geq 1, \quad ij \ 6 \ I, \ i < j,t \ G \ T && (i)\\
&\quad -fc\text{<}t + fcj^{*} \geq 1 - |I|(1 \text{"} \text{<}t\text{>}Ht)^{\cdot} \dot{h}i \text{€} I, \dot{t} < \dot{j}, t \text{€} T && (j)\\
&\quad hit - fcjt \geq 1 - |I|(1 - 1\text{>}ijt)^{\cdot} h\dot{3} \ e \ I, \ i < j,t \ 6 \ T && (k)\\
&\quad \sum m_{ijt} \leq 1, \quad i,j \in I, \ i < j && (l)
\end{aligned}
\tag{5}
$$

The objective function (a) again counts the number of host boats. Constraints (b) and (c) require to remain on their own boat if and only if they are a host crew. Constraints (d)-(f) use a disjunctive mechanism to relate $v\text{^}t$ to *ha.* They say that if *ha* = *j* (i.e., *-KXijt* and $-i/3y_y$, which say that $h_{\dot{x}t}$ is neither less than nor greater than j), then v_{ijt} = 1. Constraint (g) is the same capacity constraint as before, interpreted this time as a 0-1 inequality. 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 *ha* = /ijt, then my* = 1. As before, (1) says that a pair of crews must meet at most once.

3 The Disjunctive Interpretation of MLLP

Any instance of the MLLP model (1) can be written in the form of a disjunctive programming problem,

$$
\begin{aligned}
&\min \quad ex && (6)\\
&\text{s.t.} \quad \backslash / \ \bar{A}^{f}z \geq \bar{a}\backslash
\end{aligned}
$$

where *T* is a finite index set. This is done by letting the disjunction in (6) be

$$
\bigvee_{y} \{ A'\dot{x} \geq a^{j} \mid V_j \text{ is true}, \ j = 1, \ldots, n \}. \tag{7}
$$

Here *y* ranges over every value that satisfies the logical constraints; i.e., every value *y* for which there exists an *h* that makes every $0_i(y, h)$ true.

Because every disjunct of (6) represents a polyhedron in the a-space, the feasible set of (6) and therefore any MLLP model is a union of finitely many polyhedra. Jeroslow showed that this union can be represented by an MILP constraint set if all the polyhedra have the same recession cone. The recession cone of a polyhedron *P* is the set of all directions *d* such that for some $\bar{x} \in P$, $\bar{x} + ad \in P$ for all $a \geq 0$. This restriction on recession cones is of little practical import, because if one places laxge upper bounds on all the variables, the recession cone of all the polyhedra is the origin.

Conversely, every MILP model can be written as a disjunctive problem, because it can first be transformed to a 0-1 problem,

$$\begin{aligned} \min \quad & ex \\ \text{s.t.} \quad & Ax + By \geq a \\ & x \in \{0,1\}, \quad \text{all } j, \end{aligned}$$

which is in turn equivalent to a disjunctive model whose constraint set is

$$\bigvee_{y} Ax \geq a - By,$$

where *y* ranges over all 0-1 vectors.

An individual logical formula *gi(y,h)* can also be given a disjunctive interpretation. Its feasible set can be regarded as (7), where *y* ranges over all values that make *gi(y,h)* true for some *h*. The feasible set of $\leq_t(y, h)$ can therefore be regarded as a union of finitely many polyhedra in the x-space.

4 The Basic Algorithm

The basic MLLP algorithm branches on the truth value of propositional variables y_j and on values of the discrete variables *hj*. When branching fixes *yj* to true or false, the formula *yj* or -ij/j becomes one of the logical formulas *gi(y,h)*. When *hj* is fixed to v, the *domain Dj* of *hj* (i.e., the set of its possible values) is reduced to *{v}*.

Next a logic processing algorithm is applied to the formulas. This may generate additional formulas *(logic cuts)*. It may also fix the values of additional j/j's or remove elements from some JDJ'S. If the formulas are unsatisfiable this may or may not be discovered, depending on the strength of the algorithm, but if it is discovered the search backtracks. Section 6 below discusses logic processing algorithms in greater detail.

Linear cuts may now be generated for some of the logical constraints if desired, as discussed in Section 5. A linear programming problem is formulated whose constraints are the inequalities $A^J x \geq a^J$ that correspond to true j/j's, plus any additional cuts. If the LP is infeasible, the algorithm backtracks. Otherwise the solution $\bar{x}$ of the LP will in general satisfy certain constraint sets $A^* x \geq a^J$ and not others. If proposition *yj* is not already fixed to true or false, it is temporarily assumed true if $\bar{x}$ satisfies $A^\wedge x \geq a^J$ and false otherwise. If an unfixed *yj* corresponds to an empty constraint set, it can be given a default value that applies until it is fixed otherwise.

At this point the variables *yj* are true or false and the discrete variables *hj* may have domains of various sizes; a singleton domain fixes the variable. If these values make all formulas true, $\bar{x}$ is a feasible solution. If some *gi{y,h)* is false or has no determinate truth value (because *h*

is not fully determined), one may try to generate a *separating cut;* i.e., a valid inequality for $9x(y, h)$ that $\bar{x}$ violates. The generation of separating cuts is treated in Section 5.7. If no such inequality is generated, then $g_{\bar{x}}(y, h)$ is regarded as an unsatisfied formula. It should be noted that if $\bar{x}$ lies within the convex hull of the feasible set of $gi(y, h)$ but not within the feasible set itself, then no linear inequality can cut it off, and the $flf_t(y, h)$ will inevitably be classified as unsatisfied.

Finally, a variable yj or hj is chosen for branching. This should be a variable that when set to at least one of its values satisfies an unsatisfied formula, or perhaps brings it closer to satisfaction in some sense.

A more precise statement of the algorithm appears in Fig. 2.

5 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 propositional variables. Logical formulas that do not fix any of their variables are not represented in the LP relaxation. The latter is therefore likely to provide a weak bound, and when possible it is important to augment it with additional cuts that represent the logical formulas.

This section presents some techniques for obtaining linear relaxations of logical formulas by generating valid cuts in the continuous variables. As noted earlier, some of these cuts mimic the effect of the traditional continuous relaxation of a 0-1 model. But 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.

5.1 The Convex Hull

It was remarked earlier that any logical formula $g\%\{y^h)$ is equivalent to a disjunction of linear constraint sets,

$$\mathbf{V}_{t\in T}\ A'x^aK \qquad (8)$$

Its feasible set is therefore a union of finitely many 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 $0 \leq x \leq$ m, it is not uncommon for the convex hull of (*) to fill most or all of the box described by $0 \leq x \leq m$. A notorious example of this arises in scheduling problems. If operations 1 and 2 begin at times $x\backslash$ and x_2 and last 2 minutes, one imposes the disjunctive constraint

$$(x_2 \geq x_x + 2)\ \mathrm{V}\ (\mathrm{xi} \geq x_2 + 2)$$

to ensure that one occurs after the other. The upper bounds m represent the latest time at which an operation could be scheduled and are therefore likely to be much larger than 2. The dashed line in Fig. 3 encloses the convex hull when m = (10,10). In this case the best possible relaxation is given by $x\backslash + x_2 \geq 2$, $X\backslash + x_2 \geq 18$ and $0 \leq Xj \leq 10$. This is not much different than $0 \leq Xj \leq 10$ and is probably useless in practice.

```
Let G be a set of logical formulas, initially the formulas
   gi(y,h) in (1).
Let I 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 y9 initially y¯= (£/,...,t/).
Let D = (Z?i,...,Dm) be the domains of hu...,hm.
Let z̄ be an upper bound on the optimal value, initially oo.
Let A be the set of active nodes, initially with A= {(G,L,y¯,£>)}.
While A is nonempty:
   Remove a tuple (G, L,ȳ, D) from A.
   Apply a logic processing algorithm to G, possibly changing
      the contents of G, possibly changing some ȳj's from U to T
      or F, and possibly removing elements from some £>j's.
   If no logical contradiction is detected then
      For each ȳj changed to T, add A^x ≥ aJ to L.
      Generate inequality cuts as desired for formulas in G and
         add them to L.
      Let x̄ minimize ex subject to L.
      If ex̄ < z̄ then
         For each yj\
            If ȳj e {T,F} then let ŷj = ȳj.
            Else let ŷj = T if A*x̂¯ ≥_ a* and ŷj = F otherwise.
         Let C, initially empty, be the set of unsatisfied formulas.
         For each gi(y,h) 6 G:
            If ft(ŷ>&) is F or {/ then
               If desired, try to generate a separating cut for
                  9i(y,h) with respect to (ŷ,h̄).
               If a separating cut is generated then add it to I.
               Else add gi(y,h) to C
         If C is empty then
            If no separating cuts were generated then
               x̄ is feasible; let x* = x̄ and z̄ = ex̄.
            Else add (G,X,ȳ,D) to A.
         Else
            Choose a variable yj with ŷ/j = (/ or a variable hj with
               |JDj| > 1, such that setting yj to T or F, or setting hj
               to one of its discrete values, satisfies or tends
               to satisfy one of the formulas in C.
            If yj is chosen then
               Add (GU {%},!,5,0) and (Gu {^yj},L,ȳ,D) to A.
            Else if /ij is chosen then
               For each v G Dji
                  Set D̄-D, set Dj = {v}, and add (G,L,ȳ,D̄) to A,
If z̄ < oo then z* is an optimal solution.
Else the problem is infeasible.
```

Figure 2: *A generic MLLP branching algorithm.*

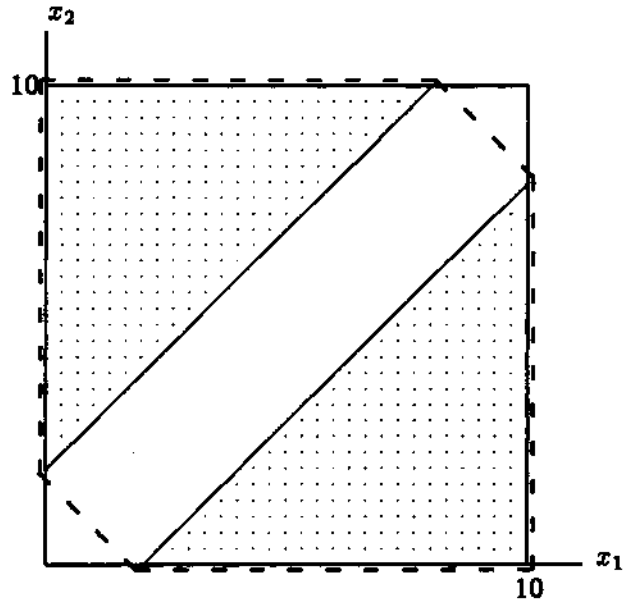

Figure 3: *Convex hull of the feasible set of a scheduling disjunction.*

An even more striking example is that of semicontinuous variables. If $0 \leq XJ \leq 4$ and the disjunction

$$(0 \leq Xj \leq 1) \vee (3 \leq Xj \leq 4)$$

is imposed, the convex hull is the entire interval [0,4]. Any conceivable relaxation is therefore useless.

5.2 Disjunctive and Dual Cuts

A relaxation of (8) can be obtained by generating valid cuts that partially or completely describe the convex hull. Balas [4] characterized valid cuts for (8) as follows. First, note that $bx \geq 0$ is a valid cut for a feasible disjunct $A^l x \geq a^1$ if and only if it is dominated by a nonnegative linear combination (or *surrogate)* of $A^l x \geq a^*$. A dominating surrogate can be written $uAx \geq$ tut, where $6 \geq$ uA, $/3 \leq ua$ and $u \geq 0$. But $bx \geq /?$ is a valid cut 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 $bx \geq ft$.

Theorem 1 (Balas) *The inequality $bx \geq 0$ is a valid cut for (8) if any only iffor each feasible system $A^*x \geq a^1$ there is a $u^* \geq 0$ such that $b \geq rfA^*$ and $/? \leq$ ti*a*.*

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

$$\max_{\backslash Uz T^1 \quad V \quad - \; t \in T^x} \{tx^t A^t\}^{>}) \; x > minima^1\}_J \qquad (9)$$

for (8), where the maximum is componentwise. Theorem 1 clearly implies that if $x \geq 0$, every valid cut is dominated by a disjunctive cut (9).

The strength and usefulness of a disjunctive cut (9) depends radically on the choice of surrogates. One could in principle generate disjunctive cuts to define every facet of the convex hull, but this is often impractical. The task of obtaining a good relaxation for (8) is in essence the task of choosing multipliers u^l judiciously.

One initially attractive choice for u^l is given by the solution of a dual problem. Each surrogate should ideally give the best possible bound on the objective function ex. That is, u^l should be chosen so that the minimum value of ex subject to $u^tA^tx \geq vta^1$ is maximized. The desired u^* is easily seen to be the optimal solution of the LP dual of min{cx | $A^*x \geq$ a*}, where u^* is the vector of dual variables. (To put it differently, the surrogate dual for linear programming is identical to the LP dual [22].)

The difficulty with this approach is that because $A^lx \geq$ a* is only a small part of the original constraint set, it may have no coupling with the objective function. That is, the variables XJ that have nonzero coefficients in ex may have zero coefficients in $A^tx \geq$ a*, and vice-versa. This means that ex provides no information to guide the choice *of u* 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 ex and $A^fx \geq$ a*. This can be done as follows. At any node of the search tree a system $Ax \geq a$ of certain linear constraints are enforced by true propositional variables. If $Ax \geq a$ is included in each term of the disjunction (8), it becomes

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

For each t one solves the dual of

$$\begin{array}{lll} \min & ex & \\ \text{s.t.} & A^*a; \geq a^* & (u^*) \\ & Ax \geq a & (u) \end{array} \qquad (10)$$

where (u^u) are the dual variables as shown. An optimal solution of the dual supplies a reasonable set of multipliers u^l for the disjunctive cut (9).

Unfortunately this approach appears to be impractical, because (10) is generally a large LP, and it is time consuming to solve the dual of (10) for each disjunct. In fact, if one branched on the disjunction by enforcing each disjunct in turn, (10) is precisely the LP one would solve at each child node. So one might as well branch on the disjunction rather than relax it. There could be some advantage in relaxing several disjunctions simultaneously, but results reported in Section 7.2 indicate that the time investment is impracticably large. The remaining discussion will therefore focus on much faster mechanisms for choosing effective multipliers u^*.

5.3 Elementary Cuts

The most common sort of disjunctive constraint (8) is one in which each disjunct is a .single inequality.

$$\bigvee_{t \in T} a^t x \geq \alpha^t. \qquad (11)$$

Beaumont [6] showed how to generate a cut for (11) that is equivalent to the continuous relaxation of the traditional 0-1 formulation of (11). The latter is

$$\begin{array}{l} a^lx \geq a^1 - M_t\{\backslash - y_t), \quad * \in T \\ \sum_{t \in T} y_t = 1 \\ 0 \leq x \leq m \\ j,_te\{o,i\}, \quad ter. \end{array} \qquad (12)$$

Each *Mt* is chosen so that a* — *Mt* is a lower bound on the value of $a^l x$. The bounds $0 \leq x \leq m$ are imposed to ensure that such a lower bound exists. It can be assumed without loss of generality that *Mt* > 0, because otherwise the inequality is vacuous and can be dropped. Beaumont obtains a cut by taking a linear combination of the inequalities in (12), where each inequality *t* receives weight I/Aft. This yields what Beaumont calls the *elementary cut* for (11),

$$\left(\sum_{t\in T} \frac{a^t}{M_t}\right) x \geq \sum_{t\in T} \frac{\alpha_t}{M_t} - |T| + 1. \tag{13}$$

Theorem 2 (Beaumont) *The elementary cut (13) is equivalent to the continous relaxation of (12). That is, the feasible set of (13) and* $0 \leq x \leq m$ *is equal to the projection of the feasible set of the continuous relaxation of (12) onto the x-space.*

One can also prove equivalence by applying Fourier elimination to (12) in order to eliminate *y*. It is easy to show that (13) and $0 \leq x \leq m$ are the resulting inequalities.

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

$$\begin{aligned} &dy \geq 6 \\ &y_t \rightarrow (a^*x \geq a_t), \quad t \in T \\ &0 \leq x \leq \text{TO}, \end{aligned} \tag{14}$$

where $d \geq 0$. It is true that (14) can be put in disjunctive form using the schema (7), but this may require a large number of disjuncts. (Disjunctions are of course a special case of $dy \geq S$ in which $6 = 1$ and each $dj \in \{0,1\}$.) The 0-1 representation of (14) is

$$\begin{aligned} &a^*x \geq a^* - M_t(l - y_t), \quad te T \\ &0 \leq x \leq m \\ &dy \geq 6 \\ &y_t e\{o,i\}, \quad te T. \end{aligned} \tag{15}$$

A linear combination of the inequalities, using weights d_t/Mtj yields the elementary cut,

$$\left(\sum_{t\in T} a^t \frac{d_t}{M_t}\right) x \geq \sum_{t\in T} \alpha_t \frac{d_t}{M_t} - \sum_{t\in T} d_t + \delta. \tag{16}$$

This is in general weaker than the continuous relaxation of (15), however. If $\wedge 2_t d_t = 6$, for example, (15) forces all the disjuncts to hold, where (16) only forces a linear combination of them to hold.

Beaumont obtains *Mt* solely from the bounds $0 \leq x \leq m$ by setting

$$a_t - M_t = \overline{53}\dot{m}\dot{n}\{O,a\$.\}mj. \tag{17}$$

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

$$at — Mt = \min_{t' \neq t} \dot{j}\min_{x} \{a^*x \mid a^{1\prime}x \geq a^*\#,\ 0 \leq x \leq m\}\dot{\backslash}_j \tag{18}$$

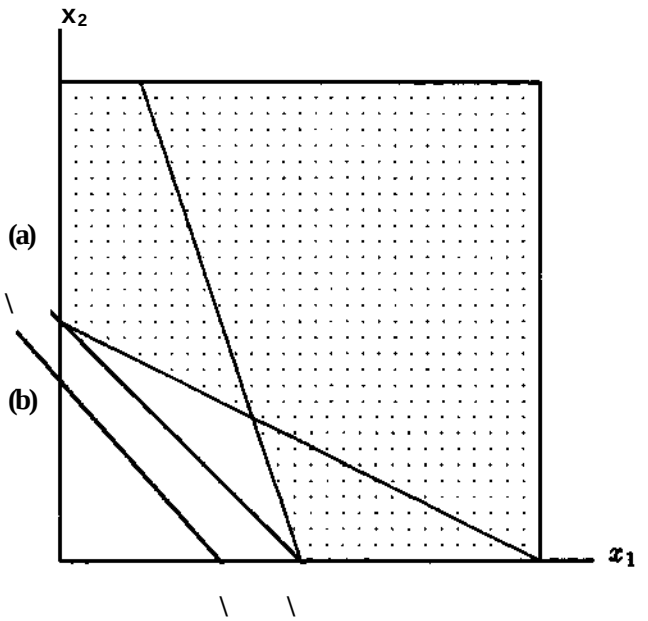

Figure 4: *A supporting elementary cut (a) and a nonsupporting elementary cut (b).*

The computation involved is negligible.

Consider for example the following constraint set, whose feasible set is the shaded area in Fig. 4.

$$(xi + 2x_2 \geq 2) \text{ V } (3xi + x_2 \geq 3)$$
$$0 \leq Xj \leq 2.$$

The 0-1 formulation is

$$x_1 + 2x_2 \geq 2 - M_1(1 - y_1)$$
$$3x_1 + x_2 \geq 3 - M_2(1 - y_2)$$
$$Jfi + V2 = 1$$
$$0 \leq x_j \leq 2, \quad y_j \in \{0, 1\}$$

Beaumont puts (Afi, M_2) = (2,3) which results in the cut $|xi + |x_2 \geq 1$. By contrast, (18) puts $(Mi, M_2) = (1,2)$, which yields the stronger cut $xi + x_2 \geq 1$. This is a *supporting* cut in the sense that it defines a supporting hyperplane for the feasible set.

Even when (18) is used to compute M*, the resulting cut may fail to be supporting. Consider the constraints (Fig. 5),

$$(-xx + 2x_2 \geq 2) \text{ V } (2xi - x_2 \geq 2)$$
$$0 \leq Xj \leq 2.$$

(17) sets (Mi, M_2) = (4,4), which results in the useless cut $xi + x_2 \geq 0$. The cut can obviously be strengthened to $xi + x_2 \geq 1$.

