LISTE DES ARTICLES PROPOSES POUR LA THESE

h) *Une Condition d'Optimalité en Programmation en Nombres Entiers*, M. Guignard, RFIRO, (1971), 108-113.

i) *The State Enumeration Method for Mixed 0-1 Programming*, M. Guignard, K. Spielberg, IBM Philadelphia Scientific Center Report 320-3000, 1971.

j) *Mixed-Integer Algorithms for the 0-1 Knapsack Problem*, M. Guignard, K. Spielberg, IBM J. Research and Development, (1972), 424-430.

k) *A Realization of the State Enumeration Procedure*, M. Guignard, K. Spielberg, IBM Philadelphia Scientific Center Report, 320-3025, 1973.

l) *Zero-One Zero-One Programming and Enumeration*, M. Guignard, K. Spielberg, in "Nonlinear Programming" ed. O.L. Mangasarian, R.R. Meyer, S.M. Robinson, Academic Press, (1975), 333-360.

m) *An Experimental Interactive System for Integer Programming*, M. Guignard, K. Spielberg, Proceedings of the Bicentennial Conference on Mathematical Programming, SIGMAP, 1976, NBS, (1978), 328-337.

n) *Maintenance Scheduling*, M. Guignard, K. Spielberg, Proceedings of the 9th International Symposium on Mathematical Programming, Budapest 1976.

p) *Reduction methods for State Enumeration Integer Programming*, M. Guignard, K. Spielberg, Annals of Discrete Mathematics, 1 (1977) 273-285.

q) *Algorithms for Exploiting the Structure of the Simple Plant Location Problem*, M. Guignard, K. Spielberg, Annals of Discrete Mathematics, 1 (1977) 247-271.

r) *Propagation, Penalty Improvement and use of logical Inequalities in Interactive 0-1 Programming*, M. Guignard, K. Spielberg, Operations Research Verfahren, XXV, teil 1, (1977) 158-171.

s) *Preferred Variables and Improved Penalties in 0-1 Programming*, M. Guignard, Technical Report n° 29, Revised Dec. 76, Dept. of Statistics, Wharton-School.

t) *Survey of Enumerative Methods for Integer Programming*, M. Guignard, K. Spielberg, U. Suhl, Proceedings, Share 51, 1978.

u) *A Direct Dual Method for the Mixed Plant Location Problem with some side constraints*, M. Guignard, K. Spielberg, Math. Prog., 17 (1979), 198-227.

v) *A Direct Dual Approach to a Transshipment Formulation for Multi-Layer Network Problems with Fixed Charges*, M. Guignard, K. Spielberg, Dept. of Stat., Wharton School, Report 43, 1979.

w) *Logical Reduction Methods in 0-1 Programming*, M. Guignard, K. Spielberg, accepted for publication in J. ORSA.

x) *Fractional Vertices, Cuts and Facets of the Simple Plant Location Problem*, Math. Prog. Study on Combinatorial Optimization, 12 (1980).

R.I.R.O.
(5e année, R-2, 1971, p. 108-113)

UNE CONDITION D'OPTIMALITE EN PROGRAMMATION EN NOMBRES ENTIERS

par Monique GUIGNARD (1)

Résumé. — *On démontre une condition nécessaire d'optimalité pour un problème de maximisation en variables entières d'une fonction convexe sur un domaine non linéaire. On trouvait déjà cette condition dans* [1] *pour le cas linéaire et dans* [2] *pour le cas non linéaire, mais une hypothèse de régularité du cône linéarisant les contraintes à l'optimum avait été faite. Cette hypothèse n'est pas nécessaire.*

Je tiens à remercier M. Huard qui m'a suggéré l'idée de cette démonstration.

Définitions et notations

Si C est un convexe de R^n, et si $\varphi : C \to R$, φ est quasi-convexe sur C si $\{ x \in C : \varphi(x) \leqslant \lambda \}$ est convexe, $\forall \lambda \in R$.

Si $C \subset R^n$, $Co(C)$ désignera l'enveloppe convexe de C.

Q désigne l'ensemble des rationnels et M_i^j est l'élément (i, j) de la matrice M.

Théorème 1

Soit C un convexe de R^n, $K \subset R^n$, $\varphi : R^n \to R$ une fonction quasi-convexe sur C.

Si $\bar{x} \in C \cap K$ maximise φ sur $C \cap K$, alors $\bar{x}$ maximise φ sur $Co(C \cap K)$.

Soit $P = Co(C \cap K)$. $\forall x \in P$,

— ou $x \in C \cap K$ et $\varphi(x) \leqslant \varphi(\bar{x})$.

— ou $x \notin C \cap K$ et d'après le théorème de Carathéodory ((4), p. 4-2)

$$\exists\, \alpha_0, \ldots, \alpha_n \quad , \quad \alpha_i \in (0, 1) \quad , \quad \Sigma \alpha_i = 1$$
$$\exists\, x_0, \ldots, x_n \quad , \quad x_i \in C \cap K \quad , \quad x = \Sigma \alpha_i x_i.$$

(1) Maître-Assistante, Laboratoire de Calcul, CUSLA, M3, Lille.

Supposons que $\varphi(x) > \varphi(\bar{x})$.

φ étant quasi-convexe sur C,

$$\varphi(x) = \varphi(\Sigma\alpha_i x_i) \leqslant \operatorname{Max}_i \varphi(x_i) = \varphi(x_j)$$

et il existerait au moins un point de $C \cap K$, à savoir x_j, tel que

$$\varphi(x_j) \geqslant \varphi(x) > \varphi(\bar{x})$$

ce qui est impossible.

Il faut donc que

$$\varphi(x) \leqslant \varphi(\bar{x}).$$

Théorème 2

Soit un cône Γ défini comme $\{ x \in R^n : Bx \leqslant b \}$, avec rang $(B) = r$. Soit $\bar{x}$ un point tel que $B\bar{x} = b$. Alors

— si $r = n$, ou $\Gamma = \{ \bar{x} \}$

ou Γ est l'enveloppe convexe de ses génératrices extrêmes.

— si $r < n$, $\Gamma = \Gamma_0 + \Delta$

où $\Delta = \{ x \in R^n : Bx = b \}$ est une variété linéaire de dimension $n - r$,

et où $\Gamma_0 = \{ x \in R^n \cap \Delta^* : Bx \leqslant 0 \}$ est un cône de dimension r, si Δ^* est un sous-espace vectoriel supplémentaire de la variété linéaire Δ.

Γ_0 dans R^r est un cône polyédrique régulier, égal à l'enveloppe convexe de ses génératrices extrêmes.

$r = n$ Le cône polyèdrique Γ a un sommet $\bar{x}$ ((3), p. 388), il est saillant ((4), p. 12-4), alors

— ou il ne contient pas de génératrice : $\Gamma = \{ \bar{x} \}$,

— ou il contient au moins une génératrice et il est l'enveloppe convexe de ses génératrices extrêmes ((4), p. 12-15) qui sont des facettes d'ordre 1 ((3), p. 388).

$r < n$ Γ n'a pas de sommet.

$\Delta = \{ x \in R^n : Bx = b \}$ est une variété linéaire de dimension $n - r$.

Soit Δ^* un sous-espace vectoriel supplémentaire.

$R^n = \Delta \oplus \Delta^*$, où $\oplus$ représente la somme directe dans R^n, donc $\forall x \in R^n$, $\exists y \in \Delta, \exists z \in \Delta^* \ni x = y + z$.

Si $x \in \Gamma$: $Bx \leqslant b$,

$y \in \Delta$: $Bx = b$,

$$B(x - y) \leqslant 0 \Rightarrow Bz \leqslant 0 \Rightarrow z \in \Delta^* \cap \{ x : Bx \leqslant 0 \}.$$

Soit $\Gamma_0 = \{ z \in \Delta^* : Bz \leqslant 0 \}$. Γ_0 est un cône de dimension égale à r, (dim $(\Delta^*) = r$, rang $(B) = r$), c'est donc un cône régulier qui dans R^r est l'enveloppe convexe de ses génératrices extrêmes.

Théorème 3

Si Γ est un cône polyédrique d'équation $Bx \leqslant b$, où

$$B_i^j \in Q \text{ et } b_i \in Q, \forall_i, \forall_j,$$

alors, si $\bar{x} \in Z^n$ tel que $B\bar{x} = b$,

$$\Gamma = Co(\Gamma \cap Z^n)$$

1 : $r = n$

ou $\Gamma = \{ \bar{x} \}$ si $\bar{x}$ vérifie $B\bar{x} = b$,

et $\Gamma = Co(\Gamma \cap Z^n)$,

ou $\Gamma = Co(G)$ où G est l'ensemble des génératrices extrêmes G_i de Γ. Une génératrice extrême est une facette d'ordre 1, i.e. vérifie un système d'équations du type

$$B_I x = b_I, \text{ où rang } (B_I) = n - 1.$$

Ce système d'équations admet une infinité de solutions entières de la forme

$$x = \overset{0}{x} + \alpha y, \alpha \in Z,$$

où $\overset{0}{x}$ est une solution entière particulière, par exemple $\bar{x}$, et où y, à composantes entières, vérifie $B_I y = 0$ (cf. résolution d'un système linéaire en entiers).