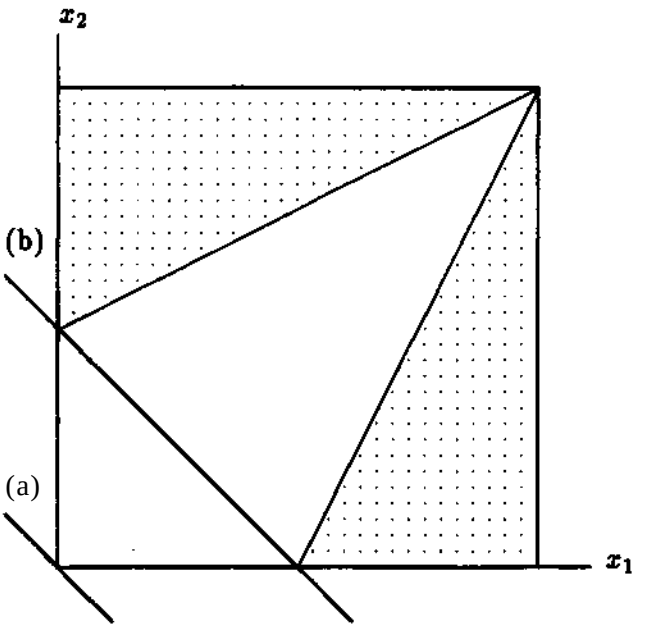

Figure 5: *An elementary cut (a) and a strengthened elementary cut (b).*

When the inequalities $a^*x \geq at$ in (14) are replaced by systems of inequalities $A^l x \geq a'$, many elementary cuts are required to achieve the effect of the traditional relaxation. Let each system $A^*x \geq a^*$ consist of inequalities $A^{ti}x \geq a\backslash$ for $t \in It$- The 0-1 formulation is

$$\begin{aligned} & A^*x \geq a^* - M^*(1 - y_t), \quad teT \\ & \mathbf{0 \leq x \leq m} \\ & \mathbf{dy \geq S} \\ & \mathbf{y_t \ e \ \{0,1\}, \ teT.} \end{aligned} \tag{19}$$

Here M^* is an array such that for each i 6 /*, $a\backslash - M\backslash$ is a lower bound on $A^u x$. Repeated applications of Fourier elimination reveal that the projection of the feasible set of (19) onto the x-space is described by the set of inequalities of the form,

$$\left(\sum_{t\in T} A^{ti_t}\frac{d_t}{M^t_{i_t}}\right)x \geq \sum_{t\in T} a^t_{i_t}\frac{d_t}{M^t_{i_t}} - \sum_{t\in T} d_t + \delta,$$

for all possible vectors $(tj, \ldots, i\backslash x\backslash)$ G /i X ... XI ^|.

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

5.4 Supporting Elementary Cuts

The example of Fig. 5 shows that an elementary cut 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 cut. 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 cut is supporting, and when it is not, to improve upon the traditional continuous relaxation the cut represents.

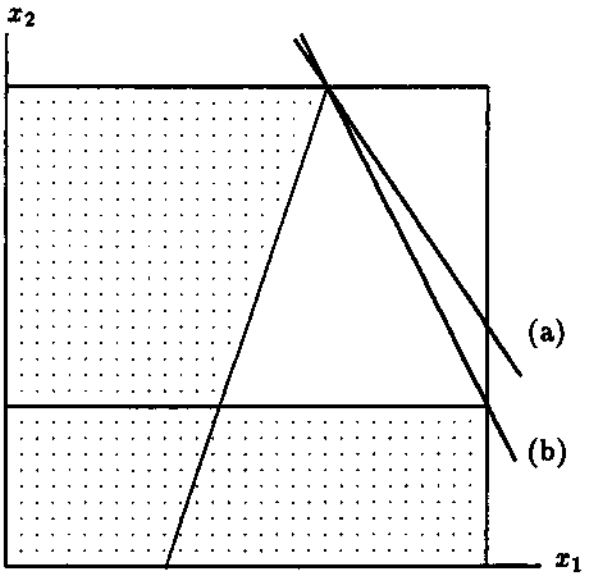

Figure 6: *A supporting elementary cut (a) and a facet-defining cut (b).*

Figures 4 and 5 may suggest that two disjuncts $a^1x \geq c^*i$, $a^2x \geq c^*2$ produce a supporting elementary cut 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. Figure 6 shows the feasible set for

$$(-3x_1 + x_2 \geq -3) \vee (-x_2 \geq -1)$$
$$0 \leq XJ \leq 3$$

The elementary cut is $3zi + 2x_2 \leq 12$, which is supporting even though (-3,1) and (0,-1) subtend an obtuse angle.

A more adequate analysis goes as follows. Let $bx \geq (3$ be the strengthened elementary cut, where bx is the left-hand side of the elementary cut (16). Because $bx \geq f3$ defines a supporting hyperplane for the feasible set of (11), /? is the smallest of the minimum values obtained by minimixing bx subject to each of the disjuncts $a^lx \geq a_t$. That is,

$$\beta = \min_{t \in T} \beta_t, \tag{20}$$

where

$$f3t = \min \{bx \mid a^lx \geq a_u \ 0 \leq x \leq 1m\}.$$

The computation of $/3_t$ is simplified if $6 \geq 0$, because in this case the upper bounds $x \leq m$ can be ignored. To this end one can introduce the change of variable,

$$\tilde{x}_j = \begin{cases} x_j & \text{if } 6i \geq 0 \\ m_j - Xj & \text{otherwise} \end{cases}$$

The strengthened elementary cut in terms of $\hat{x}$, namely $\hat{b}\hat{x} \geq /\hat{?}$, can now be computed, where $\hat{b}j = |6j|$. The right-hand side of $bx \geq /3$ can then be recovered from (20) by setting

$$\beta_t = \hat{\beta}_t + \sum_{6_j < 0} Mjbj. \tag{21}$$

It remains to compute

$$\hat{fit} = \min \{\hat{b}x \mid \hat{a}^f\hat{x} \ge d, \hat{x} \ge 0\}, \qquad (22)$$

where

$$a_3 = \cdot \quad \_\ 0 \quad \backslash\ -a) \quad \textbf{otherwise} \qquad (23)$$

and

$$\hat{\alpha}_t = \alpha_t - \sum_{\substack{3 \\ b_j < 0}} m_j a^t_j. \qquad (24)$$

Because $\hat{6} \ge 0$, LP duality applied to (22) yields that

$$\hat{fit} = \min \backslash\text{-}Jr\backslash\max\{d_t, 0\}. \qquad (25)$$

This proves,

Theorem 3 *The elementary cut (16) for the disjunction (11) is supporting if and only if its right-hand side is equal to fi, as defined by (20), (21) and (25).*

5.5 Integral 0-1 Representations

The traditional continuous relaxation of a disjunctive constraint actually has two liabilities. It can be weak, as already noted, but even when it is strong, it may permit fractional solutions 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 (8) is the following.

$$\begin{aligned} &A^*x \ge a' - M_t(1-y_t), \quad t \in T \\ &0 \le x \le m \\ &\sum_{t \in T} y_t = 1 \\ &y_t \in \{0,1\}, \quad t \in T, \end{aligned} \qquad (26)$$

where M_t is given by (18). The claim is that when x is fixed to some value $\bar{x}$, an extreme point solution $y = \bar{y}$ of (26) can be nonintegral even when $\bar{x}$ satisfies (8). An example of this is presented by a simple semicontinuous variable, $x \in \{0\} \cup [d_1, \wedge]^*$ or

$$\begin{aligned} &(-z \ge 0) \vee (x \ge s_x) \\ &0 \le x \le s_2. \end{aligned}$$

The continuous relaxation of (26) is

$$\begin{aligned} &-x \ge -s_2(1-y) \\ &x \ge s_1 - s_1 y \\ &0 \le x \le s_2 \\ &0 \le y \le 1. \end{aligned} \qquad (27)$$

If x is fixed to $\bar{x}$ and (27) is projected onto y, the result is

$$1 - \frac{\bar{x}}{s_1} \leq y \leq 1 - \frac{\bar{x}}{s_2}, \quad 0 \leq y \leq 1. \tag{28}$$

If $0 \leq \bar{x} \leq 52$, $\bar{y} = 1 - \bar{f}$ is an extreme point solution of (28) and therefore (27), and it is nonintegral whenever $0 < \bar{x} < S2$- So (27) can have extreme point solutions with fractional y even when $\bar{x}$ G [51,62]? and even though (27) is the best possible (convex hull) relaxation of (26). The extreme point solutions for $\bar{x}$ G [^1,62] are guaranteed to have integral y only when $\$_x = 52$; 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 (26) when x is fixed to $\bar{x}$. Let the continuous relaxation of (26) be *integral* if for every $(\bar{x},\bar{y})$ satisfying (26) such that $\bar{y}$ is an extreme point of P_s, V is integral.

The following characterizes integral relaxations. A disjunct of (8) 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 (8) contains no redundant disjuncts. For t,t' G T with t ^t' define*

$$y;(t^f) = \max\{t \mid M_t y_t \leq A^t x - a^1 + M_u\ A^{1\prime} x \geq a'\ 0 \leq x \leq m,\ y_t \leq 1\}.$$

Then the continuous relaxation of (26) is integral if and only if $y_t^*(t') = 0$ *for every pair* t,t^f G T *with* $t \wedge t^9$.

Proof. It is clear that $yf(t')$ can be written,

$$yW) = \max[y_t \mid A^*x \geq a^1 - M_t(1 - y\%\ A^v x > a^{*\prime}, 0 < x < m,\ y_t < 1\}. \tag{29}$$

It is convenient to let $S_t = \{x \mid A^t x \geq a^*,\ 0 \leq x \leq m\}$ for t G T.

Claim. For any $\bar{x}$ G $S_t i$ and any t ^t',

$$y;(0 = \max_y\{2/t|yGPx\}. \tag{30}$$

Proof of claim. It suffices to show that any y_t that is feasible in (30) is feasible in (29), and vice-versa. First suppose y_t is feasible in (30). Then by letting $x = \bar{x}$ it is seen to be feasible in (29), because $A^{*\prime}\bar{x} \geq a^{*\prime}$ by virtue of the fact that $\bar{x}$ G S_t/. Conversely, let y_t be feasible in (29). To see that it is feasible in (30), set $y_t i = 1 - y\%$ and $y_t» = 0$ for $t'' \wedge t,t^f$. It is enough to show

$$\wedge'' \bar{x} \wedge a \wedge' - M t \wedge 1 - y t'') \tag{31}$$

for all t'' G T. But (31) holds for $t'' = t$ by stipulation. It holds for $t'' = t'$ because $\bar{x}$ £ S_b, and it holds for $t'' \wedge t,t'$ by definition of M_t«. This proves the claim.

Now suppose that $yJT(t') > 0$ for some t,t' with $t \wedge t^f$. Because disjunct t^1 is not redundant, one can choose $\bar{x}$ G $S_b \backslash S_t$. This implies that $j£(f) < 1$, which together with $yf(t') > 0$ means that $yt(t')$ is nonintegral. Also (30) implies that some $\bar{y}$ with $\bar{y}_t = \#(*')$ is an extreme point of $P_{\bar{x}}$. It follows that (26) is not integral.

For the converse, suppose that $y_t^*(<') = 0$ for all pairs t,t^* with $t \# t'$. It suffices to show that for any $\bar{x}$ satisfying (8), any given extreme point $\bar{y}$ of $P\$$ is integral. If it is supposed that $\bar{x}$ G $5_t/$, the following can be stated.

$$\begin{aligned} &\max\{y_t^* \mid y \,\mathrm{G}\, P^*\} = 1 \\ &\max\{y, \mid j/\mathrm{G}\, P^*\} = 0, \quad t^*t^*. \end{aligned} \tag{32}$$

The first is due simply to the fact that $\bar{x}$ G S_t^*. By the above claim, the second is equivalent to $y_t^*(t') = 0$, which is given. But (32) implies that $P\$$ is a line seqment of unit length extending from the origin in a positive direction along the $y_t t$ axis. Thus any extreme point $\bar{y}$ G P^* is integral, which means that (26) is integral. •

Corollary 1 *Consider a disjunction (11) with one inequality per disjunction and bounds* $0 \leq x \leq m$. *If (11) contains no redundant disjuncts, then (26) is integral if and only if*

$$\max\{\mathbf{a}'\mathbf{z} \mid a^{1\prime}\underline{x} > a_v,\ \underline{0} < \underline{x} < \mathbf{m}\} = \mathbf{a}_t - M_t \tag{33}$$

for every t,t' £T *with* $t \pm t^*$.

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

Corollary 2 *The continuous relaxation of (26) is integral only if the feasible sets described by the disjuncts of (8) are disjoint.*

Proof. Suppose two of the feasible sets intersect, e.g. those corresponding to disjuncts / and t'. Then $Hf(f) = 1$, which violates the condition of the theorem. •

Not even disjoint feasible sets are sufficient for integrality, as the above example shows.

5.6 Beaumont's Cuts

Beaumont [6] identified a class of facet-defining cuts for disjunctive constraints in which each disjunct consists of a single inequality, as in (11). They are facet-defining in the sense that, under certain conditions, they define facets of the convex hull of the feasible set of (11). Unfortunately, the conditions are often unsatisfied, which limits the usefulness of the cuts.

Beaumont's approach is essentially a reasonable method for choosing multipliers u^* so a> to generate a disjunctive cut (9). He first incorporates the bounds $x \leq m$ into the disjunction (11) to obtain

$$\bigvee_{t \in T} \qquad , \quad t \in T.$$

The vector of nonnegative multipliers for each disjunct is u^l = (v*,u;t), where Wt corresponds to the last inequality in the disjunct. The object is to derive a disjunctive cut $bx \geq j$ that satifies

$$\begin{aligned} b &\geq wta^t — v^l \\ 0 &\leq wtat — v^t m \end{aligned}$$

for all t. For a given w_t (yet undetermined), it is reasonable to make the components of 6 as small as possible to get a tight constraint. So let

$$b = \underset{t}{\text{minima}}^*\}, \tag{34}$$

where the minimum is taken componentwise. One can now set

$$v^{\%} = u_t a^t - 6, \quad t \in T,$$

because (34) implies $v^l \geq 0$. To make the right-hand side of the cut as tight as possible, set

$$/? = \min_{t \in T} \{u_t a_t - v^t m\} . \tag{35}$$

It remains to pick values for the w_t's. Beaumont's choice is equivalent to setting $w_t = M_t$ when Aft is derived from the variable bounds as in (17) and $a^* \leq 0$. Thus

$$w_t = \frac{}{a_t - a^t m}. \tag{36}$$

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

$$w_t = \frac{1}{a_t - \min\{a^*,0\}m'} \tag{37}$$

Theorem 5 (Beaumont) *The cut $bx \geq f3$ given by (34)-(36) is facet-defining for (11) if $a_t - a^{\%} m > 0$ for all t G T.*

Beaumont's cut can therefore be superior to a supporting elementary cut. This is illustrated in Fig. 6, where Beaumont's cut is the facet-defining cut $2x\backslash + x<i \leq 7$.

Assuming $at - a^t m > 0$ 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. x = m is often feasible, however, as in the example of Fig. 4. Here $(w\backslash_y W2)$ = (—^, —|), and one must revert to (37), which yields the useless cut $3xi + 2\hat{}2 \geq -2$.

The underlying difficulty is that Beaumont's approach has no mechanism for detecting which corner of the box $0 \leq x \leq m$ should be cut off from the feasible set. An appropriate corner could in effect be identified by using a change of variable similar to one discussed earlier, namely

$$\bar{x}_j = \begin{cases} m_j - x_j & \text{when convenient} \\ x_j & \text{otherwise.} \end{cases}$$

A "convenient" transformation would be one that makes $\hat{a}_t - \hat{a}^{\%} m > 0$ for as many disjuncts t as possible, where $\hat{a}^f$ and $\hat{a}_t$ are given by (23) and (24). This poses an integer programming problem that could be solved heuristically. However, because of the computation involved this option will not be pursued further.

5*7 Optimal Separating Cuts

One way to identify an appropriate point to be cut off by a disjunctive cut is simply to cut off the solution of the current relaxation. This is also a mechanism for using information about the objective function, because the current solution was obtained by minimizing the objective function. Fortunately it is straightforward to state a small LP problem whose solution identifies an separating cut if and only if one exists. Thus if no cut is found, the current solution is known to lie within the convex hull of the feasible set, and branching is necessary to obtain a feasible solution—unless of course the current solution is already feasible. The cut is optimal in the sense that it is chosen to maximize the amount by which the current solution violates it.

Another attraction of optimal separating cuts is that they can be generated for the case in which there are several inequalities in each disjunct. A logical constraint *gi(y,h)* other than a disjunction must, however, be first put in disjunctive form.

Suppose that the solution $\bar{x}$ of the current LP relaxation is to be separated from the feasible set of the disjunctive constraint (8). Any upper bounds $x \leq m$ should be incorporated into each disjunct of (8). Because any disjunctive cut is defined by a choice of multipliers u^1, an LP model can be formulated so as to find a set of u"s that define a cut $bx \geq 0$ that is maximally violated by $\bar{x}$. Such a model is,

$$\begin{aligned} \max \quad & f3\text{-}bx^{-} \\ \text{s.t.} \quad & /? \leq u^{l}a\backslash \quad t \in T \\ & b \geq u^{l}A\backslash \quad t\ 6\ T \\ & -\mathbf{e} \leq 6 \leq \mathbf{e} \\ & \mathbf{u}^{*} \geq 0, \quad t\ 6\ T \\ & /?, 6 \text{ unrestricted.} \end{aligned} \qquad , \qquad (38)$$

Note that the variables in the model are *f3,b,u.* The objective function measures the amount by which $\bar{x}$ violates the cut. If the objective function value is zero, there is no separating cut. The constraint $-e \leq b \leq e$, where *e* is a vector of ones, ensures that the solution is bounded. It results in no loss of generality because an optimal cut can always be rescaled to satisfy the constraint.

The model (38) has an interesting dual.

$$\begin{aligned} \min \quad & (s + i)e \\ \text{s.t.} \quad & {}^{-}\!\!\ast \quad \hat{}\, x^{t} = \quad s\text{-}t \qquad (b) \\ & \qquad t\in T \\ & \text{A'ar}^{*} \geq a^{t}y_{t}, \quad t\ 6\ T \qquad (\text{«}^{*}) \\ & \sum_{t\in T} y_{t} = 1 \qquad (\beta) \\ & s, t, x^{t}, y_{t} \geq 0, \quad t \in T \end{aligned} \qquad (39)$$

If $s - tis$ fixed to zero and $\bar{x}$ is a variable, the constraint set is Balas' convex hull representation for the disjunction (8) [4]. That is, when $s - J = 0$, the projection of the feasible set of (39) onto the $\bar{x}$-space is the convex hull of the feasible set of (8). (This is related to the fact, observed by Williams [69], that the dual of the dual of a disjunctive programming problem is the convex hull representation of the problem.) The problem (39) therefore seeks a point $YlteT^{x\ast\ m}\ \wedge$ convex hull that is closest to $\bar{x}$, as measured by the rectilinear distance.

An optimal separating cut can be superior to a supporting elementary cut. Consider the example of Fig. 6, which becomes

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

The solution of (38) is $0 = -|$, $6 = (-1,-1)$, $u^{l} = (3,0,!)$, $u^{2} = (\wedge,1,0)$, which produces the facet-defining cut $2xi + X2 \leq 7$.

The optimal separating cut need not be facet-defining, however. If the convex hull of the disjunction is the box defined by $0 \leq x_3; \leq 1$ for $j = 1,2$, the optimal separating cut for $\bar{x} = (2,2)$ is $xi + x_2 \leq 2$.

6 Logic Processing

The logical aspect of MLLP revolves primarily around two issues: What repertory of logical formulas are available to express discrete constraints? What logic processing algorithms can be used to derive implications from a set of logical formulas? After a brief discussion of these two questions, subsections will be devoted to several classes of logical formulas of increasing generality. Basic logic processing algorithms will be presented for each.

Logic cuts can also be developed manually, based on insight into the problem structure. This is discussed in Section 6.7. Finally, it is shown how logic cuts can be obtained by generalizing Benders decomposition to an MLLP context.

6.1 Logical Formulas

In theory a logical formula *g%(y, h)* can represent any function of *(y, h)* that takes the values true and false. But there are certain syntactic forms that have proved useful for expressing constraints. A few basic ones will be discussed here.

Logical clauses. A logical clause is a disjunction of *literals,* which are atomic propostions y_3 or their negations -IJ/J. Thus the expression *y*\ V -132 V *y$* is a clause, where V is an inclusive "or." In theory conjunctions of logical clauses can express any function of *y* alone (i.e., any *boolean* function), but it may be convenient to use other forms as well. Implications (e.g. *yi* —• jfe) and equivalences *(y*\ = 1/2) [a]*e readily defined in terms of clauses.

Extended clauses. These have the form, "at least fc of $L_i, \ldots, I_p$ are true," where the L/s are literals. They, too, express boolean functions but are often more convenient than clauses because they have an a quasi-arithmetical as well as a logical aspect.

Knapsack constraints. The familiar 0-1 knapsack constraint *by* >_ *13,* where each *yj* G {0,1}, can also be regarded as a logical formula that is true when the sum over *bj* for which *yj* is true is at least /?. Boolean functions of this form are called *threshold* functions and are studied in the electrical engineering literature [58]. They are difficult to process logically, but they can be used to generate logic cuts in the form of clauses and extended clauses, which are easily manipulated.

Multivalent clauses. These generalize clauses to accommodate multivalent variables and are adequate to express any bivalent function of *(y, h).* They are disjunctions of terms having the form *hj* € *H,* where *H* is a subset of the domain of *hj.*

The all-different predicate. As in the case of bivalent clauses, it is useful to supplement multivalent clauses with other types of syntax. One particularly useful formula states that a set of variables *hj* all have different values, a condition that is awkward to capture with MILP inequalities.

6.2 Logic Processing Algorithms

The goal of logic processing is to extract information that is implicit in the logical formulas 5«(j/^)« It is essentially an inference process that derives *logic cuts,* or implications, from the formulas. It can be useful in two ways:

- The logic cuts may give rise to elementary or other inequality cuts that can strengthen the LP relaxation.
- They may reduce backtracking by ruling out partial assignments that cannot be extended to feasible solutions; these are known as *redundant* assignments in the constraint satisfaction literature [63]. For instance, if there is no feasible solution $(3/1,\ldots, y_n)$ in which $(yu\ J/2) = (T, F)$, a logic cut -ij/i V y_2 would prevent one from exploring a subtree defined by (yi,]b) = *(TjF)* in order to discover this.

As noted earlier, a large body of logic processing algorithms have been developed by the research communities associated with logical inference, constraint satisfaction, constraint programming and logic programming. The discussion here is limited to two basic types of algorithms that can form the logical basis of an MLLP solver. One algorithm is fast but *incomplete* (i.e., does not derive all possible inferences), and one is a much slower, complete algorithm.

The incomplete algorithm is a simple constraint propagation technique and takes the form of *unit resolution* ("forward chaining") in the case of logical clauses. It is probably adequate for most applications, but when a more powerful inference algorithm is required, the *resolution* algorithm can be used. It is a well-known complete inference method for logical clauses and can be generalized to extended and multivalent clauses. Resolution can be quite slow and is unsuitable for problems with a large number of propositional variables. There are many applications, however, in which the number of discrete variables is small relative to the continuous part of the problem. In such cases it may be worthwhile to extract as much information as possible from the logical formulas in order to avoid solving large LP problems.

6.3 Logical Clauses

In principle a logical formula that contains only bivalent variables *yj* can always be written as a conjunction of clauses, i.e., in *conjunctive normal form* (CNF). For example, the formula (yi A 3/2) V jfc, where A means "and," can be written (3/1 V 3/3) A $(y_2$ V 3/3). Also the implication *Vi* -* *V2* (or 1/1 D 1/2) can be written -iyi V y_{2y} and the equivalence 3/1 = y_2 is rendered *hVi* Vy_2)A(yi V-.y_2).

CNF is completely expressive because any formula in bivalent variables can be regarded as a boolean (true-false) function $f(y) = /(yi,\ldots, y_m)$. If $y = v^1,\ldots, v^N$ are the values of y that make *f(y)* false, then the following CNF formula is equivalent to /(y),

$$\overset{N}{\mathbf{A}}\ \overset{m}{\mathbf{V}}\ \mathbf{«(*}\mathbf{)}.$$

where yj(t>$\overset{*}{j}$) is -iyj if $v\overset{*}{j}$ = true and is *yj* if vj = false.

This conversion to CNF requires exponential time and space in the worst case. However, a formula involving the connectives -1, V, A, —•, = (and other connectives with linear-time transformations to CNF) can be converted to CNF in linear time by adding new variables. The algorithm goes as follows. If a given formula *F* has the form *AAB*, where *A* and *B* are subformulas, then apply the algorithm to *A* and *B* separately, and conjoin the results. If *F* has the form *A* —• *B* then rewrite it as -u4v*B* and apply the algorithm to the resulting formula, and similarly for A = *B*. If jPhas the form AVfl, then write it as $(y_m+\backslash$ = A)A(y_{m+2} = 2?)A(y_{m+1} Vj/$_{m+2}$), where y_m+i, y_m+2 are variables that do not occur in F, and apply the algorithm to the resulting formula (see [71] for refinements). The algorithm stops when CNF is achieved.

In practice an MLLP solver would accept several connectives that are readily converted to CNF and make the conversion. It should normally be unnecessary to introduce new variables, because formulas are usually short enough so that a CNF equivalent without them is also short.

A simple *resolution* algorithm [47, 48, 54] derives all implications of a set of clauses. Let clause $C\backslash$ *absorb* clause C_2 when all the literals of $C\backslash$ occur in $C_2\backslash C\backslash$ implies $C<i$ if and only of $C\backslash$ absorbs $C<i$* Two clauses have a (unique) *resolvent* when exactly one variable yj occurs positively in one and negatively in the other. The resolvent is a disjunction of all literals that occur in either clause except yj and -iyj. For instance, 32 V -^3 is the resolvent of 31 V y_2 and -ijh V -1J/3. Given a set 5 of clauses, the resolution algorithm picks a pair of clauses in 5 that have a resolvent that is absorbed by no clause in 5, and adds the resolvent to 5. It repeats until there is no such pair, which occurs after finitely many iterations.

Theorem 6 (Quine [47, 48]) *A clause set S implies clause C if and only if the resolution algorithm applied to S generates a clause that absorbs C. In particular,* 5 *is unsatisfiable if and only if resolution generates the empty clause.*

This theorem will follow from a more general result to be proved in Section 6.6.

Linear relaxations can be generated for clauses derived by resolution, if desired. A clause is simply a disjunction. So if the clause contains all positive literals, and each of its variables corresponds to a linear constraint set, an elementary or other type of cut can be generated as discussed in Section 5. If a derived clause is a *unit clause,* i.e., contains a single literal yj or -ij/j, then yj can be fixed accordingly.

Resolution not only has exponential complexity in the worse case [24] but can be very slow in practice [27]. A much faster inference algorithm that sacrifices the completeness of resolution is *unit resolution.* It is the same as full resolution except that one of the parents of a resolvent is always a unit clause. Its incompleteness can be seen in the example,

$$
\begin{array}{l}
y_x \vee \quad y_2 y \quad 3/3 \\
\text{yi} \vee \neg y_2 \vee \quad y_3 \\
y\backslash \vee \quad y_2 \vee \neg y_3 \\
y_1 \vee \neg y_2 \vee \neg y_3
\end{array}
$$

Resolution fixes $y\backslash$ to true, but unit resolution does nothing because there are no unit clauses to start with. Unit resolution is efficient, however, as it runs in $O(nL)$ time, if there are n variables and L literals, and it tends to be very fast in practice. A precise algorithm that is adapted to the more general case of extended clauses 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. [13, 14, 15, 56, 62]. 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 5 are written as a system $Ay \geq a$ of 0-1 inequalities in the usual fashion; i.e., a clause $\backslash Jj_e jLj$ is written $Y^jeJVjiLj) \geq 1$, where $yj(Lj)$ is yj if $Lj = yj$ and is $1 - yj$ if $Lj = \sim^yj$.

Theorem 7 (Blair, Jeroslow, Lowe [9]) *Unit resolution finds a contradiction in the clause set S if and only if the linear relaxation of the corresponding system $Ay \geq a$ of 0-1 inequalities is infeasible.*

$Ay \geq a$ 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 $0 \geq 1$ from a nonnegative linear combination of $Ay \geq a$. Conversely, if unit resolution detects no contradiction, then the inequalities that represent the remaining clauses can be satisfied by setting each $y_3 = 1/2$.

Although LP duplicates the effect of unit resolution, the latter is preferable for logic processing because of its speed.

6.4 Extended Clauses

Extended clauses seem a particularly useful compromise between arithmetic and logic because they express the notions of "at least" and "at most" but can be efficiently processed as logical formulas. In fact, Barth's constraint-based solver for 0-1 optimization problems [5] reasons with 0-1 inequalities only after converting them to extended clauses.

An extended clause of *degree k* can be written

$$\sum_{j \in J} L_j \geq k,$$

where each L_3 is a literal. Here the sum is not an arithmetical sum but simply counts the number of literals that are true. Ordinary clauses have degree 1. To say that at most k are true, one can write

$$\sum_{j \in J} L_j \leq |J| - k,$$

and one can use two extended clauses to say exactly k are true.

A complete inference algorithm ("generalized resolution") for extended clauses was presented in [27, 30] and is refined by Barth in [5]. It uses resolution as well as a *diagonal summation,* where the latter is defined as follows. An extended clause $Y_{ljeJ} L_j \geq k + I$ *is* the diagonal sum of the set of extended inequalities $\{\wedge j \wedge L_j \geq k \mid i \in J\}$ if $J = |J| \in J \wedge$ but $\wedge^{or}$ each $i \; e \; J, \; i \; \& \; J\%$. The algorithm of [27] is applied to a set 5 of extended clauses as follows. If there are two clauses $C\backslash, C_2$ of degree 1 with a resolvent C that is implied by no extended clause in 5, such that $C\backslash$ is implied by an extended clause in 5 and similarly for C_2, then add C to 5. If there is a set E of extended clauses with a diagonal sum D that is implied by no extended clause in 5, such that each clause in E is implied by some clause in 5, then add D to 5. The algorithm continues until no more clauses can be added to 5.

Theorem 8 **([27, 30])** *A set S of extended clauses implies clause C if and only if the generalized resolution algorithm applied to S generates a clause that implies C.*

Implementation of the algorithm requires recognition of when one extended clause implies another. $Y_{ljeJi} L_j \geq \&i$ implies $Y_{Lj \in J_2} \wedge V \geq \wedge \wedge^{an(* \; on}ty \wedge$

$$l \wedge il - |\{j \; eJinJ_2| \; L_{xj} = L_{2j}\}| \leq An - k_2.$$

When all of the literals of a derived extended clause are positive and correspond to sets of inequalities, a linear relaxation can be formulated using one of the methods described in Section 5. A unit resolution algorithm for extended clauses appears in Fig. 7.

```
Let 5 be a set {^ej, Ljj >_k{ 11 G /} of extended clauses,
   where each L{j is yj or -ryj.
Let P be a stack of unit clauses, initially empty.
For each i G / with |Jt|· = fc,-:
   For each j G J% add Ii;J to U.
   Let Jt = 0.
While U is nonempty:
   Remove L{ from U.
   For each f G / with t G J{\:
      If I^t = L't then
         Let fct = · fc,-l, ·Jt = Ji\{t}.
      Else
         If fct = |Jt| then stop; 5 is unsatisfiable.
         Else
            If ki = \Ji\ + 1 then
               For each j G J; \ {<} add Iij to J7.
               Let Ji = 0.
            Else
               Let Ji = Ji\{t}.
```

Figure 7: ***A unit resolution algorithm for extended clauses.***

Linear programming is a stronger inference algorithm for extended clauses than unit resolution. For example, LP detects the infeasibility of the following inequalities, but unit resolution can do nothing with the corresponding extended clauses.

$$V\backslash + 2/2 + 3/3 \geq 2$$
$$(1 - i/i) + (i - y_2) + (i - y3) \geq 2$$

No known inference algorithm has exactly the same effect as LP on extended clauses, unless one views LP algorithms as inference algorithms. Generalized resolution is of course stronger than LP.

6.5 Knapsack Constraints

A complete inference algorithm for knapsack constraints appears in [30], and an analog of unit resolution can easily be devised for them. But they are perhaps best used as a source of logic cuts that are more easily processed, such as clauses and extended clauses. The implied clauses, for example, are identical to the well-known "covering inequalities" for the constraint, and their derivation is straightforward (e.g., [23]).

It may be more effective, however, to infer extended inequalities. Although it is hard to derive all the extended inequalities that are implied by a constraint, it is easy to derive all *contiguous cuts.* Consider a 0-1 inequality $dy \geq 6$ for which it is assumed, without loss of generality, that $d\backslash \geq d_2 \geq \ldots \geq d_n > 0$; if $dj < 0$, reverse its sign and add dj to 6. A

```
Let k = 1. * = £?=!<*; > Mast =0.
For j = 1,...,n:
   Let; s = s — dj.
   If 5 < S then
      While 5 + d_k < S:
         Let 5 = s + dk , k = k + 1.
      If k > fc^ then
         Generate the cut y\ + ...+yj>k.
         Let fciast = fc.
```

Figure 8: *An algorithm for generating all 1-cuts for a knapsack constraint dy ≥ 5 in which* di ≥ d_2 ≥ ... ≥ d_n > 0.

contiguous cut for $dy \geq S$ is one of the form,

$$\sum_{j=t}^{t+w+k-1} y_j \geq k, \tag{40}$$

where k is the degree of the cut and $w < n$ the "weakness" (tu = 0 indicates a cut that fixes all of its variables). In particular (40) is a *t-cut* because the first term is y_t. (40) is valid if and only if

$$\sum_{j=1}^{t+k-1} d_j + \sum_{j=t+w+k}^{n} d_j < \delta.$$

Furthermore,

Theorem 9 ([35]) *Every t-cut of weakness wfordy>5is implied by a 1-cut of weakness w.*

The power of all t-cuts can therefore be obtained by generating only 1-cuts. The algorithm of Fig. 8, presented in [35], does this in linear time. By way of example, the knapsack constraint

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

gives rise to the 1-cuts,

$$V\backslash + V2 \geq 1$$
$$y_1 + y_2 + y_3 \geq 2$$
$$y_1 + V2 + yz + y^* + ys \geq 3.$$

The first cut could be deleted if desired, because it is redundant of the second.

6.6 Multivalent Clauses

Multivalent clauses provide a convenient and versatile syntax for expressing logical formulas that involve multivalent variables. A multivalent clause has the form

$$\bigvee_{j=1}^{m} (h_j \in H_j), \tag{41}$$

where each *Hj* is a subset of the domain *Dj* of *hj*. For notational simplicity, it is assumed that bivalent variables *yj* are regarded as 2-valued variables *hj* that take a value of, say, 1 when *yj* is true and 0 when *yj* is false. If *Hj* is empty, the term *(hj* G *Hj)* can be omitted from (41), but it is convenient to suppose that (41) contains a term for each *j*.

Any true-false function $f(h) = f(h\backslash,\ldots, h_m)$ can be expressed as a conjunction of multivalent clauses. In particular, -<*(hj* G *Hj)* can be written *(hj* G *Dj* \ *Hj),* and *(hj* G *Hj)* -+ *(hk* G *Hk)* can be written *(hj e Dj* \ *Hj) V (hk* € *Hk).* One multivalent clause *Jj(hj* G *H\j)* implies another *Jj(hj* G Jifyj) if and only if *H\j* C #2j for each *j*. As examples of a multivalent clause consider the following formula (4a) from the progressive party problem.

$$v_{ijt} = (ha = j)$$

It is formally expressed as two multivalent clauses,

$$(v_{ijt} \text{ e } \{\text{o}\text{»}\text{v}(\text{fc}\text{«} \text{ e } \{j\})$$
$$(v_{ijt} \text{ e } \{\text{i}\text{»}\text{v}(\text{fca e } D_j \setminus \{j\}).$$

Resolution can be generalized to obtain a complete inference method for multivalent clauses. The resulting algorithm is related to Cooper's algorithm for obtaining fc-consistency for a set of constraints [18]. Given a set of multivalent clauses,

$$\bigvee_{j=1}^{n} (h_j \; eHa), \quad is \; I, \tag{42}$$

the *resolvent on hk* of these clauses is

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

Ordinary bivalent resolution is a special case. To apply the resolution algorithm to a set 5 of multivalent clauses, find a subset of 5 whose resolvent *M* is implied by no clause in 5, and add *M* to 5. Continue until no further clauses can be added to 5.

The multivalent resolution algorithm is a complete inference algorithm for multivalent clauses. The proof of the theorem uses the idea of Quine's original proof for ordinary resolution.

Theorem 10 *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 implications of 5 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 (41) be £j *Hj*\. Suppose the theorem is false, and let (41) be a longest clause implied by 5 but by no clause in S'.

Claim. At least one *Hj* in (41) is missing at least two elements; i.e., *Dj* *Hj*\ ≥ 2 for some *j*. First it is clear that no *Hj* = *Dj,* because otherwise (41) would be implied by a (in fact, every) clause in S'. Suppose contrary to the claim that every *Hj* is missing exactly one element, say *VJ.* Then *h* = *v* = (vi,.. .,t>$_m$) violates (41) and must therefore violate some clause *\/j(hj* G *H'j)* in 5', because 5' implies (41). This means each *H'j* C *Dj* \ {VJ}, so that *Jj(hj* G *Hj)* implies (41), contrary to hypothesis. This proves the claim.

Now suppose *Vk,v'*$_k$ are missing from *Hk,* and consider the multivalent clauses

$$(h_k \; eHkU \; \{vk\}) \text{ V } Vj\text{^}hj \text{ G } Hj), \quad (hk \; eHkU \text{ « } \} \text{) V } \text{V}_{i|Wk}(\text{fcj € } Hj). \tag{43}$$

They must respectively be implied by clauses $M\backslash_9\ M_2$ 6 S' because they are longer than (41). This means that the resolvent of My, M_2 on hk implies (41). So by construction of the resolution algorithm, S^f contains a clause that implies (41), contrary to hypothesis. •

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

A unit resolution algorithm for multivalent clauses appears in Fig. 9. The algorithm also accommodates all-different predicates, which can serve as disjuncts of multivalent clauses alongside terms of the form hj € Hj. Examples of the latter are constraints (4b) and (4e) of the progressive party problem. The constraint (4e) is written,

$$Si \text{ V } Sj \text{ V } m_{ijt} \text{ V } (ha \# h_{jt})$$

but can be formally written as a multivalent clause as follows.

$$(*\text{• e } \{1\}) \text{ V } (Sj \text{ € } \{1\text{»} \text{ V } (mm \text{ € } \{1\}) \text{ V } \mathbf{alldiff}(h_{it}, h_{jt}).$$

6.7 Valid and Nonvalid Logic Cuts

An intuitive understanding of a problem can suggest logic cuts, both valid and nonvalid, even when no further polyhedral cuts are easily identified. The idea of a (possibly nonvalid) logic cut was defined in [37], which gives the process synthesis example discussed here as an example. Other examples include structural design problems [10], matching problems [32], and a series of standard 0-1 problems discussed by Wilson [73].

A logic cut for an MLLP model (1) has heretofore been characterized as an implication of the logical formulas in (1). Actually any logical formula implied by the constraint set as a whole is a logic cut. That is, $g(y,h)$ is a logic cut if it is true for every (x, y,h) that satisfies the constraints of (1). For example, —13 is a logic cut for the problem

$$\begin{array}{lll} \min & \text{xi} + x_2 & \\ \textbf{s.t.} & \boldsymbol{yi} \text{ — } \textbf{(xi} \geq \textbf{1)} & \boldsymbol{yi} \text{ V } \boldsymbol{y_2} \\ & \boldsymbol{V2} \text{ - } \boldsymbol{(x_2} \geq \textbf{1)} & \\ & \boldsymbol{V3} \text{ -> } \textbf{(*i + x}_2 \leq \textbf{0)} & \end{array} \qquad (44)$$

but is not implied by the formula $y\backslash$ V y_2.