Donc $\forall G_i$ génératrice extrême, $G_i \cap Z^n = \{ x \in R^n : x = \bar{x} + \alpha y, \alpha \in N \}$ et $G_i = Co(G_i \cap Z^n) = \{ x \in R^n : x = \bar{x} + \alpha y, \alpha \in R_+ \}$.

$$\left.\begin{array}{l} G \subset Co(\Gamma \cap Z^n) \subset \Gamma \\ \\ \Gamma = Co(G) \end{array}\right\} \Rightarrow \boxed{\Gamma = Co(\Gamma \cap Z^n)}$$

2 : $r < n$

$\Gamma = \Gamma_0 + \Delta$

où $\Gamma_0 = \{ x \in R^n : x \in \Delta^*, Bx \leqslant 0 \}$,

et $\Delta = \{ x \in R^n : Bx = b \}$

1. rang $(B) = r < n \Rightarrow \forall x \in \Delta \cap Z^n, x = \overset{0}{x} + Vt, t \in Z^n$,

où $\overset{0}{x}$ est une solution entière particulière, par exemple $\bar{x}$, et où V, matrice $(n, n - r)$, est à cœfficients entiers, de rang $n - r$, et vérifie $BV = 0$,

i.e. $$V \in \text{Ker } (B).$$

$$\forall x \in \Delta, x = \bar{x} + Vt, t \in R^n,$$

puisque $V \in \text{Ker}\,(B)$ et rang $(V) + \dim(B) = n$, et

$$\forall\, x \in \Delta,\ x \in Co(\Delta \cap Z^n).$$

Donc

$$\boxed{\Delta = Co(\Delta \cap Z^n)}$$

2. Pour prouver que Γ_0 est l'enveloppe convexe de ses points entiers, il suffit de montrer que l'on peut se ramener au théorème 3.1. Or

$$\Gamma_0 = \{\, x \in R^n : x \in \Delta^*,\ Bx \leqslant 0 \,\}.$$

Δ^* étant un sous-espace vectoriel de dimension r, il existe une application linéaire $F : R^n \to R^{n-r}$ telle que

$$\Delta^* = \{\, x \in R^n : Fx = 0 \,\},\ \text{et rang}\ (F) = n - r.$$

Alors, si on pose $A = \begin{bmatrix} B \\ F \\ -F \end{bmatrix}$, on peut définir Γ_0 comme

$$\{\, x \in R^n : Ax \leqslant 0 \,\}.$$

Pour que l'on se trouve dans un cas d'application de la première partie, il suffit que

1. rang $(A) = n$,
2. $A_i^j \in Q, \forall\, i, j,$

il existe en effet $\hat{x} = 0$ vérifiant $A\hat{x} = 0$.

Soit I un ensemble d'indices tel que

$$|I| = r \text{ et rang } (B_I) = r.$$

$F_1, \ldots, F_p$ ayant été déterminés, $p = 0, \ldots, n-1$, on peut déterminer F_{p+1} tel que

1. ${}^tB_j \cdot F_{p+1} = 0, \forall\, j \in I,$ et ${}^tF_i \cdot F_{p+1} = 0, i = 1, \ldots, p.$
2. $F_{p+1}^k \in Q, k = 1, \ldots, n.$

Dans ces conditions,

$$\boxed{\Gamma_0 = Co(\Gamma_0 \cap Z^n)}$$

3. $\Gamma = \Delta + \Gamma_0 = Co(\Delta \cap Z^n) + Co(\Gamma_0 \cap Z^n)$
$= Co(\Delta \cap Z^n + \Gamma_0 \cap Z^n) \subset Co(\Gamma \cap Z^n) \subset .$

car · $Co(A) + Co(B) = Co(A + B)$ ((4), p. 4-8)

et · Γ est convexe, donc $Co(\Gamma \cap K) \subset \Gamma, \forall K \subset R^n$,

donc

$$\boxed{\Gamma = Co(\Gamma \cap Z^n)}$$

Théorème 4

Soient $\varphi : R^n \to R$, $a : R^n \to R^m$, $\bar{x} \in Z^n$ tel que $a(\bar{x}) \leqslant 0$.

On suppose a et φ différentiables en $\bar{x}$, et $[\nabla a(\bar{x})]_i^j \in Q, \forall_i, \forall_j$.

Soient $A = \{ x : a(x) \leqslant 0 \}$ et $T(\bar{x}) = \{ x \in R^n : \nabla a(\bar{x}) \cdot (x - \bar{x}) \leqslant 0 \}$.

Si φ est quasi-convexe sur $T(\bar{x})$ et si $T(\bar{x}) \subset A$, alors

$$\left\{ \begin{array}{l} a(x) \leqslant 0 \\ \\ x \in Z^n \end{array} \right\} \Rightarrow \varphi(x) \leqslant \varphi(\bar{x}) \Bigg\} \Rightarrow \left\{ \begin{array}{l} \exists\, u \geqslant 0 \\ \\ \nabla \varphi(\bar{x}) = u \cdot \nabla a(\bar{x}). \end{array} \right.$$

Soit $P = Co(T(\bar{x}) \cap Z^n)$.

Alors

1. $\left.\begin{array}{l} \bar{x} \text{ maximise } \varphi \text{ sur } A \cap Z^n \\ \\ \bar{x} \in T(\bar{x}) \subset A \end{array}\right\} \Rightarrow \left.\begin{array}{l} \bar{x} \text{ maximise } \varphi \\ \text{sur } T(\bar{x}) \cap Z^n \\ \text{(théorème 1)} \end{array}\right\} \Rightarrow \bar{x}$ maximise φ sur P.

2. $[\nabla a(\bar{x})]_i^j \in Q$
$\bar{x} \in Z^n$ vérifie $\nabla a(\bar{x}) \cdot (\bar{x} - \bar{x}) = 0 \} \Rightarrow T(\bar{x}) = P.$
(théorème 3).

3. $\left.\begin{array}{l} 1. \\ \\ 2. \end{array}\right\} \Rightarrow \bar{x}$ maximise φ sur $T(\bar{x})$.

4. Les conditions de Kuhn et Tucker sur $T(\bar{x})$ s'écrivent

$$\exists\, u \geqslant 0 : \nabla \varphi(\bar{x}) = u \nabla a(\bar{x}).$$

REMARQUE

Si a est pseudo-concave et différentiable en $\bar{x}$, l'hypothèse : $T(\bar{x}) \subset A$ est vérifiée, car

$$\left.\begin{array}{l} \nabla a(\bar{x}) \cdot (x - \bar{x}) \leqslant 0 \Rightarrow a(x) - a(\bar{x}) \leqslant 0 \\ \\ a(\bar{x}) \leqslant 0 \end{array}\right\} \Rightarrow a(x) \leqslant 0.$$

EXEMPLE

$$a(x) = \begin{bmatrix} -x_1^3 + x_2 + \sqrt{2} \\ -x_2 - \sqrt{2} \end{bmatrix} \leqslant 0, \bar{x} = [1, -1], \nabla a(\bar{x}) = \begin{bmatrix} -3 & 1 \\ 0 & -1 \end{bmatrix}$$

$$T(\bar{x}) = \{ x : -3x_1 + x_2 \leqslant -4, x_2 + 1 \geqslant 0 \} \subset A.$$

Donc $u \geqslant 0 : \nabla \varphi(\bar{x}) = u \cdot \nabla a(\bar{x})$.

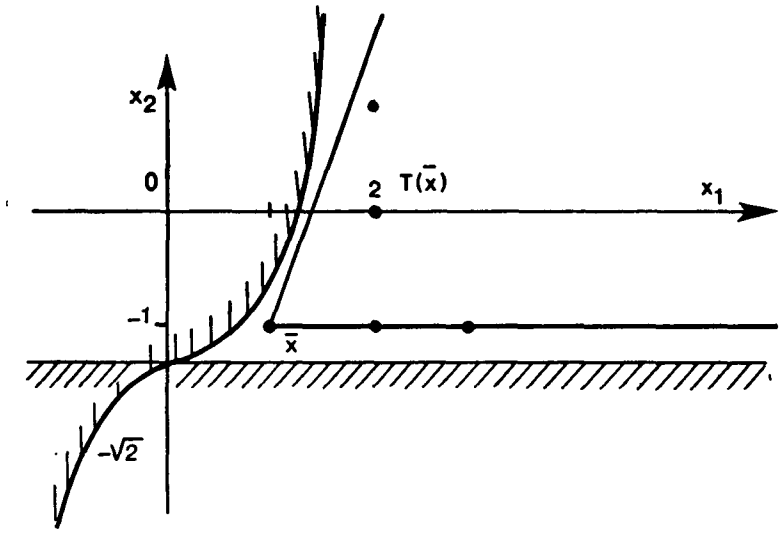


Figure 1

REFERENCES

[1] J. FREHEL, *Une méthode de trocature pour la programmation en nombres entiers*, Étude n° FF2-0072-0, février 1969.