Logic cuts can be defined in an even more general sense that permits them to be nonvalid. Let (y,h) be *feasible* in (1) if (x,y,/i) 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, j/,/i) for which ex $\leq$ ex'. Then $g(y, h)$ is a logic cut if any feasible (y, h) that makes $g(y, h)$ false is dominated by a feasible (y',h') that makes $g(y\backslash h')$ true. The cut $g(y,h)$ may be added to (1) without changing the optimal solution, but it may exclude feasible solutions.

For example, the formulas -ij/i and -132 are (nonvalid) logic cuts for (44). They are nonvalid because they exclude the feasible points (1,0,0),(1, 1,0).

6.8 Logic-Based Benders Cuts

The idea behind Benders decomposition is generalized to a logic-based setting in [33], which applies the idea to 0-1 programming and the satisfiability problem. Here the idea is applied to MLLP.

```
Let 5 be a set {C{ | i E /} of multivalent clauses, where each C_i
   has the form Vj'Li(fy € H_{ij) V \/_{t€Ti} alldiff({fy | j € J_t})
Let n% be the number of terms (hj € flj) of C% vith nonempty ifj.
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 m = 0 and \Ti\ = 1 then
      Add the alldiff predicate in C_i to A and remove i from /.
   Else if n_t = 1 and |T_t| = 0 then
      Let H{j be nonempty.
      Let Z5j = Dj n JHtj and remove % from /.
While U is nonempty:
   Remove some index k from 17.
   If Dk is empty then stop; S is unsatisfiable.
   Eoralli€/:
      If iftfc is nonempty then
         If Dfc C J?tb then remove i from /.
         Else
            Let   H_ik  =  H_ikn D_k.
            If Hik is empty then
               Let rti = Ui — 1.
               If nt = l and |Ti|=0 then
                  Let iTij be nonempty and remove i from /.
                  If Dj <£ Hij then
                     Let Dj = Dj H H^ and add j to U.
               If nt = 0 and |Ti| = 1 then
                  Remove i from /.
                  Add the alldiff predicate in C_t to A.
   For each predicate alldif{{hj \ j € J}) in A with A; € J:
      If |D*| = 1 then
         For j € J \ {k}:
            If Dk C 2>, then Let Dj = Dj \ D_k and add j to C7.
```

Figure 9: A tintt re^o/ufton *algorithm for multivalent clauses.*

A Benders cut for (1) can be generated at each node of the search tree in the following way. Let the logical formulas comprise the master problem. The subproblem consists of the LP relaxation at that node; i.e., the linear constraint sets that are enforced by propositions that are true at the node. Thus if y_j is fixed to true for j G Ji, the LP relaxation is,

$$\begin{array}{lll} \min & ex & \\ \text{s.t.} & Ax > \underline{a} & (u) \\ & A\dot{>}x\_> \dot{a}^*, \ jeJ & (t^*0 \\ & x > \underline{0}. & \end{array} \tag{45}$$

where $Ax \geq a$ represents cuts added at the root node. Let u, U_j be the dual variables as shown. Then the following is a valid bound on the optimal value z of (1):

$$z \geq ua + \sum_{j \in J_1} u^j a^j.$$

The bound can also be written

$$z \geq ua + \sum_{j \in J_1} u^j a^j y_j \tag{46}$$

if each y_j takes its current value, namely 1 (for true). In fact (46) is a valid bound for any y for which u remains feasible in the dual of (45), i.e., any y for which

$$ua + \overline{\wedge 2}\ u^j A^j y_j \leq c \tag{47}$$

This gives the Benders cut,

(47) -+ (46).

This cut can be generated at any leaf node and used at any subsequent node to obtain a lower bound on the optimal value without solving the LP relaxation. If the bound is not large enough to prune the true, the LP must be solved.

In practice it is convenient to work with reduced costs. If $\bar{z}$ is the optimal value of (45) and r is the vector of reduced costs, then (46) and (47) respectively become

$$z \geq \bar{z} - \pounds\ u^j a^j (1 - \quad) \tag{48}$$

and

$$- \sum_{i \in Ji} u^j A^j (1 - {}_{W}) \leq r. \tag{49}$$

The Benders cut is

(49) - (48).

The cut generated at each node is valid throughout the search tree. It does not strengthen the LP relaxation but provides a bound on the optimal value that may obviate solution of the relaxation. This can be useful when the LP is large or hard to solve.

7 Computational Results

The aim of the computational experiments is not to compare the best possible MLLP algorithm for a given problem with the best possible competing algorithm. Rather, the aim is to isolate the effect on performance of the specific MLLP features that are illustrated by each problem. To the end, the simplest possible MLLP algorithm is compared with the simplest possible MILP algorithm.

The MLLP algorithm is that of Fig. 2, fleshed out as follows. The branching rule is to branch on the first propositional variable in the first unsatisfied logical formula. The logic processing algorithm is unit resolution. 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. 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 [34], however, that comparison with a commercial code may provide limited insight. The details of commercial implementations are not public knowledge, and even if they were, it would be difficult to isolate the factors that explain differences in performance.

MLLP has already been shown to have advantages on the chemical engineering problems, and for these problems the computational experiments reported here confirm previous work. They are reported because the confirmation of experimental results is a key element of empirical science, one that is largely neglected in the algorithmic literature.

7.1 Flow Shop Problem

The flow shop problem illustrates two advantages of MLLP: a) it can result in a smaller search tree than MILP, because the MILP representation is not integral, and b) the processing time at each node is less, because the elimination of integer variables makes the LP relaxations smaller.

As discussed in Section 5.1, there is little reason to introduce linear relaxations of the disjunctive constraints typical of scheduling problems. They are therefore omitted. If there are m jobs and n machines, this reduces the number of variables in the LP relaxation from $2rn + mn$ to $2m$.

Furthermore, the MILP model is likely to create a larger search tree, because its continuous relaxation is nonintegral. This can be seen from Corollary 1, which implies that the MILP representation of the disjunction

$$(t_k - U \geq r_{ik}) \vee (U - t_k \geq r_{ki})$$

is integral if and only if

$$\begin{aligned} &\max\{*A - U \mid U - t_k \geq r_{ki},\ (0,0) \leq (U,t_k) \leq (mi,m_k)\} = r_{ki} - M_{ki} \\ &\max\{*_t - t_k \mid t_k - U \geq r_{tfc},\ (0,0) \leq (U,t_k) \leq (m,-,m_{fc})\} = r_{ik} - M_{ik}. \end{aligned} \tag{50}$$

Defining M_{ki}, M_{ik} by (18) yields $(M_{ki}, M_{ik}) = (r_{ki} + m_{fc}, r_{ik} + mi)$. Also it is easy to see that the two maxima in (50) are respectively equal to $-r_k\{$ and $-_{t}fc$. So (50) implies that the MILP representation is integral if and only if $(r_ki, r_{ik}) = (mfc, m_t)$, which does not occur in practice.

Number of jobs	machines	MLLP nodes	time	per node	MILP nodes	time	per node	CPLEX 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

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.*

Three flow shop problems that represent process scheduling problems in a chemical plant [52] were solved, and the results appear in Table 1. MLLP generated about 60% as many nodes as MILP and used less than half as much time per node. It therefore ran 3 to 4 times as fast as MILP on these problems.

7.2 Processing Network Design Problems

The chemical processing network problems illustrate the usefulness of (nonvalid) logic cuts as well as the advantage of an MLLP approach to modeling semicontinuous variables.

First, elementary cuts can be generated for the disjunctions $y_t \cdot V\ y[$. Because of upper and lower bounds on the variables, the corresponding constraint sets can be written

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

This expands into two disjunctions that can be relaxed.

$$(* \geq fi) \text{ V } (-«,- \geq 0) \tag{51}$$

$$\begin{pmatrix} (*i > fi) \\ V \end{pmatrix} V \left(- \sum_{(i,j)\in E} *«£ 0 \right) \tag{52}$$

Because *fi* is an upper bound on z_t, the elementary cut (13) for (51) is simply $0 \geq 0$, which is useless. But the elementary cut for (52) is

$$\pounds <> J - \sum_{M_i} \sum_{(i,j)\in E} x'' \tag{53}$$
$$f_i$$

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

Furthermore, Theorem 4 implies the 0-1 formulation of the disjunction $y_t \cdot V\ y[$ is integral. It is easily checked that if 0-1 variables j/1,1/2 correspond to the two disjuncts &,y_t', then Widte) = 1/2(1/1) = 0. This suggests that the MLLP formulation with elementary cuts may provide no advantage over the traditional continuous relaxation.

Even a cursory examination of the problem yields some useful logic cuts, however. On examination of a processing network, such as the separation network of Fig. 1, it is clear 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 cuts

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

which can be written as three clauses,

$$J/i \vee -.jfc, \quad -*y\$ \vee tfe, \quad ->ys \vee yio. \tag{54}$$

These cuts 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 cuts (54) would have no effect. Experience reported in [37, 50], however, shows that the cuts can be very effective, a fact that is confirmed here.

It is possible to concoct network design problems in which unit resolution is incomplete for the logical formulas in the model when logic cuts are added. But none of the problems solved have this property, and for this reason unit resolution alone was used for logic processing.

The relaxation of the logic cuts illustrates two points. One is that the cut $y\backslash \vee -13/3$, for example, has no relaxation because 3/3 is negated. However, because $y\$ \vee y'_3$ is given in the model, the cut implies $y\backslash \vee 3/3$, which can be written as two disjunctions,

$$(z_1 \geq f_1) \vee (-z_3 \geq 0)$$
$$(z_1 \geq f_1) \vee \left(-\textstyle\sum_j x_{3j} \geq 0\right)$$

They respectively generate the elementary cuts,

$$\frac{z_1}{f_1} \geq \frac{z_3}{f_3} \tag{55}$$

$$\frac{zl}{f_1} \leq \frac{1}{M_3} \overline{\sum_{(3,j)\in E}} x_{3j}. \tag{56}$$

The second point is that (56) can be dropped because it is implied by (53) and (55).

The synthesis problems can be modified by fixing the number of units to be installed. This is accomplished with the formula,

$$\sum_i y_i = k.$$

To generate elementary cuts, the formula is written as two inequalities.

$$E.-K\geq *. \quad E;tf\geq *.$$

Elementary cuts of the form (14) for these are respectively,

where n is the number of potential units.

Experimental results for two 5-component and two 6-component problems studied in [50] 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

Problem	MLLP No cuts	MLLP +dual cuts	MLLP +elem cuts	MILP	CPLEX	MLLP +elem. cuts +logic cuts	MLLP +elem. cuts +logic cuts +logic relax.	MILP +logic cuts	CPLEX +logic cuts
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-comporient 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

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

table indicate the importance of using relaxations. The next column illustrates the expense of generating dual cuts, as discussed in Section 5.2.

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 cuts (53). The next column adds the logic cuts described above to the MLLP model but not their relaxations. The last three columns add logic cuts to the MILP and CPLEX models and elementary relaxations of them to the MLLP model.

The results suggest that adding nonvalid logic cuts 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 [50] provide a similar indication for the OSL preprocessor. The logic cuts reduce the number of nodes for MLLP, but this is not reflected in the computation times. Comparison of methods within a group suggests that, as predicted, the traditional 0-1 formulation is at least as effective as the MLLP formulation. In fact, the addition of relaxations for the logic cuts makes the MLLP approach mode sluggish than MILP.

This is an instance in which the logical point of view provides useful cuts, but logic-based modeling confers no computational advantage.

The use of propositional variables is highly advantageous, however, when semicontinuous variables are added to the problem. It is inefficient to represent semicontinuity with integer variables, for two reasons, both noted earlier: the continuous relaxation, like any linear relaxation of semicontinuity, is useless, and the 0-1 representation is nonintegral.

The 10-process and a 38-process problem described in [55] were solved. The MLLP representation (3) for the semicontinous variables was used. No relaxation was used for the resulting disjunctions because, as just noted, any relaxation is useless. Elementary cuts were generated for the disjunctions y_r- V y{. The nonvalid logic cuts described above were used in the MLLP, MILP and CPLEX models, but no relaxations were generated for them in the MLLP model.

Problem	Nodes			Seconds5		
	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**

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

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 a) to create a relaxation with traditional 0-1 variables to represent processing units, and b) to represent semicontinuity with prepositional variables without adding further cuts to the LP relaxation. 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. In any case the object here is to isolate the effect of using a logic-based versus a 0-1 representation of semicontinuity.

7.3 Warehouse Location Problems

The warehouse location problems illustrate the generation of logic cuts from a knapsack constraint. It is also a case where a logic-based formulation is less efficient than the traditional 0-1 formulation.

The formulation of elementary cuts for the disjunctive constraints $y_i \vee y_[$ is the same as in the network synthesis problems. These cuts are again facet-defining, and the 0-1 representation is again integral. The MLLP is also a little larger than the MILP model, because it contains elementary cuts for the disjunctions, and furthermore because the MILP model combines the capacity constraints with the big-M constraints, variables $Z_{\{}$. One would therefore expect an MILP formulation to have a small advantage over an MLLP formulation.

The fact that total installed warehouse capacity must accommodate total demand gives rise to the valid knapsack constraint,

$$\sum_i k_i y_i \geq \sum_j d_j. \tag{57}$$

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

Problem	No. whse	Cap. ratio	Nodes			Seconds			Seconds per node		
			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
CAP42	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

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

model:

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

Contiguous cuts can be derived from (57) as described in Section 6.5. These clauses have the form Vie/V^{*} **"&$^{\text{nave}}$ elementary relaxations,

$$\sum z_i/f_i \geq 1.$$

Seven warehouse location problems from [7] were solved, and the results appear in Table 4. Each problem has 50 demand points with a total demand of 58,268. Bach warehouse has the same capacity, and the ratio of total warehouse capacity to total demand is shown.

The contiguous 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 contiguous logic cuts are therefore useful, but as predicts!. one should use them in a traditional MILP relaxation.

Problems 1-6 in the table were solved to test the hypothesis that contiguous 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.

7.4 The Progressive Party Problem

MLLP has substantial modeling and computational advantages for this problem. Its logical notation permits a simpler statement of the constraints, as already seen in Section 2.4. The computational advantage stems primarily from the huge number of discrete variables, only a few of which correspond to linear constraints. MLLP can process them logically without enlarging the LP relaxation, which contains only a handful of constraints at each node.

The MLLP formtdation was augmented with a simple logic cut that constrains the number of host boats to be no less than the number of periods:

$$\sum_{i \in I} \delta_i \geq |T|. \tag{58}$$

This was represented by an elementary cut 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:

$$\sum_{i \in I} K_i \delta_i \geq \sum_{i \in I} \quad . \tag{59}$$

An elementary cut for this was added to the LP relaxation. Contiguous cuts were also generated for (59) and their relaxations added to the LP. Elementary cuts were not generated for the knapsack constraints (4d). The logic processing was achieved by a section of code that in effect implements the unit resolution algorithm of Fig. 9.

The MILP model was also augmented with the logic cuts (58). There was no need to add (59) because it is a linear combination of the other constraints.

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 [60] 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. The authors of [60] 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% — a)* could serve as hosts and only crews 5-15 could visit other boats (the optimal solution uses 5 hosts). The problems were solved here in their original form. When the problem contains |/| boats, they are the |/| 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 MLLP is clear and is due primarily to a much smaller LP relaxation and the speed of logic processing.

References

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

[2] Baias, E., Disjunctive programming: Cutting planes from logical conditions, in O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds., *Nonlinear Programming* 2, Academic Press (New York, 1975) 279-312.

[3] Balas, E., A note on duality in disjunctive programming, *Journal of Optimization Theory and Applications* 21 (1977) 523-527.

[4] Balas, E., Disjunctive programming, *Annals Discrete Mathematics* 5 (1979) 3-51.

Boats	Periods	Nodes		Seconds		Seconds per node	
		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	1143973	20000*	12259	54150*	0.011	2.708
10	4	28923	20000*	319	43223*	0.011	2.161

'Computation was terminated after 20,000 nodes, without finding an integer solution.

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

[5] Barth, P., *Logic-Based 0-1 Constraint Programming,* Kluwer Academic Publishers (Boston, 1995).

[6] Beaumont, N., An algorithm for disjunctive programs, *European Journal of Operational Research* 48 (1990) 362-371.

[7] Beasley, J. E., An algorithm for solving large capacitated warehouse location problems, *European Journal of Operational Research* 3 (1988) 314-325.

[8] Blair, C, Two rules for deducing valid inequalities for 0-1 problems, *SIAM Journal of Applied Mathematics* 31 (1976) 614-617.

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

[10] Bollapragada, S., O. Ghattas and J. N. Hooker, Optimal design of truss structures by mixed logical and linear programming, manuscript, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1995).

[11] Bratko, I., *PROLOG Programming for Artificial Intelligence, International Computer Science,* Addison-Wesley (1986).

[12] BULL Corporation, *CHARME VI User's Guide and Reference Manual,* Artificial Intelligence Development Centre, BULL S.A. (France, 1990).

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

[14] Chandru, V., and J. N. Hooker, Extended Horn clauses in propositional logic, *Journal of the ACM*38 (1991) 205-221.

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

[16] Colmerauer, A., H. Kanouia, R. Pasero and P. Roussel, Un système de communication homme-machine en francjais, technical report, Université d'Aix-Marseilles II, Groupe intelligence artificielle (1973).

[17] Colmerauer, A., An introduction to Prolog III, *Communications of the ACM* 33 (1990) 52-68.

[18] Cooper, M. C, An optimal fc-consistency algorithm, *Artificial Intelligence* 41 (1989) 89-95.

[19] Dincbas, M., 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.

[20] Drexl, A., and C. Jordan, A comparison of logic and mixed-integer programming solvers for batch sequencing with sequence-dependent setups, to appear in *INFORMS Journal on Computing.*

[21] Freuder, E. C, Exploiting structure in constraint satisfaction problems, in B. Mayoh, E. Tyugu and J. Penjam, eds., *Constraint Programming,* Springer (1993) 50-74.

[22] Glover, F., Surrogate constraint duality in mathematical programming, *Operations Research* 23 434-451.

[23] Granot, F., and P. L. Hammer, On the use of boolean functions in 0-1 linear programming, *Methods of Operations Research* (1971) 154-184.

[24] Haken, A., The intractability of resolution, *Theoretical Computer Science* 39 (1985) 297-308.

[25] Hammer, P. L., and S. Rudeanu, *Boolean Methods in Operations Research and Related Areas,* Springer Verlag (Berlin, New York, 1968).

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

[27] Hooker, J. N., Generalized resolution and cutting planes, *Annals of Operations R(*<arnh* 12 (1988) 217-239.

[28] Hooker, J. N., A quantitative approach to logical inference, *Decision Support Syst** nu> 4 (1988) 45-69.

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

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

[31] Hooker, J. N., Logical inference and polyhedral projection, Proceeedings, Computer Science Logic Workshop (CSL'91), *Lecture Notes in Computer Science* 626 (1992) 184-200.

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

[33] Hooker, J. N., Logic-based Benders decomposition, manuscript, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1996).

[34] Hooker, J. N., Testing heuristics: We have it all wrong, *Journal of Heuristics* 1 (1995) 33-42.

[35] Hooker, J. N., and N. R. Natraj, Solving 0-1 optimization problems with fc-tree relaxation, in preparation.

[36] Hooker, J. N., and G. Rago, Partial instantiation methods for logic programming, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1995).

[37] Hooker, J. N., H. Yan, I Grossmann, and R. Raman, Logic cuts for processing networks with fixed charges, *Computers and Operations Research* 21 (1994) 265-279.

[38] Howard, R. A., and J. E. Matheson, Influence diagrams, in R. A. Howard and J. E. Matheson, eds., *The Principles and Applications of Decision Analysis,* v. 2, Strategic Decision Group, Menlo Park, CA (1981).

[39] Jaffar, J., and J.-L. Lassez, Constraint logic programming, *Proceedings of the 14th ACM Symposium on Principles of Programming Languages,* München, ACM (1987) 111-119.