[2] M. GUIGNARD, *Contraintes additionnelles en variables bivalentes*, Publications du Laboratoire de Calcul de la Faculté des Sciences de Lille, rapport n° 20, mars 1970.

[3] M. SIMMONARD, *Programmation linéaire*, Dunod, 1962.

[4] G. COQUET, *Ensembles convexes de R^n*, Publications du Laboratoire de Calcul de la Faculté des Sciences de Lille, rapport n° 12, juin 1968.

THE STATE ENUMERATION METHOD

FOR

MIXED ZERO-ONE PROGRAMMING

M. GUIGNARD* , K. SPIELBERG**

Presented at the 7th Mathematical Programming Symposium,

The Hague - The Netherlands, September 1970.

* Laboratoire de Calcul, Faculté des Sciences de l'Université de Lille - France

** IBM Philadelphia Scientific Center, currently at Th. J. Watson Research Center, Yorktown Heights, USA.

1 . Introduction

The State Enumeration Method was first outlined in [1]. It appears to us to be the natural generalization of enumerative methods, such as described in [2], [3], [4] and [5], for mixed zero-one programming problems :

PROBLEM P :

$$\begin{aligned} \min \quad \zeta &= \delta \,.\, \xi + \gamma \,.\, \eta \\ & D \,.\, \xi + C \,.\, \eta \leq \beta \\ & \xi \geq 0, \quad \eta_k \in \{0,1\} , \quad k \in K = \{1, 2, \ldots, p\} . \end{aligned} \tag{1}$$

It may be assumed that neither the set of continuous components ξ_j, nor that of bivalent components η_k (or K) is void.

"State Enumeration" proceeds via a tree search of considerable flexibility (a highly desirable feature in view of the numerical results reported in [6]), which is essentially guided by linear inequalities obtained from a sequence of problems replacing P.

In this note we shall describe a particular realization of the method, both for the general and for a very special problem (the Knapsack problem). Numerical examples and results will illustrate the difficulties inherent in and the potential of such an algorithm.

2 . Tree Search, State

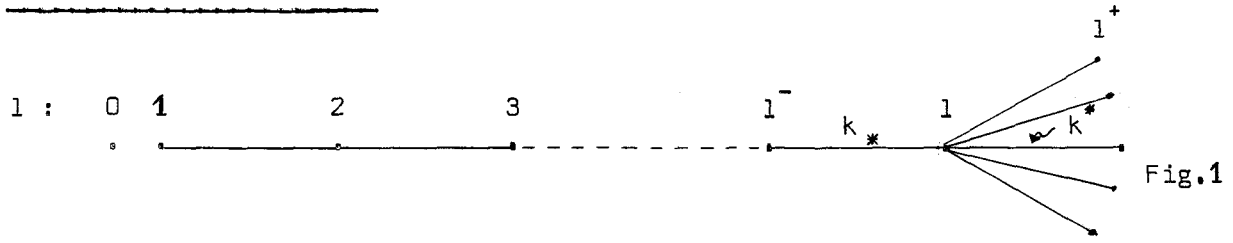


Fig.1

The structure of the algorithm is illustrated best, though not very precisely, by the simple sketch of Fig.1. An explicit step (iteration) of the algorithm involves either

(i) moving from a given "level" l of the search tree to a successor level l^+ (fixing a variable n_{k^*} at 0 or 1 at l^+, depending on the "state" at l), or

(ii) moving from l to the predecessor level l^- (backup), thereby fixing n_{k^*} at l^- to the value complementary to the one it had at l.

The pseudo-level 0 serves as a formal predecessor to level 1. The search terminates when all possibilities have been explored at level 1, i.e., when a backup to $1^- = 0$ would be indicated.

The various "branches" at l other than k^* indicate other "free" variables, i.e. variables which can yet be fixed at either 0 or 1. A major objective of the algorithm is to rule out or "cancel" otherwise feasible branches by means of various tests, again usually dependent on the inequality constraints.

The <u>state</u> at l is a decomposition of the integer variables $k \in K$:

$$K^l = (E, Z, F1, F2, F3)^l \qquad (2)$$

It serves primarily to determine the precise meaning of branching (forward steps) and of a <u>basic state problem</u> SP^l . SP^l is P with the values :

Fixed variables :

$$k \in E^l : \; n_k = 1 \quad , \quad k \in Z^l : \; n_k = 0$$

Free variables (3)

$$k \in F1^l : \; n_k = 1 \quad , \quad k \in F2^l : \; n_k = 0$$

$$k \in F3^1 \; : \quad 0 \leq \eta_k \leq 1$$

On forward steps, variables in F1 are fixed at 0, those in F2 at 1. Variables in F3 are usually, at least as long as possible, not chosen as branch variables.

The enumerative algorithm differs from a single-branch branch-bound algorithm primarily in the fact that the state problems are more restricted than P in the variables of F1 and F2. The branch-bound problems are in contrast always relaxed in all free variables.

Note the following basic facts (see e.g. [1] and [5] for details) :

(i) The totality of all SP^1 yields all feasible solutions to (1).

(ii) Each state problem can be used to yield inequalities which are global conditions on the integer variables η_k, and must either be satisfied by all feasible solutions :

$$b \cdot \eta \leq p \qquad (4)$$

or by those which are to give objective functions Z better than a known bound Z^*

$$a \cdot \eta < Z^* + p \qquad (5)$$

For example, these may be inequalities of the type introduced by Benders [7], with the coefficients a, b, p obtained from the reduced costs of the final tableau of SP^1 ; (4) being obtained for infeasible, (5) for feasible SP^1.
Or (4) may represent inequalities obtained from the group problems considered by Gomory and Johnson ([8]), etc..

Such inequalities may also be derived from the occasional solution of the auxiliary problem AP^1, which is the relaxed version of SP^1, in which all $0 \leq \eta_k \leq 1$, $k \in F1^1 + F2^1 + F3^1$.

3 . The basic Algorithm

1 . Given at 1^- :

The State S^{1^-} , k_* , Z^* ;

a solution to SP^{1^-} and the associated inequality, or a set of inequalities

$$A \quad . \quad \eta \quad \leqslant \quad p_A + Z^* . e \qquad (6)$$

$$B \quad . \quad \eta \quad \leqslant \quad p_B$$

(where $e^\tau = (1, 1, \ldots, 1)$ of appropriate size).

2 . Determine the State S^1 by either

(i) derivation from the state at 1^-
(for $1^- = 0$ one may impose an initial state), or :

(ii) an "educated" guess, for example :
solution of an auxiliary problem + rounding, limited application of other mixed integer methods : cutting plane pivot steps (Gomory), CBV method (Huard) , etc..

A "good" state may lead to new integer solutions and usually yields relatively strong inequalities.

3 . Solve SP^1

Record integer solution, if any.
Construct and save the associated inequalities.

4 . Exploit the associated inequalities or accumulated set of inequalities by "complete reduction" (for an earlier version of this, see [4]).
By this we mean the following simple procedures for combining the given inequalities with the hyper-cube constraints

$$0 \quad \leqslant \quad \eta_k \quad \leqslant \quad 1.$$

(i) For purposes of fixing variables or finding backup conditions (all variables fixed or inequality infeasible) :
Change all coefficients to non-negative form by means of complementation (transformation : $\eta_k \rightarrow \bar{\eta}_k = 1 - \eta_k$). Fix a variable at zero if its coefficient exceeds the right hand side.

Example : $-5\eta_1 + 3\eta_2 - 4\eta_3 + 2\eta_4 \leq -7$

$5\bar{\eta}_1 + 3\eta_2 + 4\bar{\eta}_3 + 2\eta_4 \leq 2$

$\rightarrow \eta_1 = \eta_3 = 1, \quad \eta_2 = 0.$

(ii) For purposes of finding a branch variable k^* :

With each row i of (6) determine a preferred set $\pi(i)$ of indices corresponding to a derived inequality

$$\sum_{j \in \pi(i)} \tilde{\eta}_j \geq 1 \qquad (7)$$

where $\tilde{\eta}_j$ is either η_j or $\bar{\eta}_j$ (which in the examples will be indicated by a + or - sign associated with the index).

This determination can be done

a) independent of the state (it might even be used as one means of determining it), or

b) dependent on the current or some imposed state.