[40] Jaffar, J., and J.-L. Lassez, From unification to constraints, *Logic programming 87, Proceedings of the 6th Conference,* Springer (1987) 1-18.

[41] Jeroslow, R. E., Representability in mixed integer programming, I: Characterization results, *Discrete Applied Mathematics* 17 (1987) 223-243.

[42] Jeroslow, R. E., and J. K. Lowe, Modeling with integer variables, *Mathematical Programming Studies* 22 (1984) 167-184.

[43] Kowalski, R. A., Predicate logic as programming language, *Proceedings of the IFIP Congress,* North-Holland (Amsterdam, 1974) 569-574.

[44] McAloon, K., and C. Tretkoff, 2LP: Linear programming and logic programming, in P. van Hentenryck and V. Saraswat, eds., *Principles and Practice of Constraint Programming,* MIT Press (1995) 99-114.

[45] McAloon, K., and C. Tretkoff, *Optimization and Computational Logic,* to be published by Wiley.

[46] Puget, J.-F., A C++ implementation of CLP, Technical report 94-01, ILOG S.A., Gentilly, France (1994).

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

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

[49] Raman, R., and I. E. Grossmann, Relation between MILP modeling and logical inference for chemical process synthesis, *Computer and Chemical Engineering* 15 (1991) 73-84.

[50] Raman, R., and I. E. Grossmann, Symbolic integration of logic in MILP branch and bound methods for the synthesis of process networks, *Annals of Operations Research* 42 (1993) 169-191.

[51] Raman, R., and I. E. Grossmann, Symbolic integration of logic in mixed-integer linear programming techniques for process synthesis, *Computers and Chemical Engineering* 17 (1993) 909-927.

[52] Raman, R., and I. E. Grossmann, Modeling and conputational techniques for logic based integer programming, *Computer and Chemical Engineering* 18 (1994) 563-578.

[53] Remy, C, Programming by constraints, *Micro Systemes* No. 104 (1990) 147-150.

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

[55] Sahinidis, N. V., I. E. Grossmann, R. E. Fornari and M. Chathrathi, Optimization model for long range planning in the chemical industry, *Computers and Chemical Engineering* 30 (1989) 1049-1063.

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

[57] Sciamma, D., 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 Intelligence, Nagoya, Japan (1990) 71-76.

[58] Sheng, C.-L., *Threshold Logic,* Academic Press (New York, 1969).

[59] Simonis, H., and M. Dincbas, Propositional calculus problems in CHIP, in F. Benhamou and A. Colmerauer, eds., *Constraint Logic Programming: Selected Research,* MIT Press (Cambridge, MA, 1993) 269-285.

[60] Smith, B. M., S. C. Brailsford, P. M. Hubbard, H. P. Williams, The progressive party problem: Integer linear programming and constraint programming compared, in U. Montanari and F. Rossi, eds., *Proceedings of Principles and Pratice of Constraint Programming,* Cassis, France, Springer (1995) 36-52.

[61] Sterling, L., and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques,* MIT Press (Cambridge, MA, 1986).

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

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

[64] Turkay, M., and I. E. Grossmann, Logic-based MINLP algorithms for the optimal synthesis of process networks, *Computer and Chemical Engineering* 20 (1996) 959-978.

[65] Van Hentenryck, P., *Constraint Satisfaction in Logic Programming,* MIT Press (Cambridge, MA, 1988).

[66] Williams, H. P., Fourier-Motzkin elimination extension to integer programming problems, *Journal of Combinatorial Theory* 21 (1976) 118-123.

[67] Williams, H. P., Logical problems and integer programming, *Bulletin of the Institute of Mathematics and its Implications* 13 (1977) 18-20.

[68] Williams, H. P., Linear and integer programming applied to the prepositional calculus, *International Journal of Systems Research and Information Science* 2 (1987) 81-100.

[69] Williams, H. P., An alternative explanation of disjunctive formulations, *European Journal of Operational Research* 72 (1994) 200-203.

[70] Williams, H. P., Logic applied to integer programming and integer programming applied to logic, *European Journal of Operational Research* 81 (1995) 605-616.

[71] Wilson, J. M., Compact normal forms in proposition^ logic and integer programming formulations, *Computers and Operations Research* 90 (1990) 309-314.

[72] Wilson, J. M., Generating cuts in integer programming with families of specially ordered sets, *European Journal of Operational Research* 46 (1990) 101-108.

[73] Wilson, J. M., A note on logic cuts and valid inequalities for certain standard (0-1) integer programs, manuscript, Loughborough University Business School, Loughborough, Leicestershire LZE11 3TU, U.K. (1995).

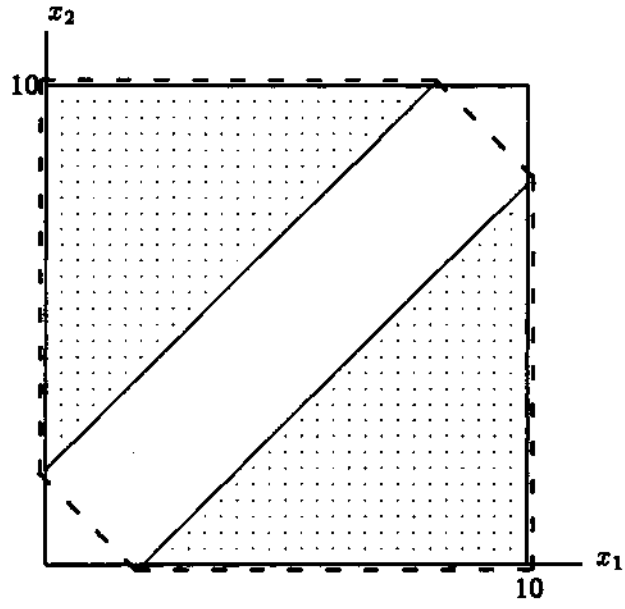

Figure 3: *Convex hull of the feasible set of a scheduling disjunction.*

An even more striking example is that of semicontinuous variables. If $0 \leq XJ \leq 4$ and the disjunction

$$(0 \leq Xj \leq 1) \vee (3 \leq Xj \leq 4)$$

is imposed, the convex hull is the entire interval [0,4]. Any conceivable relaxation is therefore useless.

5.2 Disjunctive and Dual Cuts

A relaxation of (8) can be obtained by generating valid cuts that partially or completely describe the convex hull. Balas [4] characterized valid cuts for (8) as follows. First, note that $bx \geq 0$ is a valid cut for a feasible disjunct $A^l x \geq a^1$ if and only if it is dominated by a nonnegative linear combination (or *surrogate)* of $A^l x \geq a^*$. A dominating surrogate can be written $uAx \geq$ tut, where $6 \geq$ uA, $/3 \leq ua$ and $u \geq 0$. But $bx \geq /?$ is a valid cut 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 $bx \geq ft$.

Theorem 1 (Balas) *The inequality $bx \geq 0$ is a valid cut for (8) if any only iffor each feasible system $A^*x \geq a^1$ there is a $u^* \geq 0$ such that $b \geq rfA^*$ and $/? \leq$ ti*a*.*

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

$$\max_{\backslash Uz T^1 \quad V \quad - \; t \in T^x} \{tx^t A^t\}^{>}) \; x > minima^1\}_J \qquad (9)$$

for (8), where the maximum is componentwise. Theorem 1 clearly implies that if $x \geq 0$, every valid cut is dominated by a disjunctive cut (9).

The strength and usefulness of a disjunctive cut (9) depends radically on the choice of surrogates. One could in principle generate disjunctive cuts to define every facet of the convex hull, but this is often impractical. The task of obtaining a good relaxation for (8) is in essence the task of choosing multipliers u^l judiciously.

One initially attractive choice for u^l is given by the solution of a dual problem. Each surrogate should ideally give the best possible bound on the objective function ex. That is, u^l should be chosen so that the minimum value of ex subject to $u^tA^tx \geq vta^1$ is maximized. The desired u^* is easily seen to be the optimal solution of the LP dual of min{cx | $A^*x \geq$ a*}, where u^* is the vector of dual variables. (To put it differently, the surrogate dual for linear programming is identical to the LP dual [22].)

The difficulty with this approach is that because $A^lx \geq$ a* is only a small part of the original constraint set, it may have no coupling with the objective function. That is, the variables XJ that have nonzero coefficients in ex may have zero coefficients in $A^tx \geq$ a*, and vice-versa. This means that ex provides no information to guide the choice *of u* 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 ex and $A^fx \geq$ a*. This can be done as follows. At any node of the search tree a system $Ax \geq a$ of certain linear constraints are enforced by true propositional variables. If $Ax \geq a$ is included in each term of the disjunction (8), it becomes

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

For each t one solves the dual of

$$\begin{array}{lll} \min & ex & \\ \text{s.t.} & A^*a; \geq a^* & (u^*) \\ & Ax \geq a & (u) \end{array} \qquad (10)$$

where (u^u) are the dual variables as shown. An optimal solution of the dual supplies a reasonable set of multipliers u^l for the disjunctive cut (9).

Unfortunately this approach appears to be impractical, because (10) is generally a large LP, and it is time consuming to solve the dual of (10) for each disjunct. In fact, if one branched on the disjunction by enforcing each disjunct in turn, (10) is precisely the LP one would solve at each child node. So one might as well branch on the disjunction rather than relax it. There could be some advantage in relaxing several disjunctions simultaneously, but results reported in Section 7.2 indicate that the time investment is impracticably large. The remaining discussion will therefore focus on much faster mechanisms for choosing effective multipliers u^*.

5.3 Elementary Cuts

The most common sort of disjunctive constraint (8) is one in which each disjunct is a .single inequality.

$$\bigvee_{t \in T} a^t x \geq \alpha^t. \qquad (11)$$

Beaumont [6] showed how to generate a cut for (11) that is equivalent to the continuous relaxation of the traditional 0-1 formulation of (11). The latter is

$$\begin{array}{l} a^lx \geq a^1 - M_t\{\backslash - y_t), \quad * \in T \\ \sum_{t \in T} y_t = 1 \\ 0 \leq x \leq m \\ j,_te\{o,i\}, \quad ter. \end{array} \qquad (12)$$

Each *Mt* is chosen so that a* — *Mt* is a lower bound on the value of $a^l x$. The bounds $0 \leq x \leq m$ are imposed to ensure that such a lower bound exists. It can be assumed without loss of generality that *Mt* > 0, because otherwise the inequality is vacuous and can be dropped. Beaumont obtains a cut by taking a linear combination of the inequalities in (12), where each inequality *t* receives weight I/Aft. This yields what Beaumont calls the *elementary cut* for (11),

$$\left(\sum_{t\in T} \frac{a^t}{M_t}\right) x \geq \sum_{t\in T} \frac{\alpha_t}{M_t} - |T| + 1. \tag{13}$$

Theorem 2 (Beaumont) *The elementary cut (13) is equivalent to the continous relaxation of (12). That is, the feasible set of (13) and* $0 \leq x \leq m$ *is equal to the projection of the feasible set of the continuous relaxation of (12) onto the x-space.*

One can also prove equivalence by applying Fourier elimination to (12) in order to eliminate *y*. It is easy to show that (13) and $0 \leq x \leq m$ are the resulting inequalities.

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

$$\begin{aligned} &dy \geq 6 \\ &y_t \rightarrow (a^*x \geq a_t), \quad t \in T \\ &0 \leq x \leq \text{TO}, \end{aligned} \tag{14}$$

where $d \geq 0$. It is true that (14) can be put in disjunctive form using the schema (7), but this may require a large number of disjuncts. (Disjunctions are of course a special case of $dy \geq S$ in which $6 = 1$ and each $dj \in \{0,1\}$.) The 0-1 representation of (14) is

$$\begin{aligned} &a^*x \geq a^* - M_t(l - y_t), \quad te T \\ &0 \leq x \leq m \\ &dy \geq 6 \\ &y_t e\{o,i\}, \quad te T. \end{aligned} \tag{15}$$

A linear combination of the inequalities, using weights d_t/Mtj yields the elementary cut,

$$\left(\sum_{t\in T} a^t \frac{d_t}{M_t}\right) x \geq \sum_{t\in T} \alpha_t \frac{d_t}{M_t} - \sum_{t\in T} d_t + \delta. \tag{16}$$

This is in general weaker than the continuous relaxation of (15), however. If $\wedge 2_t d_t = 6$, for example, (15) forces all the disjuncts to hold, where (16) only forces a linear combination of them to hold.

Beaumont obtains *Mt* solely from the bounds $0 \leq x \leq m$ by setting

$$a_t - M_t = \overline{53}\dot{m}\dot{n}\{O,a\$.\}mj. \tag{17}$$

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

$$at — Mt = \min_{t' \neq t} \dot{j}\min_{x} \{a^*x \mid a^{1\prime}x \geq a^*\#,\ 0 \leq x \leq m\}\dot{\backslash}_j \tag{18}$$

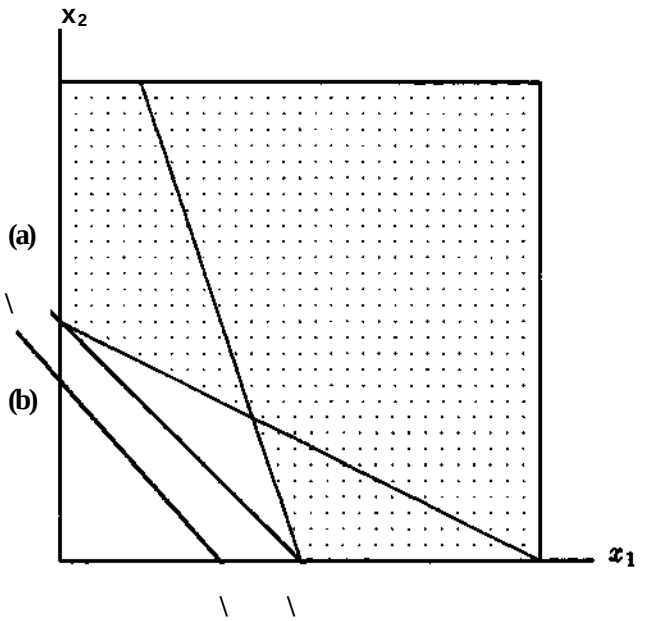

Figure 4: *A supporting elementary cut (a) and a nonsupporting elementary cut (b).*

The computation involved is negligible.

Consider for example the following constraint set, whose feasible set is the shaded area in Fig. 4.

$$(xi + 2x_2 \geq 2) \text{ V } (3xi + x_2 \geq 3)$$
$$0 \leq Xj \leq 2.$$

The 0-1 formulation is

$$x_1 + 2x_2 \geq 2 - M_1(1 - y_1)$$
$$3x_1 + x_2 \geq 3 - M_2(1 - y_2)$$
$$Jfi + V2 = 1$$
$$0 \leq x_j \leq 2, \quad y_j \in \{0, 1\}$$

Beaumont puts (Afi, M_2) = (2,3) which results in the cut $|xi + |x_2 \geq 1$. By contrast, (18) puts $(Mi, M_2) = (1,2)$, which yields the stronger cut $xi + x_2 \geq 1$. This is a *supporting* cut in the sense that it defines a supporting hyperplane for the feasible set.

Even when (18) is used to compute M*, the resulting cut may fail to be supporting. Consider the constraints (Fig. 5),

$$(-xx + 2x_2 \geq 2) \text{ V } (2xi - x_2 \geq 2)$$
$$0 \leq Xj \leq 2.$$

(17) sets (Mi, M_2) = (4,4), which results in the useless cut $xi + x_2 \geq 0$. The cut can obviously be strengthened to $xi + x_2 \geq 1$.

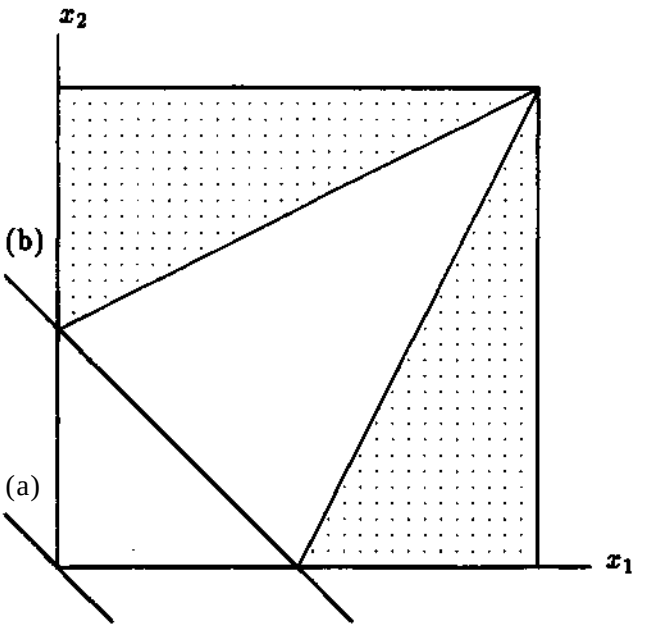

Figure 5: *An elementary cut (a) and a strengthened elementary cut (b).*

When the inequalities $a^*x \geq at$ in (14) are replaced by systems of inequalities $A^l x \geq a'$, many elementary cuts are required to achieve the effect of the traditional relaxation. Let each system $A^*x \geq a^*$ consist of inequalities $A^{ti}x \geq a\backslash$ for $t \in It$- The 0-1 formulation is

$$\begin{aligned} & A^*x \geq a^* - M^*(1 - y_t), \quad teT \\ & \mathbf{0 \leq x \leq m} \\ & \mathbf{dy \geq S} \\ & \mathbf{y_t \ e \ \{0,1\}, \ teT.} \end{aligned} \tag{19}$$

Here M^* is an array such that for each i 6 /*, $a\backslash - M\backslash$ is a lower bound on $A^u x$. Repeated applications of Fourier elimination reveal that the projection of the feasible set of (19) onto the x-space is described by the set of inequalities of the form,

$$\left(\sum_{t\in T} A^{ti_t}\frac{d_t}{M^t_{i_t}}\right)x \geq \sum_{t\in T} a^t_{i_t}\frac{d_t}{M^t_{i_t}} - \sum_{t\in T} d_t + \delta,$$

for all possible vectors $(tj, \ldots, i\backslash x\backslash)$ G /i X ... XI ^|.

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

5.4 Supporting Elementary Cuts

The example of Fig. 5 shows that an elementary cut 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 cut. 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 cut is supporting, and when it is not, to improve upon the traditional continuous relaxation the cut represents.

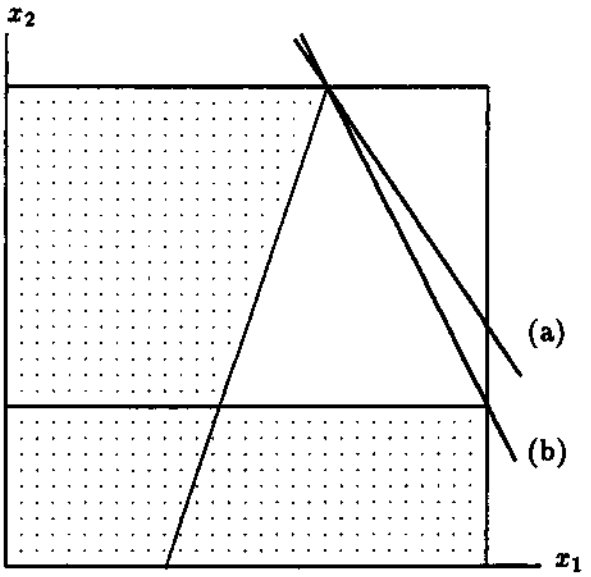

Figure 6: *A supporting elementary cut (a) and a facet-defining cut (b).*

Figures 4 and 5 may suggest that two disjuncts $a^1x \geq c^*i$, $a^2x \geq c^*2$ produce a supporting elementary cut 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. Figure 6 shows the feasible set for

$$(-3x_1 + x_2 \geq -3) \vee (-x_2 \geq -1)$$
$$0 \leq XJ \leq 3$$

The elementary cut is $3zi + 2x_2 \leq 12$, which is supporting even though (-3,1) and (0,-1) subtend an obtuse angle.

A more adequate analysis goes as follows. Let $bx \geq (3$ be the strengthened elementary cut, where bx is the left-hand side of the elementary cut (16). Because $bx \geq f3$ defines a supporting hyperplane for the feasible set of (11), /? is the smallest of the minimum values obtained by minimixing bx subject to each of the disjuncts $a^lx \geq a_t$. That is,

$$\beta = \min_{t \in T} \beta_t, \tag{20}$$

where

$$f3t = \min \{bx \mid a^lx \geq a_u \ 0 \leq x \leq 1m\}.$$

The computation of $/3_t$ is simplified if $6 \geq 0$, because in this case the upper bounds $x \leq m$ can be ignored. To this end one can introduce the change of variable,

$$\tilde{x}_j = \begin{cases} x_j & \text{if } 6i \geq 0 \\ m_j - Xj & \text{otherwise} \end{cases}$$

The strengthened elementary cut in terms of $\hat{x}$, namely $\hat{b}\hat{x} \geq /\hat{?}$, can now be computed, where $\hat{b}j = |6j|$. The right-hand side of $bx \geq /3$ can then be recovered from (20) by setting

$$\beta_t = \hat{\beta}_t + \sum_{6_j < 0} Mjbj. \tag{21}$$

It remains to compute

$$\hat{fit} = \min \{\hat{b}x \mid \hat{a}^f\hat{x} \geq d, \hat{x} \geq O\},\tag{22}$$

where

$$a_3 = \cdot \quad \_ 0 \quad \backslash \ -a) \quad \text{otherwise} \tag{23}$$

and

$$\hat{\alpha}_t = \alpha_t - \sum_{\substack{3 \\ b_j < 0}} m_j a_j^t. \tag{24}$$

Because $\hat{6} \geq 0$, LP duality applied to (22) yields that

$$\hat{fit} = \min \backslash\text{-}Jr\backslash \max\{d_t, 0\}. \tag{25}$$

This proves,

Theorem 3 *The elementary cut (16) for the disjunction (11) is supporting if and only if its right-hand side is equal to fi, as defined by (20), (21) and (25).*

5.5 Integral 0-1 Representations

The traditional continuous relaxation of a disjunctive constraint actually has two liabilities. It can be weak, as already noted, but even when it is strong, it may permit fractional solutions 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 (8) is the following.

$$\begin{aligned} &A^{**} \geq a' - M_t(1-y_t), \quad t \in T \\ &0 \leq x \leq m \\ &\sum_{t \in T} y_t = 1 \\ &y_t \in \{0,1\}, \quad t \in T, \end{aligned} \tag{26}$$

where M_t is given by (18). The claim is that when x is fixed to some value $\bar{x}$, an extreme point solution $y = \bar{y}$ of (26) can be nonintegral even when $\bar{x}$ satisfies (8). An example of this is presented by a simple semicontinuous variable, $x \in \{0\} \cup [d_1, \wedge]^*$ or

$$\begin{aligned} &(-z \geq 0) \vee (x \geq s_x) \\ &0 \leq x \leq s_2. \end{aligned}$$

The continuous relaxation of (26) is

$$\begin{aligned} &-x \geq -\wedge 2(1-y) \\ &x \geq S_1 - s_1 y \\ &0 \leq x \leq s_2 \\ &0 \leq y \leq 1. \end{aligned} \tag{27}$$

If x is fixed to $\bar{x}$ and (27) is projected onto y, the result is

$$1 - \frac{\bar{x}}{s_1} \leq y \leq 1 - \frac{\bar{x}}{s_2}, \quad 0 \leq y \leq 1. \tag{28}$$

If $0 \leq \bar{x} \leq 52$, $\bar{y} = 1 - \bar{f}$ is an extreme point solution of (28) and therefore (27), and it is nonintegral whenever $0 < \bar{x} < S2$- So (27) can have extreme point solutions with fractional y even when $\bar{x}$ G [51,62]? and even though (27) is the best possible (convex hull) relaxation of (26). The extreme point solutions for $\bar{x}$ G [^1,62] are guaranteed to have integral y only when $\$_x = 52$; 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 (26) when x is fixed to $\bar{x}$. Let the continuous relaxation of (26) be *integral* if for every $(\bar{x},\bar{y})$ satisfying (26) such that $\bar{y}$ is an extreme point of P_s, V is integral.

The following characterizes integral relaxations. A disjunct of (8) 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 (8) contains no redundant disjuncts. For t,t' G T with t ^t' define*

$$y;(t^f) = \max\{t \mid M_t y_t \leq A^t x - a^1 + M_u\ A^{1\prime} x \geq a'\ 0 \leq x \leq m,\ y_t \leq 1\}.$$

Then the continuous relaxation of (26) is integral if and only if $y_t^*(t') = 0$ *for every pair* t,t^f G T *with* $t \wedge t^9$.

Proof. It is clear that $yf(t')$ can be written,

$$yW) = \max[y_t \mid A^*x \geq a^1 - M_t(1 - y\%\ A^v x > a^{*\prime}, 0 < x < m,\ y_t < 1\}. \tag{29}$$

It is convenient to let $S_t = \{x \mid A^t x \geq a^*,\ 0 \leq x \leq m\}$ for t G T.

Claim. For any $\bar{x}$ G $S_t i$ and any t ^t',

$$y;(0 = \max_y\{2/t|yGPx\}. \tag{30}$$

Proof of claim. It suffices to show that any y_t that is feasible in (30) is feasible in (29), and vice-versa. First suppose y_t is feasible in (30). Then by letting $x = \bar{x}$ it is seen to be feasible in (29), because $A^{*\prime}\bar{x} \geq a^{*\prime}$ by virtue of the fact that $\bar{x}$ G S_t/. Conversely, let y_t be feasible in (29). To see that it is feasible in (30), set $y_t i = 1 - y\%$ and $y_t» = 0$ for $t'' \wedge t,t^f$. It is enough to show

$$\wedge'' \bar{x} \wedge a \wedge' - M t \wedge 1 - y t'') \tag{31}$$

for all t'' G T. But (31) holds for $t'' = t$ by stipulation. It holds for $t'' = t'$ because $\bar{x}$ £ S_b, and it holds for $t'' \wedge t,t'$ by definition of M_t«. This proves the claim.

Now suppose that $yJT(t') > 0$ for some t,t' with $t \wedge t^f$. Because disjunct t^1 is not redundant, one can choose $\bar{x}$ G $S_b \backslash S_t$. This implies that $j£(f) < 1$, which together with $yf(t') > 0$ means that $yt(t')$ is nonintegral. Also (30) implies that some $\bar{y}$ with $\bar{y}_t = \#(*')$ is an extreme point of $P_{\bar{x}}$. It follows that (26) is not integral.

For the converse, suppose that $y_t^*(<') = 0$ for all pairs t,t^* with $t \# t'$. It suffices to show that for any $\bar{x}$ satisfying (8), any given extreme point $\bar{y}$ of $P\$$ is integral. If it is supposed that $\bar{x}$ G $5_t/$, the following can be stated.

$$\begin{aligned} &\max\{y_t^* \mid y \,\mathrm{G}\, P^*\} = 1 \\ &\max\{y, \mid j/\mathrm{G}\, P^*\} = 0, \quad t^*t^*. \end{aligned} \tag{32}$$

The first is due simply to the fact that $\bar{x}$ G S_t^*. By the above claim, the second is equivalent to $y_t^*(t') = 0$, which is given. But (32) implies that $P\$$ is a line seqment of unit length extending from the origin in a positive direction along the $y_t t$ axis. Thus any extreme point $\bar{y}$ G P^* is integral, which means that (26) is integral. •

Corollary 1 *Consider a disjunction (11) with one inequality per disjunction and bounds* $0 \leq x \leq m$. *If (11) contains no redundant disjuncts, then (26) is integral if and only if*

$$\max\{\mathbf{a}'\mathbf{z} \mid a^{1\prime}\underline{x} > a_v,\ \underline{0} < \underline{x} < \mathbf{m}\} = \mathbf{a}_t - M_t \tag{33}$$

for every t,t' £T *with* $t \pm t^*$.

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

Corollary 2 *The continuous relaxation of (26) is integral only if the feasible sets described by the disjuncts of (8) are disjoint.*

Proof. Suppose two of the feasible sets intersect, e.g. those corresponding to disjuncts / and t'. Then $Hf(f) = 1$, which violates the condition of the theorem. •

Not even disjoint feasible sets are sufficient for integrality, as the above example shows.

5.6 Beaumont's Cuts

Beaumont [6] identified a class of facet-defining cuts for disjunctive constraints in which each disjunct consists of a single inequality, as in (11). They are facet-defining in the sense that, under certain conditions, they define facets of the convex hull of the feasible set of (11). Unfortunately, the conditions are often unsatisfied, which limits the usefulness of the cuts.

Beaumont's approach is essentially a reasonable method for choosing multipliers u^* so a> to generate a disjunctive cut (9). He first incorporates the bounds $x \leq m$ into the disjunction (11) to obtain

$$\bigvee_{t \in T} \qquad , \quad t \in T.$$

The vector of nonnegative multipliers for each disjunct is u^l = (v*,u;t), where Wt corresponds to the last inequality in the disjunct. The object is to derive a disjunctive cut $bx \geq j$ that satifies

$$\begin{aligned} b &\geq wta^t — v^l \\ 0 &\leq wtat — v^t m \end{aligned}$$

for all t. For a given w_t (yet undetermined), it is reasonable to make the components of 6 as small as possible to get a tight constraint. So let

$$b = \underset{t}{\text{minima}}^*\}, \tag{34}$$

where the minimum is taken componentwise. One can now set

$$v^{\%} = u_t a^t - 6, \quad t \in T,$$

because (34) implies $v^l \geq 0$. To make the right-hand side of the cut as tight as possible, set

$$/? = \min_{t \in T} \{u_t a_t - v^t m\} . \tag{35}$$

It remains to pick values for the w_t's. Beaumont's choice is equivalent to setting $w_t = M_t$ when Aft is derived from the variable bounds as in (17) and $a^* \leq 0$. Thus

$$w_t = \frac{}{a_t - a^t m}. \tag{36}$$

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

$$w_t = \frac{1}{a_t - \min\{a^*,0\}m'} \tag{37}$$

Theorem 5 (Beaumont) *The cut $bx \geq f3$ given by (34)-(36) is facet-defining for (11) if $a_t - a^{\%} m > 0$ for all t G T.*

Beaumont's cut can therefore be superior to a supporting elementary cut. This is illustrated in Fig. 6, where Beaumont's cut is the facet-defining cut $2x\backslash + x<i \leq 7$.

Assuming $at - a^t m > 0$ 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. x = m is often feasible, however, as in the example of Fig. 4. Here $(w\backslash_y W2)$ = (—^, —|), and one must revert to (37), which yields the useless cut $3xi + 2\hat{}2 \geq -2$.

The underlying difficulty is that Beaumont's approach has no mechanism for detecting which corner of the box $0 \leq x \leq m$ should be cut off from the feasible set. An appropriate corner could in effect be identified by using a change of variable similar to one discussed earlier, namely

$$\bar{x}_j = \begin{cases} m_j - x_j & \text{when convenient} \\ x_j & \text{otherwise.} \end{cases}$$

A "convenient" transformation would be one that makes $\hat{a}_t - \hat{a}^{\%} m > 0$ for as many disjuncts t as possible, where $\hat{a}^f$ and $\hat{a}_t$ are given by (23) and (24). This poses an integer programming problem that could be solved heuristically. However, because of the computation involved this option will not be pursued further.

5*7 Optimal Separating Cuts

One way to identify an appropriate point to be cut off by a disjunctive cut is simply to cut off the solution of the current relaxation. This is also a mechanism for using information about the objective function, because the current solution was obtained by minimizing the objective function. Fortunately it is straightforward to state a small LP problem whose solution identifies an separating cut if and only if one exists. Thus if no cut is found, the current solution is known to lie within the convex hull of the feasible set, and branching is necessary to obtain a feasible solution—unless of course the current solution is already feasible. The cut is optimal in the sense that it is chosen to maximize the amount by which the current solution violates it.

Another attraction of optimal separating cuts is that they can be generated for the case in which there are several inequalities in each disjunct. A logical constraint *gi(y,h)* other than a disjunction must, however, be first put in disjunctive form.

Suppose that the solution $\bar{x}$ of the current LP relaxation is to be separated from the feasible set of the disjunctive constraint (8). Any upper bounds $x \leq m$ should be incorporated into each disjunct of (8). Because any disjunctive cut is defined by a choice of multipliers u^1, an LP model can be formulated so as to find a set of u"s that define a cut $bx \geq 0$ that is maximally violated by $\bar{x}$. Such a model is,

$$\begin{aligned} \max \quad & f3\text{-}bx^{-} \\ \text{s.t.} \quad & /? \leq u^{l}a\backslash \quad t \in T \\ & b \geq u^{l}A\backslash \quad t\ 6\ T \\ & -\mathbf{e} \leq 6 \leq \mathbf{e} \\ & \mathbf{u}^{*} \geq 0, \quad t\ 6\ T \\ & /?, 6 \text{ unrestricted.} \end{aligned} \qquad , \qquad (38)$$

Note that the variables in the model are *f3,b,u.* The objective function measures the amount by which $\bar{x}$ violates the cut. If the objective function value is zero, there is no separating cut. The constraint $-e \leq b \leq e$, where *e* is a vector of ones, ensures that the solution is bounded. It results in no loss of generality because an optimal cut can always be rescaled to satisfy the constraint.

The model (38) has an interesting dual.

$$\begin{aligned} \min \quad & (s + i)e \\ \text{s.t.} \quad & {}^{-}\!\!\ast \quad \hat{}\, x^{t} = \quad s\text{-}t \qquad (b) \\ & \qquad t\in T \\ & \text{A'ar}^{*} \geq a^{t}y_{t}, \quad t\ 6\ T \qquad (\text{«}^{*}) \\ & \sum_{t\in T} y_{t} = 1 \qquad (\beta) \\ & s, t, x^{t}, y_{t} \geq 0, \quad t \in T \end{aligned} \qquad (39)$$

If $s - tis$ fixed to zero and $\bar{x}$ is a variable, the constraint set is Balas' convex hull representation for the disjunction (8) [4]. That is, when $s - J = 0$, the projection of the feasible set of (39) onto the $\bar{x}$-space is the convex hull of the feasible set of (8). (This is related to the fact, observed by Williams [69], that the dual of the dual of a disjunctive programming problem is the convex hull representation of the problem.) The problem (39) therefore seeks a point $YlteT^{x\ast\ m}\ \wedge$ convex hull that is closest to $\bar{x}$, as measured by the rectilinear distance.

An optimal separating cut can be superior to a supporting elementary cut. Consider the example of Fig. 6, which becomes

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

The solution of (38) is $0 = -|$, $6 = (-1,-1)$, $u^{l} = (3,0,!)$, $u^{2} = (\wedge,1,0)$, which produces the facet-defining cut $2xi + X2 \leq 7$.

The optimal separating cut need not be facet-defining, however. If the convex hull of the disjunction is the box defined by $0 \leq x_3; \leq 1$ for $j = 1,2$, the optimal separating cut for $\bar{x} = (2,2)$ is $xi + x_2 \leq 2$.

6 Logic Processing

The logical aspect of MLLP revolves primarily around two issues: What repertory of logical formulas are available to express discrete constraints? What logic processing algorithms can be used to derive implications from a set of logical formulas? After a brief discussion of these two questions, subsections will be devoted to several classes of logical formulas of increasing generality. Basic logic processing algorithms will be presented for each.

Logic cuts can also be developed manually, based on insight into the problem structure. This is discussed in Section 6.7. Finally, it is shown how logic cuts can be obtained by generalizing Benders decomposition to an MLLP context.

6.1 Logical Formulas

In theory a logical formula *g%(y, h)* can represent any function of *(y, h)* that takes the values true and false. But there are certain syntactic forms that have proved useful for expressing constraints. A few basic ones will be discussed here.

Logical clauses. A logical clause is a disjunction of *literals,* which are atomic propostions y_3 or their negations -IJ/J. Thus the expression *y*\ V -132 V *y$* is a clause, where V is an inclusive "or." In theory conjunctions of logical clauses can express any function of *y* alone (i.e., any *boolean* function), but it may be convenient to use other forms as well. Implications (e.g. *yi* —• jfe) and equivalences *(y*\ = 1/2) [a]*e readily defined in terms of clauses.

Extended clauses. These have the form, "at least fc of $L_i, \ldots, I_p$ are true," where the L/s are literals. They, too, express boolean functions but are often more convenient than clauses because they have an a quasi-arithmetical as well as a logical aspect.

Knapsack constraints. The familiar 0-1 knapsack constraint *by* >_ *13,* where each *yj* G {0,1}, can also be regarded as a logical formula that is true when the sum over *bj* for which *yj* is true is at least /?. Boolean functions of this form are called *threshold* functions and are studied in the electrical engineering literature [58]. They are difficult to process logically, but they can be used to generate logic cuts in the form of clauses and extended clauses, which are easily manipulated.

Multivalent clauses. These generalize clauses to accommodate multivalent variables and are adequate to express any bivalent function of *(y, h).* They are disjunctions of terms having the form *hj* € *H,* where *H* is a subset of the domain of *hj.*

The all-different predicate. As in the case of bivalent clauses, it is useful to supplement multivalent clauses with other types of syntax. One particularly useful formula states that a set of variables *hj* all have different values, a condition that is awkward to capture with MILP inequalities.

6.2 Logic Processing Algorithms

The goal of logic processing is to extract information that is implicit in the logical formulas 5«(j/^)« It is essentially an inference process that derives *logic cuts,* or implications, from the formulas. It can be useful in two ways:

- The logic cuts may give rise to elementary or other inequality cuts that can strengthen the LP relaxation.
- They may reduce backtracking by ruling out partial assignments that cannot be extended to feasible solutions; these are known as *redundant* assignments in the constraint satisfaction literature [63]. For instance, if there is no feasible solution $(3/1,\ldots, y_n)$ in which $(yu\ J/2) = (T, F)$, a logic cut -ij/i V y_2 would prevent one from exploring a subtree defined by (yi,]b) = *(TjF)* in order to discover this.

As noted earlier, a large body of logic processing algorithms have been developed by the research communities associated with logical inference, constraint satisfaction, constraint programming and logic programming. The discussion here is limited to two basic types of algorithms that can form the logical basis of an MLLP solver. One algorithm is fast but *incomplete* (i.e., does not derive all possible inferences), and one is a much slower, complete algorithm.

The incomplete algorithm is a simple constraint propagation technique and takes the form of *unit resolution* ("forward chaining") in the case of logical clauses. It is probably adequate for most applications, but when a more powerful inference algorithm is required, the *resolution* algorithm can be used. It is a well-known complete inference method for logical clauses and can be generalized to extended and multivalent clauses. Resolution can be quite slow and is unsuitable for problems with a large number of propositional variables. There are many applications, however, in which the number of discrete variables is small relative to the continuous part of the problem. In such cases it may be worthwhile to extract as much information as possible from the logical formulas in order to avoid solving large LP problems.

6.3 Logical Clauses

In principle a logical formula that contains only bivalent variables *yj* can always be written as a conjunction of clauses, i.e., in *conjunctive normal form* (CNF). For example, the formula (yi A 3/2) V jfc, where A means "and," can be written (3/1 V 3/3) A $(y_2$ V 3/3). Also the implication *Vi* -* *V2* (or 1/1 D 1/2) can be written -iyi V y_{2y} and the equivalence 3/1 = y_2 is rendered *hVi* Vy_2)A(yi V-.y_2).

CNF is completely expressive because any formula in bivalent variables can be regarded as a boolean (true-false) function $f(y) = /(yi,\ldots, y_m)$. If $y = v^1,\ldots, v^N$ are the values of y that make *f(y)* false, then the following CNF formula is equivalent to /(y),

$$\overset{N}{\mathbf{A}}\ \overset{m}{\mathbf{V}}\ \mathbf{«(*}\mathbf{)}.$$

where yj(t>$\overset{*}{j}$) is -iyj if $v\overset{*}{j}$ = true and is *yj* if vj = false.