In either case, substitute the values of the fixed variables (and of Z^*). Then render all coefficients of F1 + F2 + F3 for case a, those of F3 only for case b, negative. This transformation (and the state in case b) determines the nature of the $\tilde{\eta}_j$. If the resultant right hand side is non-negative, a relatively rare and unimportant case, skip i. *If it is negative and there are no negative left hand side coefficients, the inequality can not be satisfied (→ backup in the tree search). If no negative left hand side coefficient is larger than the right hand side, then the set of indices corresponding to negative coefficients constitutes π (i).

Otherwise, delete the largest negative coefficient and subtract it from the right hand side (imposing the hyper-cube constraint).
This leaves the right hand side negative and permits a continuation of the procedure at *above.

In this simple fashion one derives relatively strong constraints (7) (few terms). The minimal $\pi(i)$ over i, identifies the "strongest" constraint, or as we may say the "preferred row" i^* and a "preferred set" $\pi(i^*)$, except for the case that no row yields a preferred set (which is rare enough to be unimportant and can be treated by ad hoc heuristic methods).

At level 1 only variables in $\pi(i^*)$, an often relatively small subset of all free variables, need ever be considered, so that one may have identified an important reduction of the region to be searched. k^* may then e. g. be chosen as the variable with smallest negative coefficient, or by some other criterion, according to the effect of the choice on the inequalities.

Examples :

$$-5\eta_1 + 3\eta_2 - 4\eta_3 + 2\eta_4 + 7\eta_5 \leq -1$$

Case a (no imposed state)

$$-5\eta_1 - 3\bar{\eta}_2 - 4\eta_3 - 2\bar{\eta}_4 - 7\bar{\eta}_5 \leq -13$$

$$-5\eta_1 - 3\bar{\eta}_2 - 4\eta_3 \qquad - 7\bar{\eta}_5 \leq -11, \text{ etc. ,}$$

$$\rightarrow \qquad -5\eta_1 \qquad\qquad\qquad -7\bar{\eta}_5 \leq -4$$

$$\rightarrow \qquad \eta_1 + \bar{\eta}_5 \geq 1 : \text{either } \eta_1 = 1 \text{ or } \eta_5 = 0$$

$$\pi = (1, -5)$$

Case b : imposed state $F1 = \{1,4\}$, $F2 = \{2\}$, $F3 = \{3,5\}$

Introduce $\bar{\eta}_j$ for $j \in F1$, then proceed as above in regard to variables in F3 only.

$$5\bar{\eta}_1 + 3\eta_2 - 4\eta_3 - 2\bar{\eta}_4 + 7\eta_5 \leq 2$$

$$5\bar{\eta}_1 + 3\eta_2 - 4\eta_3 - 2\bar{\eta}_4 - 7\bar{\eta}_5 \leq -5$$

$$5\bar{\eta}_1 + 3\eta_2 - 4\eta_3 \qquad - 7\bar{\eta}_5 \leq -3$$

$$\pi = (3, -5)$$

In the first case one might decide to branch on $k^* = 1$ (setting $\eta_1 = 1$ on a forward step), preferring to keep variables in F3 as long as they show no tendency to become fractional (as one does in branch-bound methods).

5. If there exists k^*, selected from among the $k \in \pi(i^*)$ or among the $k \in F$ if $\pi(i^*)$ is non-existent, branch on k_* from 1^- to 1, replace 1^- by 1, S^{1^-} by S^1, k_* by k^*, etc., and resume at 1.

6. If not, i.e. if $\pi(i^*)$ (or F) has been exhausted, then exploit the accumulated inequalities at level 1^- in exactly the some fashion as detailed in 4 to effect cancellations at 1^- and to discover the possibility of a backup from 1^- to 1^{--}.
In case of a backup, one replaces 1 by 1^-, 1^- by 1^{--}, recomputes the new S^{1^-} by simple computations and reference to a few items stored in memory (such as k_*, the branch from 1^- to 1), and resumes at 1 (unless $1^- = 0$, which indicates termination).

If there is no backup, one computes a new k_* in a manner exactly analogous to that of 4, and resumes at 1.

4 . Implementation . An Example.

The basic state enumeration algorithm leaves open the question of strategy, i.e. the exercice of particular choices among various options (e.g. in the selection of k_* and k^*, etc.). An implementation of the algorithm with a set of strategies easily available to the user is a non-trivial task.

We have some experience with an experimental all-Fortran code developed with the support of IBM France (J. M . Gauthier), and more recently and importantly with an experimental system developed by the Philadelphia Scientific Center (formerly the New York Scientific Center) of IBM.

This system, NYLPS/MIPIS, is a set of Fortran subroutines which utilizes an ØS version of the linear programming system LPS and is itself designed to execute certain standard tasks of mixed integer programming, such as the fixing of bounds or variables, the storage and retrieval of data for tree-search or other methods, etc..

Within this system it was rather easy to implement a number of branch-bound algorithms (Dakin algorithm, Land-Doig algorithm ; both for "single branch" as well as "multi-branch" trees (see [5] or [9] for this terminology)). The coding of an efficient state enumeration algorithm is considerably more complicated and we still have an inadequate understanding of what constitutes an appropriate set of strategies, especially for large problems (the largest problem which we have tried to solve, with partial success , had 368 rows and 397 variables, with 25 integer variables, a problem which has been solved by a mixed integer code described in [10]).

A well known test problem of size (6,12), i.e. with 6 rows and 12 variables (all zero-one), given in [2] , was solved by the resolution of 11 linear programs with a total of 57 pivot steps. It is clear that the set of inequalities generated depends on the particular strategy chosen. In this particular case the 8 following inequalities were generated, after omission of duplicate inequalities (which is easy) . The optimal objective function for the continuous problem was 6.85, that for the zero-one problem was 13.

	η_1	η_2	η_3	η_4	η_5	η_6	η_7	η_8	η_9	η_{10}	η_{11}	η_{12}	
①	3.54	1.82			9.73		6.74		7.13		4.62	1.22	$\leq Z^*-6.85$
②		5	6		12	7		3	1	-8		5	≤ 8
③	6.92	8.94	-33.12	-13.55	3.77	17.77		-9.02		10.18			$\leq Z^*-35.19$
④		-1	- 4.36	- 1.46	1		-8.64	1.27	-1.36	3.82	1	5.73	-7 ≤ -9.55
⑤		.667	-12	.333	1		7	-1			3	-5	-1 ≤ -5.67
⑥		-4.67		- 3.33	4			-3	-5	-1		1	1 ≤ -3.33
⑦	5	1	3	2	6	4	7	2	4	1	1	5	$\leq Z^*-0$
⑧		5.67	-6	.333	13	14	-1	3	1	-5	-5	4	≤ 2.33

These inequalities are of course generated during the solution process, in the order given above. Nevertheless it is instructive to ask oneself what to do with them. The simple procedures of the given algorithm ("complete reduction") need not, of course, be effective enough to yield immediate results. In the example above, one requires knowledge of some value for Z^*, or fixed values for some variables, as are introduced during the search .

Suppose the value of Z^* is known to be 13, as became manifest in iteration 6. Then, for example, inequality ① implies $\eta_5 = \eta_7 = \eta_9 = 0$. After that one might compute the preferred sets. E. g. :

$\pi(1)$: $-3.54\ \bar{\eta}_1 - 1.82\ \bar{\eta}_2 - 4.62\ \bar{\eta}_{11} - 1.22\bar{\eta}_{12} \leq 13 - 6.85 - 11.20 = -5.05$

$\pi(1) = (-1, -11)$

Similarly : $\pi(2) = (-3, -6, 10)$, $\pi(3) = (3, -6)$, $\pi(4) = (6,12)$

$\pi(5) = (3, -6)$, $\pi(6) = (2,4,11)$, $\pi(7) = (-1, -9, -12)$, $\pi(8) = (3, -6)$

Thus, there are several possibilities for the preferred set $\pi(i^*)$, among which one may choose according to a secondary criterion. At any rate, it is clear that one need not explore more than two possibilities, such as ($\eta_1 = 0$ or $\eta_{11} = 0$) or ($\eta_3 = 1$ or $\eta_6 = 0$) etc., at level 1.

In the actual program the first branch was $\eta_{10} = 1$ (the above information was not available, of course).

Consider now the situation after this possibility has been explored. The search has returned to level 1 and $\eta_{10} = 0$. When $\eta_{11} = 0$ was tried, the auxiliary linear program became infeasible, which proved $\eta_{11} = 1$. With this information, the search terminates ; for :

① $3.54\ \eta_1 + 1.82\ \eta_2 + 1.22\ \eta_{12} \leq 1.53 \rightarrow \eta_1 = \eta_2 = 0$

④ $4.36\ \bar{\eta}_3 + 1.46\ \bar{\eta}_4 + 8.64\ \bar{\eta}_6 + 1.36\ \bar{\eta}_8 + 7\bar{\eta}_{12} \leq -9.55 - 5.73 + 22.82 = 7.54$

$\rightarrow \quad \eta_6 = 1$

One now has : $\eta_1 = \eta_2 = \eta_5 = \eta_7 = \eta_9 = \eta_{10} = 0, \quad \eta_6 = \eta_{11} = 1$

⑤ $12\ \bar{\eta}_3 + .333\ \eta_4 + \bar{\eta}_{12} \leq -5.67 - 7 + 5 + 12 + 1 = 5.33 \rightarrow \eta_3 = 1$

⑥ $3.33\ \bar{\eta}_4 + 5\ \bar{\eta}_8 + \eta_{12} \leq -3.33 - 1 + 8.33 = 4 \rightarrow \eta_8 = 1$

⑦ $2\ \eta_4 + 5\ \eta_{12} \leq 13 - 3 - 2 - 1 = 3 \rightarrow \eta_{12} = 0$

⑧ $.333\ \eta_4 \leq 2.33 + 6 - 14 - 3 + 5 = -3.67$ infeasible.

Ideas similar to "complete reduction" have recently also been proposed in [11].

5. Selected Results - Discussion

It is clear that some of the elements of the state enumeration scheme can be applied also to branch-bound methods. In particular one may attempt to utilize inequalities for the fixing of variables, exactly as outlined in 3.4.

In Table 1 we give results for three small problems, of sizes (6,12) and (28,35) with all 12 and 35 variables bivalent, and size (28,89) with the first 31 of the 89 variables bivalent.

The methods used were :

1. Branch-bound single-branch
2. Branch-bound multi-branch
3. As 1, plus Benders inequalities
4. As 2, plus Benders inequalities
5. State enumeration.

Entries - mean that the corresponding problem was not run. Otherwise, the first number indicates how many linear programs had to be resolved before termination, The second number is the total number of pivot steps.

TABLE 1. ZERO-ONE MIXED INTEGER PROBLEMS

PROBLEM / METHOD	(6,12)	(28,35)	(28,89)31
1. BB, S-B	17,102	-	-
2. BB, M-B	-	59,629	153,1612
3. BB, S-B + INEQU.	25,86	65,183	97,752
4. BB, M-B + INEQ.	13,82	77,253	171,1221
5. SE	11,57	90,663	-

The results for the state enumeration method are still disappointing, we believe due to inadequate strategy, but our use of inequalities is evidently beneficial. In considering such preliminary results, one must keep a number of items and possibilities in mind.

1 . Our programs have so far not exploited the option of impressing an initial state. The potentiel benefits of such an approach were demonstrated in [6] and [12]. Also, in the case of very large problems one may be forced to resort to heuristic exploration schemes.
One natural way of doing this, is to select "search origins" (impressed states") at random, or in some pseudo-random fashion, and explore the vicinity of such origins (e.g. by limiting the level 1 of the search). Branch-bound approaches do not appear to be well suited for such a task.

2.. There are obvious possibilities for extracting more information from the collected sets of inequalities. Among these one may mention :

(i) Deducing new inequalities by methods other (and possibly stronger) than complete reduction. Cf. [14].

(ii) Ranking, replacing and using inequalities in accordance with certain "figures of merit" which may be designed to give an approximate notion of the strength of an inequality.

(iii) All-integer programming over a given set of inequalities (possibly to be interrupted after a prespecified no. of pivot steps).

(iiii) Linear programming over the inequalities, (may be necessary instead of iii when the inequalities of [8] introduce continuous slack variables).

Evidently this list can be enlarged further. The solution of large mixed integer programming problems will require considerable ingenuity in such directions.

6 . The Knapsack Problem

An important problem, both in itself and as an example of what can be done with problems of special structure, is the Knapsack problem.

$$\begin{aligned} \min \quad & \sum c_k \eta_k = Z \\ & \sum b_k \eta_k + S = r \end{aligned} \qquad (8)$$

$$S \geq 0, \quad \eta_k \in \{0,1\}, \qquad k \in K = (1,2,\ldots,p)$$

The c_k and b_k may be taken to be positive, and the ratios c_k/b_k are assumed ordered :

$$c_k/b_k \leq c_k/b_{k+1}, \quad k = 1, 2, \ldots, p-1 \qquad (9)$$

The solution procedure is somewhat different from that given for the general problem. At a given level l we first resolve the linear program (auxiliary problem AP^l), i.e. we consider K^l partitioned into $(E, Z, F)^l$. We may assume that this problem is feasible (infeasibility would permit a cancellation of branch k^* from $\bar{l}$ to l). The solution is always trivial and determined by the index $f \in F$ of the one variable which becomes fractional. (If, by chance, this variable takes on an integral value, then one may have a new solution, and one certainly may discard k^*). F is partitioned into the sets L (indices $k < f$), $\{f\}$ and R (indices $k > f$). In the linear programming solution, the variables η_j, $j \in L$, have value 1, those with $j \in R$ have value 0.

Example : $c = -(15, 27, 10, 15; 18; 10, 41, 32, 62, 70)$

$r = 74$, $b = (11, 25, 10, 15; 20; 12, 50, 40, 80, 100)$

LP solution : $f = 5$, $L = \{1, 2, 3, 4\}$, $R = \{6, 7, 8, 9, 10\}$, $E = Z = \emptyset$

$$\eta = (1, 1, 1, {}^{13}/_{20}, 0, 0, 0, 0, 0)$$

Of course, one may obtain an integral solution by replacing the fractional variable by zero ((i)"first approximation"). One may consider two more simple approximations) as follows :

(ii) round η_f to zero. Then scan right and try to refill the Knapsack (as cheaply as possible, as usual).

(iii) round η_f to one. This renders the solution infeasible. Scan left and set, consecutively, all variables η_j, $j \in L$, one at a time only, to zero. If L is void, of course, there is no feasible solution of this type. Otherwise, one usually finds feasible solutions and takes the best. Or, dropping any one variable leaves the solution infeasible, for any $j \in L$. In that case one sets $\eta_j = 0$ for the right most index in L and resumes the scan from this point. After one has found a good feasible solution, one may also exhibit the associated linear programming solution by letting the rounded down variable (index f_3) assume the fractional value which renders $S = 0$.

In the case of the above example one exhibits three LP solutions η^1, η^2, η^3.

(i) $(1,1,1,1, {}^{3}/_{20}, 0,0,0,0,0)$, $f_1 = 5$, $Z_1 = -67$

(ii) $(1,1,1,1,0,1, {}^{1}/_{50}, 0,0,0)$, $f_2 = 7$, $Z_2 = -77$

(iii) $(1,1, {}^{3}/_{10}, 1,1,0,0,0,0,0)$, $f_3 = 3$, $Z_3 = -75$.

The objective function values Z_1, Z_2, Z_3 are those of the corresponding feasible integer solutions, obtained by setting the fractional components η_{f_1}, η_{f_2} and η_{f_3} equal to zero, respectively. Note that in case (ii) one has adjoined f to Z, in case (iii) one adjoins f to E.

One may associate a Benders type inequality with each case. It has the form

$$\sum h_k \eta_k \leq Z^* + p \qquad (10)$$

$$p = \sum_{LUE} h_k - Z_\lambda , \quad h_k = \left(c_k - \frac{c_f}{b_f} b_k\right)$$

In the above Z_λ is the objective function of the linear program (i.e. with variable $\eta_5 = {}^3/_{20}$ for (i), $\eta_7 = \frac{1}{50}$ for (ii) and $\eta_3 = \frac{3}{10}$ for (iii)). L is the index set of the free variables to the left of $f_1 = 5$, $f_2 = 7$, $f_3 = 3$, respectively.

L and R, in a certain sense, play the role of F1 and F2 in the general case. F3 is considered void. Note that one has always a value for Z^* (unless the initially posed problem is infeasible).

The Knapsack problem is also an ideal vehicule for testing some of the important ideas of Gomory and Johnson [8]. We are grateful to them for making this material available to us before its general dissemination. We also want to acknowledge the generous advice of Ellis Johnson, given freely in many discussions.

Starting with a linear programming solution with fractional variable η_f, and corresponding index sets E, Z, L, R, {f}, we may write the well known group problem as follows :

$$\min \quad -\sum_{LUE} h_k \bar{\eta}_k + \sum_{RUZ} h_k \eta_k - \left({}^{c_f}/_{b_f}\right) S \qquad (11)$$

$$\frac{1}{b_f}\left\{-\sum_{LUE} b_k \bar{\eta}_k + \sum_{RUZ} b_k \eta_k\right\} + \frac{1}{b_f} S \equiv v_f \pmod 1,$$

$$\eta_k \in \{0,1\}, \quad k \in LUR, \qquad S \geq 0$$

in which the complementary variables $\bar{\eta}_k = 1 - \eta_k$ have been introduced for

$k \in$ LUE , and v_f is the correspondingly modified right hand side of the constraint.