This conversion to CNF requires exponential time and space in the worst case. However, a formula involving the connectives -1, V, A, —•, = (and other connectives with linear-time transformations to CNF) can be converted to CNF in linear time by adding new variables. The algorithm goes as follows. If a given formula *F* has the form *AAB*, where *A* and *B* are subformulas, then apply the algorithm to *A* and *B* separately, and conjoin the results. If *F* has the form *A* —• *B* then rewrite it as -u4v*B* and apply the algorithm to the resulting formula, and similarly for A = *B*. If jPhas the form AVfl, then write it as $(y_m+\backslash$ = A)A(y_{m+2} = 2?)A(y_{m+1} Vj/$_{m+2}$), where y_m+i, y_m+2 are variables that do not occur in F, and apply the algorithm to the resulting formula (see [71] for refinements). The algorithm stops when CNF is achieved.

In practice an MLLP solver would accept several connectives that are readily converted to CNF and make the conversion. It should normally be unnecessary to introduce new variables, because formulas are usually short enough so that a CNF equivalent without them is also short.

A simple *resolution* algorithm [47, 48, 54] derives all implications of a set of clauses. Let clause $C\backslash$ *absorb* clause C_2 when all the literals of $C\backslash$ occur in $C_2\backslash C\backslash$ implies $C<i$ if and only of $C\backslash$ absorbs $C<i$* Two clauses have a (unique) *resolvent* when exactly one variable yj occurs positively in one and negatively in the other. The resolvent is a disjunction of all literals that occur in either clause except yj and -iyj. For instance, 32 V -^3 is the resolvent of 31 V y_2 and -ijh V -1J/3. Given a set 5 of clauses, the resolution algorithm picks a pair of clauses in 5 that have a resolvent that is absorbed by no clause in 5, and adds the resolvent to 5. It repeats until there is no such pair, which occurs after finitely many iterations.

Theorem 6 (Quine [47, 48]) *A clause set S implies clause C if and only if the resolution algorithm applied to S generates a clause that absorbs C. In particular,* 5 *is unsatisfiable if and only if resolution generates the empty clause.*

This theorem will follow from a more general result to be proved in Section 6.6.

Linear relaxations can be generated for clauses derived by resolution, if desired. A clause is simply a disjunction. So if the clause contains all positive literals, and each of its variables corresponds to a linear constraint set, an elementary or other type of cut can be generated as discussed in Section 5. If a derived clause is a *unit clause,* i.e., contains a single literal yj or -ij/j, then yj can be fixed accordingly.

Resolution not only has exponential complexity in the worse case [24] but can be very slow in practice [27]. A much faster inference algorithm that sacrifices the completeness of resolution is *unit resolution.* It is the same as full resolution except that one of the parents of a resolvent is always a unit clause. Its incompleteness can be seen in the example,

$$
\begin{array}{l}
y_x \vee \quad y_2 y \quad 3/3 \\
\text{yi} \vee \neg y_2 \vee \quad y_3 \\
y\backslash \vee \quad y_2 \vee \neg y_3 \\
y_1 \vee \neg y_2 \vee \neg y_3
\end{array}
$$

Resolution fixes $y\backslash$ to true, but unit resolution does nothing because there are no unit clauses to start with. Unit resolution is efficient, however, as it runs in $O(nL)$ time, if there are n variables and L literals, and it tends to be very fast in practice. A precise algorithm that is adapted to the more general case of extended clauses 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. [13, 14, 15, 56, 62]. 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 5 are written as a system $Ay \geq a$ of 0-1 inequalities in the usual fashion; i.e., a clause $\backslash Jj_e jLj$ is written $Y^jeJVjiLj) \geq 1$, where $yj(Lj)$ is yj if $Lj = yj$ and is $1 - yj$ if $Lj = \sim^yj$.

Theorem 7 (Blair, Jeroslow, Lowe [9]) *Unit resolution finds a contradiction in the clause set S if and only if the linear relaxation of the corresponding system $Ay \geq a$ of 0-1 inequalities is infeasible.*

$Ay \geq a$ 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 $0 \geq 1$ from a nonnegative linear combination of $Ay \geq a$. Conversely, if unit resolution detects no contradiction, then the inequalities that represent the remaining clauses can be satisfied by setting each $y_3 = 1/2$.

Although LP duplicates the effect of unit resolution, the latter is preferable for logic processing because of its speed.

6.4 Extended Clauses

Extended clauses seem a particularly useful compromise between arithmetic and logic because they express the notions of "at least" and "at most" but can be efficiently processed as logical formulas. In fact, Barth's constraint-based solver for 0-1 optimization problems [5] reasons with 0-1 inequalities only after converting them to extended clauses.

An extended clause of *degree k* can be written

$$\sum_{j \in J} L_j \geq k,$$

where each L_3 is a literal. Here the sum is not an arithmetical sum but simply counts the number of literals that are true. Ordinary clauses have degree 1. To say that at most k are true, one can write

$$\sum_{j \in J} L_j \leq |J| - k,$$

and one can use two extended clauses to say exactly k are true.

A complete inference algorithm ("generalized resolution") for extended clauses was presented in [27, 30] and is refined by Barth in [5]. It uses resolution as well as a *diagonal summation,* where the latter is defined as follows. An extended clause $Y_{ljeJ} L_j \geq k + I$ *is* the diagonal sum of the set of extended inequalities $\{\wedge j \wedge L_j \geq k \mid i \in J\}$ if $J = |J| \in J \wedge$ but $\wedge^{or}$ each $i \; e \; J, \; i \; \& \; J\%$. The algorithm of [27] is applied to a set 5 of extended clauses as follows. If there are two clauses $C\backslash, C_2$ of degree 1 with a resolvent C that is implied by no extended clause in 5, such that $C\backslash$ is implied by an extended clause in 5 and similarly for C_2, then add C to 5. If there is a set E of extended clauses with a diagonal sum D that is implied by no extended clause in 5, such that each clause in E is implied by some clause in 5, then add D to 5. The algorithm continues until no more clauses can be added to 5.

Theorem 8 **([27, 30])** *A set S of extended clauses implies clause C if and only if the generalized resolution algorithm applied to S generates a clause that implies C.*

Implementation of the algorithm requires recognition of when one extended clause implies another. $Y_{ljeJi} L_j \geq \&i$ implies $Y_{Lj \in J_2} \wedge V \geq \wedge \wedge^{an(* \; on}ty \wedge$

$$l \wedge il - |\{j \; eJinJ_2| \; L_{xj} = L_{2j}\}| \leq An - k_2.$$

When all of the literals of a derived extended clause are positive and correspond to sets of inequalities, a linear relaxation can be formulated using one of the methods described in Section 5. A unit resolution algorithm for extended clauses appears in Fig. 7.

```
Let 5 be a set {^ej, Ljj >_k{ 11 G /} of extended clauses,
   where each L{j is yj or -ryj.
Let P be a stack of unit clauses, initially empty.
For each i G / with |Jt|· = fc,-:
   For each j G J% add Ii;J to U.
   Let Jt = 0.
While U is nonempty:
   Remove L{ from U.
   For each f G / with t G J{\:
      If I^t = L't then
         Let fct = · fc,-l, ·Jt = Ji\{t}.
      Else
         If fct = |Jt| then stop; 5 is unsatisfiable.
         Else
            If ki = \Ji\ + 1 then
               For each j G J; \ {<} add Iij to J7.
               Let Ji = 0.
            Else
               Let Ji = Ji\{t}.
```

Figure 7: ***A unit resolution algorithm for extended clauses.***

Linear programming is a stronger inference algorithm for extended clauses than unit resolution. For example, LP detects the infeasibility of the following inequalities, but unit resolution can do nothing with the corresponding extended clauses.

$$V\backslash + 2/2 + 3/3 \geq 2$$
$$(1 - i/i) + (i - y_2) + (i - y3) \geq 2$$

No known inference algorithm has exactly the same effect as LP on extended clauses, unless one views LP algorithms as inference algorithms. Generalized resolution is of course stronger than LP.

6.5 Knapsack Constraints

A complete inference algorithm for knapsack constraints appears in [30], and an analog of unit resolution can easily be devised for them. But they are perhaps best used as a source of logic cuts that are more easily processed, such as clauses and extended clauses. The implied clauses, for example, are identical to the well-known "covering inequalities" for the constraint, and their derivation is straightforward (e.g., [23]).

It may be more effective, however, to infer extended inequalities. Although it is hard to derive all the extended inequalities that are implied by a constraint, it is easy to derive all *contiguous cuts.* Consider a 0-1 inequality $dy \geq 6$ for which it is assumed, without loss of generality, that $d\backslash \geq d_2 \geq \ldots \geq d_n > 0$; if $dj < 0$, reverse its sign and add dj to 6. A

```
Let k = 1. * = £?=!<*; > Mast =0.
For j = 1,...,n:
   Let; s = s — dj.
   If 5 < S then
      While 5 + d_k < S:
         Let 5 = s + dk , k = k + 1.
      If k > fc^ then
         Generate the cut y\ + ...+yj>k.
         Let fciast = fc.
```

Figure 8: *An algorithm for generating all 1-cuts for a knapsack constraint dy ≥ 5 in which* di ≥ d_2 ≥ ... ≥ d_n > 0.

contiguous cut for $dy \geq S$ is one of the form,

$$\sum_{j=t}^{t+w+k-1} y_j \geq k, \tag{40}$$

where k is the degree of the cut and $w < n$ the "weakness" (tu = 0 indicates a cut that fixes all of its variables). In particular (40) is a *t-cut* because the first term is y_t. (40) is valid if and only if

$$\sum_{j=1}^{t+k-1} d_j + \sum_{j=t+w+k}^{n} d_j < \delta.$$

Furthermore,

Theorem 9 ([35]) *Every t-cut of weakness wfordy>5is implied by a 1-cut of weakness w.*

The power of all t-cuts can therefore be obtained by generating only 1-cuts. The algorithm of Fig. 8, presented in [35], does this in linear time. By way of example, the knapsack constraint

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

gives rise to the 1-cuts,

$$V\backslash + V2 \geq 1$$
$$y_1 + y_2 + y_3 \geq 2$$
$$y_1 + V2 + yz + y^* + ys \geq 3.$$

The first cut could be deleted if desired, because it is redundant of the second.

6.6 Multivalent Clauses

Multivalent clauses provide a convenient and versatile syntax for expressing logical formulas that involve multivalent variables. A multivalent clause has the form

$$\bigvee_{j=1}^{m} (h_j \in H_j), \tag{41}$$

where each *Hj* is a subset of the domain *Dj* of *hj*. For notational simplicity, it is assumed that bivalent variables *yj* are regarded as 2-valued variables *hj* that take a value of, say, 1 when *yj* is true and 0 when *yj* is false. If *Hj* is empty, the term *(hj* G *Hj)* can be omitted from (41), but it is convenient to suppose that (41) contains a term for each *j*.

Any true-false function $f(h) = f(h\backslash,\ldots, h_m)$ can be expressed as a conjunction of multivalent clauses. In particular, -<*(hj* G *Hj)* can be written *(hj* G *Dj* \ *Hj),* and *(hj* G *Hj)* -+ *(hk* G *Hk)* can be written *(hj e Dj* \ *Hj) V (hk* € *Hk).* One multivalent clause *Jj(hj* G *H\j)* implies another *Jj(hj* G Jifyj) if and only if *H\j* C #2j for each *j*. As examples of a multivalent clause consider the following formula (4a) from the progressive party problem.

$$v_{ijt} = (ha = j)$$

It is formally expressed as two multivalent clauses,

$$(v_{ijt} \text{ e } \{\text{o}\text{»}\text{v}(\text{fc}\text{«} \text{ e } \{j\})$$
$$(v_{ijt} \text{ e } \{\text{i}\text{»}\text{v}(\text{fca e } D_j \setminus \{j\}).$$

Resolution can be generalized to obtain a complete inference method for multivalent clauses. The resulting algorithm is related to Cooper's algorithm for obtaining fc-consistency for a set of constraints [18]. Given a set of multivalent clauses,

$$\bigvee_{j=1}^{n} (h_j \; eHa), \quad is \; I, \tag{42}$$

the *resolvent on hk* of these clauses is

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

Ordinary bivalent resolution is a special case. To apply the resolution algorithm to a set 5 of multivalent clauses, find a subset of 5 whose resolvent *M* is implied by no clause in 5, and add *M* to 5. Continue until no further clauses can be added to 5.

The multivalent resolution algorithm is a complete inference algorithm for multivalent clauses. The proof of the theorem uses the idea of Quine's original proof for ordinary resolution.

Theorem 10 *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 implications of 5 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 (41) be £j *Hj*\. Suppose the theorem is false, and let (41) be a longest clause implied by 5 but by no clause in S'.

Claim. At least one *Hj* in (41) is missing at least two elements; i.e., *Dj* *Hj*\ ≥ 2 for some *j*. First it is clear that no *Hj* = *Dj,* because otherwise (41) would be implied by a (in fact, every) clause in S'. Suppose contrary to the claim that every *Hj* is missing exactly one element, say *VJ.* Then *h* = *v* = (vi,.. .,t>$_m$) violates (41) and must therefore violate some clause *\/j(hj* G *H'j)* in 5', because 5' implies (41). This means each *H'j* C *Dj* \ {VJ}, so that *Jj(hj* G *Hj)* implies (41), contrary to hypothesis. This proves the claim.

Now suppose *Vk,v'*$_k$ are missing from *Hk,* and consider the multivalent clauses

$$(h_k \; eHkU \; \{vk\}) \text{ V } Vj\text{^}hj \text{ G } Hj), \quad (hk \; eHkU \text{ « } \} \text{) V } \text{V}_{i|Wk}(\text{fcj € } Hj). \tag{43}$$

They must respectively be implied by clauses $M\backslash_9\ M_2$ 6 S' because they are longer than (41). This means that the resolvent of My, M_2 on hk implies (41). So by construction of the resolution algorithm, S^f contains a clause that implies (41), contrary to hypothesis. •

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

A unit resolution algorithm for multivalent clauses appears in Fig. 9. The algorithm also accommodates all-different predicates, which can serve as disjuncts of multivalent clauses alongside terms of the form hj € Hj. Examples of the latter are constraints (4b) and (4e) of the progressive party problem. The constraint (4e) is written,

$$Si \text{ V } Sj \text{ V } m_{ijt} \text{ V } (ha \# h_{jt})$$

but can be formally written as a multivalent clause as follows.

$$(*\text{• e } \{1\}) \text{ V } (Sj \text{ € } \{1\text{»} \text{ V } (mm \text{ € } \{1\}) \text{ V } \mathbf{alldiff}(h_{it}, h_{jt}).$$

6.7 Valid and Nonvalid Logic Cuts

An intuitive understanding of a problem can suggest logic cuts, both valid and nonvalid, even when no further polyhedral cuts are easily identified. The idea of a (possibly nonvalid) logic cut was defined in [37], which gives the process synthesis example discussed here as an example. Other examples include structural design problems [10], matching problems [32], and a series of standard 0-1 problems discussed by Wilson [73].

A logic cut for an MLLP model (1) has heretofore been characterized as an implication of the logical formulas in (1). Actually any logical formula implied by the constraint set as a whole is a logic cut. That is, $g(y,h)$ is a logic cut if it is true for every (x, y,h) that satisfies the constraints of (1). For example, —13 is a logic cut for the problem

$$\begin{array}{lll} \min & \text{xi} + x_2 & \\ \textbf{s.t.} & \boldsymbol{yi} \text{ — } \textbf{(xi} \geq \textbf{1)} & \boldsymbol{yi} \text{ V } \boldsymbol{y_2} \\ & \boldsymbol{V2} \text{ - } \boldsymbol{(x_2} \geq \textbf{1)} & \\ & \boldsymbol{V3} \text{ -> } \textbf{(*i + x}_2 \leq \textbf{0)} & \end{array} \qquad (44)$$

but is not implied by the formula $y\backslash$ V y_2.

Logic cuts can be defined in an even more general sense that permits them to be nonvalid. Let (y,h) be *feasible* in (1) if (x,y,/i) 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, j/,/i) for which ex $\leq$ ex'. Then $g(y, h)$ is a logic cut if any feasible (y, h) that makes $g(y, h)$ false is dominated by a feasible (y',h') that makes $g(y\backslash h')$ true. The cut $g(y,h)$ may be added to (1) without changing the optimal solution, but it may exclude feasible solutions.

For example, the formulas -ij/i and -132 are (nonvalid) logic cuts for (44). They are nonvalid because they exclude the feasible points (1,0,0),(1, 1,0).

6.8 Logic-Based Benders Cuts

The idea behind Benders decomposition is generalized to a logic-based setting in [33], which applies the idea to 0-1 programming and the satisfiability problem. Here the idea is applied to MLLP.

```
Let 5 be a set {C{ | i E /} of multivalent clauses, where each C_i
   has the form Vj'Li(fy € H_{ij) V \/_{t€Ti} alldiff({fy | j € J_t})
Let n% be the number of terms (hj € flj) of C% vith nonempty ifj.
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 m = 0 and \Ti\ = 1 then
      Add the alldiff predicate in C_i to A and remove i from /.
   Else if n_t = 1 and |T_t| = 0 then
      Let H{j be nonempty.
      Let Z5j = Dj n JHtj and remove % from /.
While U is nonempty:
   Remove some index k from 17.
   If Dk is empty then stop; S is unsatisfiable.
   Eoralli€/:
      If iftfc is nonempty then
         If Dfc C J?tb then remove i from /.
         Else
            Let   H_ik  =  H_ikn D_k.
            If Hik is empty then
               Let rti = Ui — 1.
               If nt = l and |Ti|=0 then
                  Let iTij be nonempty and remove i from /.
                  If Dj <£ Hij then
                     Let Dj = Dj H H^ and add j to U.
               If nt = 0 and |Ti| = 1 then
                  Remove i from /.
                  Add the alldiff predicate in C_t to A.
   For each predicate alldif{{hj \ j € J}) in A with A; € J:
      If |D*| = 1 then
         For j € J \ {k}:
            If Dk C 2>, then Let Dj = Dj \ D_k and add j to C7.
```

Figure 9: A tintt re^o/ufton *algorithm for multivalent clauses.*

A Benders cut for (1) can be generated at each node of the search tree in the following way. Let the logical formulas comprise the master problem. The subproblem consists of the LP relaxation at that node; i.e., the linear constraint sets that are enforced by propositions that are true at the node. Thus if y_j is fixed to true for j G Ji, the LP relaxation is,

$$\begin{array}{lll} \min & ex & \\ \text{s.t.} & Ax > \underline{a} & (u) \\ & A\dot{>}x\_> \dot{a}^*, \ jeJ & (t^*0 \\ & x > \underline{0}. & \end{array} \tag{45}$$

where $Ax \geq a$ represents cuts added at the root node. Let u, U_j be the dual variables as shown. Then the following is a valid bound on the optimal value z of (1):

$$z \geq ua + \sum_{j \in J_1} u^j a^j.$$

The bound can also be written

$$z \geq ua + \sum_{j \in J_1} u^j a^j y_j \tag{46}$$

if each y_j takes its current value, namely 1 (for true). In fact (46) is a valid bound for any y for which u remains feasible in the dual of (45), i.e., any y for which

$$ua + \overline{\wedge 2}\ u^j A^j y_j \leq c \tag{47}$$

This gives the Benders cut,

(47) -+ (46).

This cut can be generated at any leaf node and used at any subsequent node to obtain a lower bound on the optimal value without solving the LP relaxation. If the bound is not large enough to prune the true, the LP must be solved.

In practice it is convenient to work with reduced costs. If $\bar{z}$ is the optimal value of (45) and r is the vector of reduced costs, then (46) and (47) respectively become

$$z \geq \bar{z} - \pounds\ u^j a^j (1 - \quad) \tag{48}$$

and

$$- \sum_{i \in Ji} u^j A^j (1 - {}_{W}) \leq r. \tag{49}$$

The Benders cut is

(49) - (48).

The cut generated at each node is valid throughout the search tree. It does not strengthen the LP relaxation but provides a bound on the optimal value that may obviate solution of the relaxation. This can be useful when the LP is large or hard to solve.

7 Computational Results

The aim of the computational experiments is not to compare the best possible MLLP algorithm for a given problem with the best possible competing algorithm. Rather, the aim is to isolate the effect on performance of the specific MLLP features that are illustrated by each problem. To the end, the simplest possible MLLP algorithm is compared with the simplest possible MILP algorithm.