Gomory and Johnson show (in [8]) how one may obtain valid and relatively strong inequalities of the form

$$\sum_{k \in \text{LUE}} \pi_k \bar{\eta}_k + \sum_{k \in \text{RUZ}} \pi_k \eta_k + \pi^{\pm} S \geq 1 \quad , \qquad (12)$$

by taking the faces of tabulated corner polyhedra for cyclic groups (see [13]) and "interpolating" in various ways. The numerical results below were obtained for the simplest possible case involving the two faces of a cyclic group of order two. One may well expect that the utilization of higher order groups, not in principle very much more complicated, could lead to substantially superior results.

After the π_k and $\pi^{\pm}$ have been obtained, one may in the case of the Knapsack Problem eliminate the slack variable S between (8) and (12), to arrive at an inequality of the form (4). We also use the obvious inequality

$$\sum c_k \eta_k < Z^* \qquad (13)$$

which is of the form (5).

To get an idea as to the comparative merit of algorithms and the utility of approximations and inequalities, we implemented eight algorithms within a modular system of subroutines.

1 . A branch-bound algorithm of multi-branch type (i.e., with as many pending modes as are generated stored and retrieved according to objective fonction value. Approximation (i) used. No "penalty" computation (as is customary in branch-bound codes).

2 . The same as above plus "group penalties", computed from a cyclic group of order seven at the suggestion of E.Johnson. Customary penalties could be expected to be weaker, so that any reasonable branch-bound code would have an efficiency somewhere between 1 and 2.

3 . The state enumeration algorithm with approximation (i) and corresponding Benders inequalities.

4 . State enumeration with approximation (i) and Gomory-Johnson inequalities (computed from the cyclic group of order 2) as described earlier.

5 . State enumeration with approximation (i) and Benders as well as Gomory-Johnson inequalities 6, 7, 8. Same as 3, 4, 5 except with all approximations (i), (ii) and (iii).

In table 2 we give, for each method, results for 12 Knapsack problems of small size (10 zero-one variables only). A number of 75 and 100 variable problems were also run and do not cause any particular difficulty.
The 12 problems consist of 3 different sets of constraint data, each of which was run with the 4 right hand sides 74, 80, 139 and 250.

When there is one entry in a place of the table, it denotes the no of linear programs which were executed during the solution process.

The second entry, below the first, gives the no. of inequalities which were generated and utilized. Where there are two numbers below the first entry, they give the number of Benders inequalities and the number of Gomory-Johnson inequalities, respectively.

- ... number of LP's or inequalities was not recorded

L ... large number.

As can be seen, the state enumeration algorithm performs creditably. The use of inequalities substantially reduces the no. of required linear programs. Some of the best results, resolution of problems within execution of 1 or 2 or 3 linear programs, depend on the structure of the problem (as exploited in the approximations).
Nevertheless, there appear to be good prospects for the intelligent use of techniques similar to those utilized in this study.

TABLE 2 : 12 KNAPSACK PROBLEMS , RESOLVED BY METHODS 1 - 8

PROBLEM	1	2	3	4	5	6	7	8	9	10	11	12
OBJ.FUNCTION	77	80	127	215	79.654	84.015	147.42	260.97	73.55	73.55	145.65	219.20
METHOD												
1. BB	26	40	54	30	16	32	30	96	18	18	18	16
2. BB + GR.P.	6	14	18	10	14	32	30	78	16	18	14	16
3. SE , (i) BENDERS	11 3	27 5	34 5	14 5	10 3	11 3	12 2	62 6	15 2	24 2	8 2	10 2
4. SE , (i) GOMORY,JOHNSON	13 31	16 35	20 38	16 35	7 -	6 -	12 -	26 -	2 -	1 -	1 -	2 -
5. SE (i) B , G - J	10 5,16	14 5,19	11 5,18	8 5,13	5 4,7	4 4,6	7 3,8	21 5,35	2 2,2	2 2,2	1 2,2	2 2,2
6. SE (i) (ii) (iii) B	2 3	4 5	3 5	2 3	2 4	8 4	6 4	33 11	4 3	9 3	4 2	16 3
7. SE (i) (ii) (iii) G - J	6 15	- -	6 15	4 12	4 8	4 8	8 18	19 61	1 3	1 3	1 3	1 3
8. SE (i) (ii) (iii) B , G - J	2 5,2	3 5,6	3 5,6	1 4,2	1 4,2	3 6,6	3 4,6	16 13,32	1 3,2	1 3,2	1 2,2	1 3,2

[1] GUIGNARD, M. , K. SPIELBERG,
"Search Techniques with Adaptive Features for certain Integer and Mixed Integer Programming Problems", Proceedings, IFIPS Congress, 1968

[2] BALAS, E. ,
"An Additive Algorithm for Solving Linear Programs with zero-one Variables", Op. Res. , Vol. 13 , 1965

[3] GEOFFRION, A. M. ,
"Integer Programming by Implicit Enumeration and Balas Method", Siam Review, Vol. 9 , 1967

[4] LEMKE, C. E. , K. SPIELBERG,
"Direct Search Algorithms for zero-one and Mixed Integer Programming", Op. Res. , Vol. 15 , N° 5 , 1967

[5] BALINSKI, M. L. , K. SPIELBERG,
"Methods for Integer Programming : Algebraic, Combinatorial and Enumerative," Progress in Operations Research, Vol. III , editor J. Aronofsky, John Wiley & Sons, 1969

[6] SPIELBERG , K. ,
"Plant Location with Generalized Search Origin" , Manag. Sci. , Vol. 16 , N° 3 , 1969

[7] BENDERS, J. F. ,
"Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol. 4 , 1962

[8] GOMORY, R.E. , JOHNSON, E. L.
"Some Continuous Functions related to Corner Polyhedra and their Application to Integer Programming", 7th Mathematical Programming Symposium, the Hague , 1970

[9] COLMIN, J. , K. SPIELBERG,
"Branch and Bound Schemes for Mixed Integer Programming", Technical Report No. 320-2972, Philadelphia Scientific Center, IBM , 1969

[10] BENICHOU, M. , GAUTHIER, J.M. , GIRODET, P. , HENTGES, G. , RIBIERE, G. , VINCENT, O.
"Experiments in Mixed-Integer Linear Programming",
7th Mathematical Programming Symposium, The Hague , 1970

[11] GRANOT, F. , HAMMER, P.
"On the Use of Boolean Functions in 0 - 1 Programming"
7th Mathematical Programming Symposium, the Hague, 1970

[12] SALKIN, H. , SPIELBERG, K.
"Adaptive Binary Programming",
Technical Report No. 320 - 2951 , Philadelphia Scientific Center,
IBM , 1958

[13] GOMORY, R.E. ,
"Some Polyhedra Related to Combinatorial Problems",
Linear Algebra and its Applications, Vol. 2 , 1969

[14] GUIGNARD, M. ,
"Contraintes additionnelles en variables bivalentes",
Publication Laboratoire de Calcul, Université de Lille,
Rapport N° 20 , Mars 1970

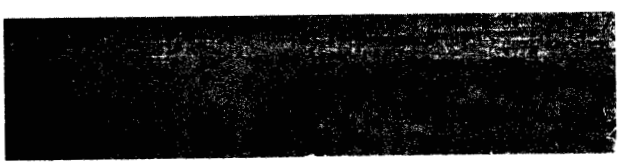
Reprinted from

IBM

Vol. 16 | No. 4 | July 1972

M. M. Guignard
K. Spielberg

Mixed-integer Algorithms for the (0,1) Knapsack Problem



M. M. Guignard
K. Spielberg

Mixed-integer Algorithms for the (0,1) Knapsack Problem

Abstract: An enumerative scheme is presented for the (0,1) knapsack problem as a specialization of the state enumeration method. Techniques are expiored for rendering search procedures more efficient by systematic use of information generated during execution of the algorithm. The inequalities of Benders and Gomory-Johnson are exploited to yield implicit enumeration tests in the special case of the knapsack problem. In a comparative study of eight algorithms and of the utility of certain approximations and inequalities, computational results are given for twelve knapsack problems, each having ten (0,1) variables. The effectiveness of these enumerative algorithms are thus tested in a relatively simple framework.

Introduction

The knapsack problem is concerned with filling some allocated space, e.g., a knapsack, of volume r with various items k (taken once only, or not at all) of volume b_k and having associated "profit" p_k selected from a set of p items, so as to maximize the overall profit:

$$\max \sum p_k t_k = z,$$
$$\sum b_k t_k + s = r \qquad s \geq 0, \quad t_k \in \{0, 1\},$$
$$k \in K = (1, 2, \cdots, p). \qquad (1)$$

The optimal solution will not, in general, completely fill the space r but will leave a "slack" portion s unfilled.