The MLLP algorithm is that of Fig. 2, fleshed out as follows. The branching rule is to branch on the first propositional variable in the first unsatisfied logical formula. The logic processing algorithm is unit resolution. 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. 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 [34], however, that comparison with a commercial code may provide limited insight. The details of commercial implementations are not public knowledge, and even if they were, it would be difficult to isolate the factors that explain differences in performance.

MLLP has already been shown to have advantages on the chemical engineering problems, and for these problems the computational experiments reported here confirm previous work. They are reported because the confirmation of experimental results is a key element of empirical science, one that is largely neglected in the algorithmic literature.

7.1 Flow Shop Problem

The flow shop problem illustrates two advantages of MLLP: a) it can result in a smaller search tree than MILP, because the MILP representation is not integral, and b) the processing time at each node is less, because the elimination of integer variables makes the LP relaxations smaller.

As discussed in Section 5.1, there is little reason to introduce linear relaxations of the disjunctive constraints typical of scheduling problems. They are therefore omitted. If there are m jobs and n machines, this reduces the number of variables in the LP relaxation from $2rn + mn$ to $2m$.

Furthermore, the MILP model is likely to create a larger search tree, because its continuous relaxation is nonintegral. This can be seen from Corollary 1, which implies that the MILP representation of the disjunction

$$(t_k - U \geq r_{ik}) \vee (U - t_k \geq r_{ki})$$

is integral if and only if

$$\begin{aligned} &\max\{*A - U \mid U - t_k \geq r_{ki},\ (0,0) \leq (U,t_k) \leq (mi,m_k)\} = r_{ki} - M_{ki} \\ &\max\{*_t - t_k \mid t_k - U \geq r_{tfc},\ (0,0) \leq (U,t_k) \leq (m,-,m_{fc})\} = r_{ik} - M_{ik}. \end{aligned} \tag{50}$$

Defining M_{ki}, M_{ik} by (18) yields $(M_{ki}, M_{ik}) = (r_{ki} + m_{fc}, r_{ik} + mi)$. Also it is easy to see that the two maxima in (50) are respectively equal to $-r_k\{$ and $-_{t}fc$. So (50) implies that the MILP representation is integral if and only if $(r_ki, r_{ik}) = (mfc, m_t)$, which does not occur in practice.

Number of jobs	machines	MLLP nodes	time	per node	MILP nodes	time	per node	CPLEX 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

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.*

Three flow shop problems that represent process scheduling problems in a chemical plant [52] were solved, and the results appear in Table 1. MLLP generated about 60% as many nodes as MILP and used less than half as much time per node. It therefore ran 3 to 4 times as fast as MILP on these problems.

7.2 Processing Network Design Problems

The chemical processing network problems illustrate the usefulness of (nonvalid) logic cuts as well as the advantage of an MLLP approach to modeling semicontinuous variables.

First, elementary cuts can be generated for the disjunctions $y_t \cdot V\ y[$. Because of upper and lower bounds on the variables, the corresponding constraint sets can be written

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

This expands into two disjunctions that can be relaxed.

$$(* \geq fi) \text{ V } (-«,- \geq 0) \tag{51}$$

$$\begin{pmatrix} (*i > fi) \\ V \end{pmatrix} V \left(- \sum_{(i,j)\in E} *«£ 0 \right) \tag{52}$$

Because *fi* is an upper bound on z_t, the elementary cut (13) for (51) is simply $0 \geq 0$, which is useless. But the elementary cut for (52) is

$$\pounds <> J - \sum_{M_i} \sum_{(i,j)\in E} x'' \tag{53}$$
$$f_i$$

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

Furthermore, Theorem 4 implies the 0-1 formulation of the disjunction $y_t \cdot V\ y[$ is integral. It is easily checked that if 0-1 variables j/1,1/2 correspond to the two disjuncts &,y_t', then Widte) = 1/2(1/1) = 0. This suggests that the MLLP formulation with elementary cuts may provide no advantage over the traditional continuous relaxation.

Even a cursory examination of the problem yields some useful logic cuts, however. On examination of a processing network, such as the separation network of Fig. 1, it is clear 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 cuts

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

which can be written as three clauses,

$$J/i \vee -.jfc, \quad -*y\$ \vee tfe, \quad ->ys \vee yio. \tag{54}$$

These cuts 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 cuts (54) would have no effect. Experience reported in [37, 50], however, shows that the cuts can be very effective, a fact that is confirmed here.

It is possible to concoct network design problems in which unit resolution is incomplete for the logical formulas in the model when logic cuts are added. But none of the problems solved have this property, and for this reason unit resolution alone was used for logic processing.

The relaxation of the logic cuts illustrates two points. One is that the cut $y\backslash \vee -13/3$, for example, has no relaxation because 3/3 is negated. However, because $y\$ \vee y'_3$ is given in the model, the cut implies $y\backslash \vee 3/3$, which can be written as two disjunctions,

$$(z_1 \geq f_1) \vee (-z_3 \geq 0)$$
$$(z_1 \geq f_1) \vee \left(-\textstyle\sum_j x_{3j} \geq 0\right)$$

They respectively generate the elementary cuts,

$$\frac{z_1}{f_1} \geq \frac{z_3}{f_3} \tag{55}$$

$$\frac{zl}{f_1} \leq \frac{1}{M_3} \overline{\sum_{(3,j)\in E}} x_{3j}. \tag{56}$$

The second point is that (56) can be dropped because it is implied by (53) and (55).

The synthesis problems can be modified by fixing the number of units to be installed. This is accomplished with the formula,

$$\sum_i y_i = k.$$

To generate elementary cuts, the formula is written as two inequalities.

$$E.-K\geq *. \quad E;tf\geq *.$$

Elementary cuts of the form (14) for these are respectively,

where n is the number of potential units.

Experimental results for two 5-component and two 6-component problems studied in [50] 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

Problem	MLLP No cuts	MLLP +dual cuts	MLLP +elem cuts	MILP	CPLEX	MLLP +elem. cuts +logic cuts	MLLP +elem. cuts +logic cuts +logic relax.	MILP +logic cuts	CPLEX +logic cuts
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-comporient 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

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

table indicate the importance of using relaxations. The next column illustrates the expense of generating dual cuts, as discussed in Section 5.2.

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 cuts (53). The next column adds the logic cuts described above to the MLLP model but not their relaxations. The last three columns add logic cuts to the MILP and CPLEX models and elementary relaxations of them to the MLLP model.

The results suggest that adding nonvalid logic cuts 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 [50] provide a similar indication for the OSL preprocessor. The logic cuts reduce the number of nodes for MLLP, but this is not reflected in the computation times. Comparison of methods within a group suggests that, as predicted, the traditional 0-1 formulation is at least as effective as the MLLP formulation. In fact, the addition of relaxations for the logic cuts makes the MLLP approach mode sluggish than MILP.

This is an instance in which the logical point of view provides useful cuts, but logic-based modeling confers no computational advantage.

The use of propositional variables is highly advantageous, however, when semicontinuous variables are added to the problem. It is inefficient to represent semicontinuity with integer variables, for two reasons, both noted earlier: the continuous relaxation, like any linear relaxation of semicontinuity, is useless, and the 0-1 representation is nonintegral.

The 10-process and a 38-process problem described in [55] were solved. The MLLP representation (3) for the semicontinous variables was used. No relaxation was used for the resulting disjunctions because, as just noted, any relaxation is useless. Elementary cuts were generated for the disjunctions y_r- V y{. The nonvalid logic cuts described above were used in the MLLP, MILP and CPLEX models, but no relaxations were generated for them in the MLLP model.

Problem	Nodes			Seconds5		
	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**

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

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 a) to create a relaxation with traditional 0-1 variables to represent processing units, and b) to represent semicontinuity with prepositional variables without adding further cuts to the LP relaxation. 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. In any case the object here is to isolate the effect of using a logic-based versus a 0-1 representation of semicontinuity.

7.3 Warehouse Location Problems

The warehouse location problems illustrate the generation of logic cuts from a knapsack constraint. It is also a case where a logic-based formulation is less efficient than the traditional 0-1 formulation.

The formulation of elementary cuts for the disjunctive constraints $y_i \vee y_[$ is the same as in the network synthesis problems. These cuts are again facet-defining, and the 0-1 representation is again integral. The MLLP is also a little larger than the MILP model, because it contains elementary cuts for the disjunctions, and furthermore because the MILP model combines the capacity constraints with the big-M constraints, variables $Z_{\{}$. One would therefore expect an MILP formulation to have a small advantage over an MLLP formulation.

The fact that total installed warehouse capacity must accommodate total demand gives rise to the valid knapsack constraint,

$$\sum_i k_i y_i \geq \sum_j d_j. \tag{57}$$

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

Problem	No. whse	Cap. ratio	Nodes			Seconds			Seconds per node		
			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
CAP42	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

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

model:

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

Contiguous cuts can be derived from (57) as described in Section 6.5. These clauses have the form Vie/V^{*} **"&$^{\text{nave}}$ elementary relaxations,

$$\sum z_i/f_i \geq 1.$$

Seven warehouse location problems from [7] were solved, and the results appear in Table 4. Each problem has 50 demand points with a total demand of 58,268. Bach warehouse has the same capacity, and the ratio of total warehouse capacity to total demand is shown.

The contiguous 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 contiguous logic cuts are therefore useful, but as predicts!. one should use them in a traditional MILP relaxation.

Problems 1-6 in the table were solved to test the hypothesis that contiguous 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.

7.4 The Progressive Party Problem

MLLP has substantial modeling and computational advantages for this problem. Its logical notation permits a simpler statement of the constraints, as already seen in Section 2.4. The computational advantage stems primarily from the huge number of discrete variables, only a few of which correspond to linear constraints. MLLP can process them logically without enlarging the LP relaxation, which contains only a handful of constraints at each node.

The MLLP formtdation was augmented with a simple logic cut that constrains the number of host boats to be no less than the number of periods:

$$\sum_{i \in I} \delta_i \geq |T|. \tag{58}$$

This was represented by an elementary cut 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:

$$\sum_{i \in I} K_i \delta_i \geq \sum_{i \in I} \quad . \tag{59}$$

An elementary cut for this was added to the LP relaxation. Contiguous cuts were also generated for (59) and their relaxations added to the LP. Elementary cuts were not generated for the knapsack constraints (4d). The logic processing was achieved by a section of code that in effect implements the unit resolution algorithm of Fig. 9.

The MILP model was also augmented with the logic cuts (58). There was no need to add (59) because it is a linear combination of the other constraints.

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 [60] 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. The authors of [60] 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% — a)* could serve as hosts and only crews 5-15 could visit other boats (the optimal solution uses 5 hosts). The problems were solved here in their original form. When the problem contains |/| boats, they are the |/| 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 MLLP is clear and is due primarily to a much smaller LP relaxation and the speed of logic processing.

References

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

[2] Baias, E., Disjunctive programming: Cutting planes from logical conditions, in O. L. Mangasarian, R. R. Meyer and S. M. Robinson, eds., *Nonlinear Programming* 2, Academic Press (New York, 1975) 279-312.

[3] Balas, E., A note on duality in disjunctive programming, *Journal of Optimization Theory and Applications* 21 (1977) 523-527.

[4] Balas, E., Disjunctive programming, *Annals Discrete Mathematics* 5 (1979) 3-51.

Boats	Periods	Nodes		Seconds		Seconds per node	
		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	1143973	20000*	12259	54150*	0.011	2.708
10	4	28923	20000*	319	43223*	0.011	2.161

'Computation was terminated after 20,000 nodes, without finding an integer solution.

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

[5] Barth, P., *Logic-Based 0-1 Constraint Programming,* Kluwer Academic Publishers (Boston, 1995).

[6] Beaumont, N., An algorithm for disjunctive programs, *European Journal of Operational Research* 48 (1990) 362-371.

[7] Beasley, J. E., An algorithm for solving large capacitated warehouse location problems, *European Journal of Operational Research* 3 (1988) 314-325.

[8] Blair, C, Two rules for deducing valid inequalities for 0-1 problems, *SIAM Journal of Applied Mathematics* 31 (1976) 614-617.

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

[10] Bollapragada, S., O. Ghattas and J. N. Hooker, Optimal design of truss structures by mixed logical and linear programming, manuscript, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1995).

[11] Bratko, I., *PROLOG Programming for Artificial Intelligence, International Computer Science,* Addison-Wesley (1986).

[12] BULL Corporation, *CHARME VI User's Guide and Reference Manual,* Artificial Intelligence Development Centre, BULL S.A. (France, 1990).

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

[14] Chandru, V., and J. N. Hooker, Extended Horn clauses in propositional logic, *Journal of the ACM*38 (1991) 205-221.

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

[16] Colmerauer, A., H. Kanouia, R. Pasero and P. Roussel, Un système de communication homme-machine en francjais, technical report, Université d'Aix-Marseilles II, Groupe intelligence artificielle (1973).

[17] Colmerauer, A., An introduction to Prolog III, *Communications of the ACM* 33 (1990) 52-68.

[18] Cooper, M. C, An optimal fc-consistency algorithm, *Artificial Intelligence* 41 (1989) 89-95.

[19] Dincbas, M., 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.

[20] Drexl, A., and C. Jordan, A comparison of logic and mixed-integer programming solvers for batch sequencing with sequence-dependent setups, to appear in *INFORMS Journal on Computing.*

[21] Freuder, E. C, Exploiting structure in constraint satisfaction problems, in B. Mayoh, E. Tyugu and J. Penjam, eds., *Constraint Programming,* Springer (1993) 50-74.

[22] Glover, F., Surrogate constraint duality in mathematical programming, *Operations Research* 23 434-451.

[23] Granot, F., and P. L. Hammer, On the use of boolean functions in 0-1 linear programming, *Methods of Operations Research* (1971) 154-184.

[24] Haken, A., The intractability of resolution, *Theoretical Computer Science* 39 (1985) 297-308.

[25] Hammer, P. L., and S. Rudeanu, *Boolean Methods in Operations Research and Related Areas,* Springer Verlag (Berlin, New York, 1968).

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

[27] Hooker, J. N., Generalized resolution and cutting planes, *Annals of Operations R(*<arnh* 12 (1988) 217-239.

[28] Hooker, J. N., A quantitative approach to logical inference, *Decision Support Syst** nu> 4 (1988) 45-69.

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

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

[31] Hooker, J. N., Logical inference and polyhedral projection, Proceeedings, Computer Science Logic Workshop (CSL'91), *Lecture Notes in Computer Science* 626 (1992) 184-200.

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

[33] Hooker, J. N., Logic-based Benders decomposition, manuscript, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1996).

[34] Hooker, J. N., Testing heuristics: We have it all wrong, *Journal of Heuristics* 1 (1995) 33-42.

[35] Hooker, J. N., and N. R. Natraj, Solving 0-1 optimization problems with fc-tree relaxation, in preparation.

[36] Hooker, J. N., and G. Rago, Partial instantiation methods for logic programming, Graduate School of Industrial Administration, Carnegie Mellon University, Pittsburgh, PA 15213 USA (1995).

[37] Hooker, J. N., H. Yan, I Grossmann, and R. Raman, Logic cuts for processing networks with fixed charges, *Computers and Operations Research* 21 (1994) 265-279.

[38] Howard, R. A., and J. E. Matheson, Influence diagrams, in R. A. Howard and J. E. Matheson, eds., *The Principles and Applications of Decision Analysis,* v. 2, Strategic Decision Group, Menlo Park, CA (1981).

[39] Jaffar, J., and J.-L. Lassez, Constraint logic programming, *Proceedings of the 14th ACM Symposium on Principles of Programming Languages,* München, ACM (1987) 111-119.

[40] Jaffar, J., and J.-L. Lassez, From unification to constraints, *Logic programming 87, Proceedings of the 6th Conference,* Springer (1987) 1-18.

[41] Jeroslow, R. E., Representability in mixed integer programming, I: Characterization results, *Discrete Applied Mathematics* 17 (1987) 223-243.

[42] Jeroslow, R. E., and J. K. Lowe, Modeling with integer variables, *Mathematical Programming Studies* 22 (1984) 167-184.

[43] Kowalski, R. A., Predicate logic as programming language, *Proceedings of the IFIP Congress,* North-Holland (Amsterdam, 1974) 569-574.

[44] McAloon, K., and C. Tretkoff, 2LP: Linear programming and logic programming, in P. van Hentenryck and V. Saraswat, eds., *Principles and Practice of Constraint Programming,* MIT Press (1995) 99-114.

[45] McAloon, K., and C. Tretkoff, *Optimization and Computational Logic,* to be published by Wiley.

[46] Puget, J.-F., A C++ implementation of CLP, Technical report 94-01, ILOG S.A., Gentilly, France (1994).

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

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

[49] Raman, R., and I. E. Grossmann, Relation between MILP modeling and logical inference for chemical process synthesis, *Computer and Chemical Engineering* 15 (1991) 73-84.

[50] Raman, R., and I. E. Grossmann, Symbolic integration of logic in MILP branch and bound methods for the synthesis of process networks, *Annals of Operations Research* 42 (1993) 169-191.

[51] Raman, R., and I. E. Grossmann, Symbolic integration of logic in mixed-integer linear programming techniques for process synthesis, *Computers and Chemical Engineering* 17 (1993) 909-927.

[52] Raman, R., and I. E. Grossmann, Modeling and conputational techniques for logic based integer programming, *Computer and Chemical Engineering* 18 (1994) 563-578.

[53] Remy, C, Programming by constraints, *Micro Systemes* No. 104 (1990) 147-150.

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

[55] Sahinidis, N. V., I. E. Grossmann, R. E. Fornari and M. Chathrathi, Optimization model for long range planning in the chemical industry, *Computers and Chemical Engineering* 30 (1989) 1049-1063.

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

[57] Sciamma, D., 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 Intelligence, Nagoya, Japan (1990) 71-76.

[58] Sheng, C.-L., *Threshold Logic,* Academic Press (New York, 1969).

[59] Simonis, H., and M. Dincbas, Propositional calculus problems in CHIP, in F. Benhamou and A. Colmerauer, eds., *Constraint Logic Programming: Selected Research,* MIT Press (Cambridge, MA, 1993) 269-285.

[60] Smith, B. M., S. C. Brailsford, P. M. Hubbard, H. P. Williams, The progressive party problem: Integer linear programming and constraint programming compared, in U. Montanari and F. Rossi, eds., *Proceedings of Principles and Pratice of Constraint Programming,* Cassis, France, Springer (1995) 36-52.

[61] Sterling, L., and E. Shapiro, *The Art of Prolog: Advanced Programming Techniques,* MIT Press (Cambridge, MA, 1986).

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

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

[64] Turkay, M., and I. E. Grossmann, Logic-based MINLP algorithms for the optimal synthesis of process networks, *Computer and Chemical Engineering* 20 (1996) 959-978.

[65] Van Hentenryck, P., *Constraint Satisfaction in Logic Programming,* MIT Press (Cambridge, MA, 1988).

[66] Williams, H. P., Fourier-Motzkin elimination extension to integer programming problems, *Journal of Combinatorial Theory* 21 (1976) 118-123.

[67] Williams, H. P., Logical problems and integer programming, *Bulletin of the Institute of Mathematics and its Implications* 13 (1977) 18-20.

[68] Williams, H. P., Linear and integer programming applied to the prepositional calculus, *International Journal of Systems Research and Information Science* 2 (1987) 81-100.

[69] Williams, H. P., An alternative explanation of disjunctive formulations, *European Journal of Operational Research* 72 (1994) 200-203.

[70] Williams, H. P., Logic applied to integer programming and integer programming applied to logic, *European Journal of Operational Research* 81 (1995) 605-616.

[71] Wilson, J. M., Compact normal forms in proposition^ logic and integer programming formulations, *Computers and Operations Research* 90 (1990) 309-314.

[72] Wilson, J. M., Generating cuts in integer programming with families of specially ordered sets, *European Journal of Operational Research* 46 (1990) 101-108.

[73] Wilson, J. M., A note on logic cuts and valid inequalities for certain standard (0-1) integer programs, manuscript, Loughborough University Business School, Loughborough, Leicestershire LZE11 3TU, U.K. (1995).