We shall assume that all b_k and p_k are positive and that the ratios p_k/b_k are in decreasing order from left to right:

$$p_k/b_k \geq p_{k+1}/b_{k+1}, \quad k = p-1, \quad p-2, \cdots, 1. \qquad (2)$$

The knapsack problem is an important zero-one integer or mixed-integer problem, both in its own right and as a possible aid in the solution of general mixed zero-one problems. We give in this paper an enumeration procedure which may be viewed as a specialization of the state enumeration method [1, 2]. However, the particular nature of the problem suggests a number of deviations and modifications of the enumerative scheme, e.g., the use of three states derived from easily obtained approximating solutions, and the incorporation of inequalities from the Gomory asymptotic problem.

Our primary objective is to demonstrate how well suited is the state enumeration technique, or a variant of it, for a problem of special structure. We believe that elements of the method could be incorporated advantageously in other algorithms for the knapsack problem (see, e.g., Refs. 3 and 4). We suggest also that other special problems which can be solved readily as linear programs and can be altered easily to give integer-feasible solutions, may be particularly amenable to solution by techniques such as those advocated here. The set covering problem and the plant location problem are two examples [5, 6].

A secondary objective is to explore methods for rendering search procedures more efficient by the systematic use of information generated during execution of the algorithm. In particular we have exploited to this end the inequalities of Benders and the asymptotic problem of Gomory. For the all-integer (0,1) knapsack problem, the Gomory-Johnson inequalities [7, 8], which are in general over all variables, including the slacks, can be transformed to inequalities over the zero-one variables only and can then be used in exactly the same manner as the Benders inequalities. Hence, we were able to test the effectiveness of these new ideas within a relatively simple framework.

We can thus use the zero-one inequalities to yield efficient implicit enumeration tests, leading to the fixing of variables or to the establishment of infeasibility [1, 2, 9].

For the mixed (0,1) knapsack problem and for general mixed (0,1) problems, one would have to use somewhat more complicated devices, e.g., linear programming over the inequalities. For such methods see Ref. 8. The actual computational results of this paper are for the pure (0,1) case.

This paper first points out some of the differences between branch-and-bound and enumerative programming and then shows how modifications of the state enumeration method can be applied to the solution of the given knapsack problem. Various forms of heuristic devices are next discussed, including the use of the linear programming solution, of three different integer solutions, and of Benders and Gomory-Johnson inequalities. Numerical results for these various algorithms are then compared with branch-and-bound techniques for the solution of twelve knapsack problems.

Branch-and-bound and enumeration procedures

There is a good deal of confusion in the literature as to the difference between (implicit) enumeration and branch-and-bound (BB) programming, especially "single branch" BB programming, in which one branch of the search tree is pursued to the end, with backtracking, before another is taken up. The reader may wish to consult Ref. 9 on such matters, e.g., on the terminology (a fairly obvious one) of single-branch and multibranch trees. Reference 9 should also be consulted for further references to the basic papers on BB and enumerative programming.

The essential difference lies in the nature of the problems that are actually solved: *relaxed problems* (i.e., problems in which the constraints $t_k \in \{0, 1\}$ are relaxed to $0 \leq t_k \leq 1$ in the case of BB programming) and *constrained problems*, in which specific integer values are substituted for all or a part of the t_k, in the case of enumeration.

This distinction can become somewhat blurred by the fact that an enumeration algorithm may also employ relaxed auxiliary problems intermittently. In the enumeration scheme of this paper, relaxed auxiliary problems are always used, even when all t_k are (0, 1) variables. In strict enumeration one could avoid all linear programs but usually only at the expense of losing valuable information.

In the state enumeration algorithm [1, 2], an auxiliary linear program may suggest (e.g., by rounding) a "state", which is a determination as to which of the "free" variables ($k \in F$) – the set of variables t_k which have not been explicitly fixed at 0 or 1 – are to be set to 1 ($k \in F1$), 0($k \in F2$), or be left free ($k \in F3$) in the solution of certain basic constrained "state problems."

The manner in which F is partitioned into $F1, F2, F3$, determines the order in which the enumeration proceeds and the inequalities that are generated. The arbitrariness available in the choice of the state furnishes the essential flexibility of the state enumeration method. Of course, it may not be easy to assign a state (to partition) intelligently. See Refs. 1 and 2 for details. The usefulness of a flexible procedure was, we believe, sufficiently demonstrated by the computational results of Ref. 5.

It ought to be emphasized that, in general, linear programming need not necessarily be used. Linear programming can be used intermittently only, e.g., only at the start. The state can be defined by means of some other method. Moreover, even the state problems need not be solved at all nodes, but only at the "terminal" nodes at which all zero-one variables have been fixed. In contrast to this, BB programming is intrinsically based on the resolution of two linear programs (or, more generally, two relaxed programs) at each node of the search tree.

For the knapsack problem the situation is quite special, in that any linear programming solution of (1), with some variables possibly fixed at 0($k \in Z$) and some at 1($k \in E$), can be made to have the property that no more than one component in the solution, say t_f, has a fractional value.

A BB algorithm, then, will always round variable t_f to 0 for one branch and to 1 for the other. It will then proceed from one of the pending nodes in the same fashion. The computational results in this paper will always refer to this true "multibranch" BB method with, in general, many pending nodes, as opposed to the single-branch approach.

The more interesting approach for us, however, is that of state enumeration, modified so as to take into account the special nature of the problem. For such an enumeration algorithm the inequalities become the motive force. Since the inequalities are almost solely responsible for the curtailment of the search process, the manner in which they are generated (e.g., by correct choices of state) becomes a factor of overriding importance.

State enumeration for the knapsack problem

Consider problem (1) as given initially, i.e., with all variables free:

$$F = K, \quad E = \varnothing, \quad Z = \varnothing. \tag{3}$$

All variables are free, i.e., none are fixed at 0 or 1. No distinction has yet been made among the free variables. In general, the state enumeration algorithm has the following raw structure:

1) Define l, the "level", to be (in a sense) the distance from the origin of the search, i.e., let it be the number of variables that have been fixed explicitly in the last sequence of "forward steps" from the origin (level 0). Initially, the level is set to 0. It is increased in forward steps, decreased in "backups".

2) If there are free variables left, go to 3). Otherwise, solve a terminal state problem, i.e., solve the original problem with the integer variables fixed as specified by E and Z. Go to 4).

3) *Forward step*
a) Determine a state, i.e., partition F into the three sets $F1, F2, F3$. In other words partition K:

$$K = E + Z + F1 + F2 + F3. \quad (4)$$

This state can be determined trivially, from the previous state, or by means of auxiliary computations. Pick a branch variable k^* in F.
b) Increase l by one.
Set $t_{k^*} = 1$ if k^* is in $F2$ or $F3$, $t_{k^*} = 0$ if k^* is in $F1$; i.e., adjoin k^* to E if it was in $F2$ or $F3$ and adjoin k^* to Z if it was in $F1$. Of course, modify $F1$, $F2$, $F3$ correspondingly. Store k^* for later use in backups. Go to 2).

4) *Backup*
Retrieve the value of k^* which led to the current node (level).
Complement the variable, i.e., if t_{k^*} is currently 1(0) fix it now at 0(1).
Reduce l by one. If $l = -1$, terminate. Otherwise, go to 2).

This description, of course, lacks motivation and all those details and additional features which are necessary to make the procedure effective, such as:

- In 3a) one must decide on a method of determining the state. Frequently, one may decide to solve a non-terminal-state problem, obtained from the original problem by substitution of $t_k = 1$ for k in $F1$ and $t_k = 0$ for k in $F2$, t_k being left to vary between 0 and 1 for k in $F3$. In the initial phases of the computation one would, of course, like to get a solution to the original problem. In general, however, computational evidence indicates that, wherever possible, it is desirable to choose the state close to a hypothesized good solution [5].
- When a state problem or an auxiliary problem has been solved, one can always obtain a Benders inequality [9, 10] over the integer variables. From the auxiliary problem one usually is able to obtain Gomory-Johnson inequalities [7, 8]. All of those are necessary conditions for obtaining integer solutions. The Benders inequalities may involve z^* (a given or already computed bound on the objective function) in which case they are conditions for obtaining integer solutions with objective functions better than, or equal to, z^*.

  These inequalities can be utilized systematically in a search for the possibility of fixing ("cancelling") variables at the current level l. On a backup to a previous level, all cancellations at the current level must be rescinded.
- In the general case, the inequalities may afford the best way of choosing k^*. One will choose k^* such that the forward step alters the state so that the new state problem at the successor state differs from the previous one.

For details, the reader is referred to Ref. 2 or 8, although we realize that those descriptions may be too concise to be sufficiently clear. It is in fact difficult to define any general rules for selecting effective computational features. It is up to the analyst to decide on such matters in relationship to the particular problem at hand. In this paper we have tried to do this to some extent for the knapsack problem.

Approaches to solving the knapsack problem

The linear programming solution for (1) is obtained rather easily. We therefore made the natural decision to solve it, as an auxiliary problem, in every iteration. From the linear programming solution we then construct heuristically three different solutions, if possible, to the original problem. With each such integer solution we may, if we wish, associate a state and also a corresponding linear programming solution and tableau. We shall give details presently, including examples.

From each linear programming (LP) tableau we may, by inspection, deduce a linear inequality after Benders. (The Benders inequalities derived from the integer solutions are only integral multiples of the original constraints and are of no interest here). Also, we can in general derive an asymptotic problem after Gomory and construct a Gomory-Johnson inequality over the t_k and s. Computationally this will generally require the construction of a row in an updated LP tableau and a table look-up with interpolation.

It so happens that s can be eliminated between (1) and the Gomory-Johnson inequality, so that one again arrives at a final inequality involving the t_k alone.

There are, then, actually three states at level l. Each yields, from the associated linear programming tableau, one or more inequalities. The branch variable k^*, then, could be chosen with reference to any of these three different states and is in fact chosen from the first state.

In our computational work we have, because of time limitations, refrained from some of these options. For example, we construct Gomory-Johnson inequalities only from the first LP tableau at level l, i.e., from the tableau determining the first state. See the next two sections for a description of the three LP solutions. The associated LP tableaux can be visualized easily.

We choose the branch variable, trivially, as the first fractional variable, characterized by the index f. This

makes the algorithm somewhat similar to a single-branch BB algorithm. But the other options do exist and may very well be worth exploring further.

• *Linear programming solution*

At a particular node of the search, let the state be given by (E, Z, F). For the time being, F is not yet partitioned further, nor does the further partitioning play a determining role as in the general case of state enumeration. We may assume that the problem is feasible and nonintegral. (If the program is not feasible a backup in the search is indicated.) The solution is then uniquely determined by the index $f \in F$ of the one variable that becomes fractional.

As a matter of fact, F is partitioned into the sets L (consisting of all indices in F that are to the left of f), $\{f\}$, and R (all indices of F to the right of f):

$$F = L + \{f\} + R. \tag{5}$$

In the LP solution, the variables t_k, $k \in L$, have value one and those with $k \in R$ have value zero.

Example (Note that the p_k/b_k are ordered):

$$p = (15, 27, 10, 15, 18, 10, 41, 32, 62, 70),$$

$$b = (11, 25, 10, 15, 20, 12, 50, 40, 80, 100), \qquad r = 74$$

$$E = Z = \varnothing, \quad F = K.$$

$$\text{LP solution}: f = 5, \quad L = \{1, 2, 3, 4\}, \quad R = \{6, 7, 8, 9, 10\},$$

$$t = (1, 1, 1, 1, 13/20, 0, 0, 0, 0, 0).$$

• *Three integer solution approximations*

Not all solutions need exist, in general. The solution of i) may be good, and ii) and iii) may occasionally not give an improvement. But usually the simple heuristics of ii) and iii) will be found effective.

i) A trivial solution is obtained by rounding down t_f.
ii) Round t_f to zero. Then scan right and try to refill the knapsack at unit levels from variables in R.
iii) Round t_f to one. This renders the solution infeasible.
   1. Set $j = f$, $J = L$.
   2. If J is void, abandon the search without solution. Otherwise, scan left (from j) and set, consecutively, all t_k, $k \in J$, one at a time, to zero. If there are feasible solutions, take the best one and stop.
   3. If there are no feasible solutions, set t_k to 0 for the rightmost index in J. Drop this rightmost index from J and set j to the new rightmost index of J (if possible). Go to 2.

With each of the three integer solution approximations one can associate a linear programming solution by giving the first variable from the left in F a fractional value, which ensures that the knapsack is full.

Example: In the case of the example given above, one has three LP solutions, z_λ being the objective function for an LP solution and z that for an integer solution):

$$t^1 = (1, 1, 1, 1, 13/20, 0, 0, 0, 0, 0), \qquad f^1 = 5, \quad z^1{}_\lambda = 78.7,$$

$$t^2 = (1, 1, 1, 1, 0, 1, 1/50, 0, 0, 0), \qquad f^2 = 7, \quad z^2{}_\lambda = 77.82,$$

$$t^3 = (1, 1, 3/10, 1, 1, 0, 0, 0, 0, 0), \qquad f^3 = 3, \quad z^3{}_\lambda = 78.$$

The three integer approximations are the same with $t_5 = 0$ for t^1, $t_7 = 0$ for t^2, and $t_3 = 0$ for t^3. The integer objective functions are $z^1 = 67$, $z^2 = 77$, $z^3 = 75$. The second solution happens to be optimal.

It would probably not be very difficult to devise somewhat more elaborate heuristic rules.

• *Benders inequalities*

Instead of giving a general discussion of Benders inequalities and their use in enumerative programming we refer the reader to Refs. 9 or 10. It should suffice to note that, for any optimal LP tableau, one may look at the top row which expresses z as a linear combination of the nonbasic variables $(k \in N)$. Among these variables are, in general, some integer variables y_k,

$$z = z_\lambda - \sum_{N \cap K} \tilde{h}_k \tilde{y}_k - \cdots. \tag{6}$$

The summation is over the nonbasic variables only. The $\tilde{h}_k$ are essentially the reduced costs. At optimality, all $\tilde{h}_k$ are nonnegative for $k \in F$, but not necessarily for $k \in E$ or $k \in Z$. The $\tilde{y}_k$ stand for $y_k (k \in R + Z)$ and for $\tilde{y}_k = 1 - y_k (k \in L + E)$. If z^* is the best-known value for an integer solution, then an improvement can be found only for values of the $\tilde{y}_k$ such that

$$z_\lambda - \sum \tilde{h}_k \tilde{y}_k > z^*. \tag{7}$$

Therefore, rewriting (7) in terms of the original variables y_k, or rather t_k for the knapsack problem, one has a linear inequality over the t_k. Fortunately, the knapsack problem affords the rare opportunity of writing the inequality in the closed form:

$$\sum_{N \cap K} h_k t_k < z_\lambda - z^* + \sum_{(L+E) \cap N} h_k, \tag{8}$$

$$h_k = (-p_k + b_k p_f / b_f).$$

The summation is over all nonbasic variables that are either fixed at one, or are at one in the first LP solution, i.e., are to the left of the index F that determines the first LP solution.

In general, one obtains such an inequality for each of the three linear programming tableaus. For the knapsack problem one always has a value for z^*. In a certain sense, L and R and $\{f\}$ play the role of $F1$, $F2$, and $F3$ of the general state enumeration algorithm.

• *Gomory-Johnson inequalities*

The knapsack problem is also a good vehicle for testing some of the important new ideas of Gomory and Johnson [7], to whom we are grateful for making this material available before its general dissemination. We also wish to acknowledge the generous advice of Ellis Johnson, given freely in many discussions. We shall give only a brief description and suggest Ref. 8 for a rather detailed exposition of the basic ideas.

Starting with a linear programming solution with fractional index f, and corresponding index sets $E, Z, L, R, \{f\}$, one may write the well-known group problem, the asymptotic problem of Gomory, as follows:

Let $K1 = (L+E) \cap N$, $K2 = (R+Z) \cap N$. Then

$$\min -\sum_{K1} h_k t_k + \sum_{K2} h_k t_k - (c_f/b_f)s,$$

$$-\left\{\sum_{K1} b_k \bar{t}_k + \sum_{K2} b_k t_k\right\} \Big/ b_f + s/b_f = v \pmod 1, \qquad (9)$$

$$t_k \in \{0, 1\}, \quad k \in (L+R) \cap N, \qquad s \geq 0,$$

in which the complementary variables $\bar{t}_k = 1 - t_k$ have been introduced for $k \in L+E$, and v is a correspondingly modified right-hand side of the constraint. We have chosen to treat the original problem (1) as a minimization problem, $\min c_k t$, where c_k is identified with $-p_k$.

Gomory and Johnson show in Ref. 7 how one may obtain valid and relatively strong inequalities of the form

$$\sum_{K1} \pi_k \bar{t}_k + \sum_{K2} \pi_k t_k + \pi^{\pm} s \geq 1, \qquad (10)$$

by taking the faces of tabulated corner polyhedra for cyclic groups, (see Gomory [11]) and by "interpolating" in various ways (see [8] for details). The numerical results below were obtained for the simplest possible case involving the two faces of a cyclic group of order two. One may well expect that the utilization of higher-order groups, not in principle very much more complicated, could lead to substantially superior results.

After the π_k and $\pi^{\pm}$ have been obtained, one may in the case of the knapsack problem eliminate the slack variable s between (1) and (10) to arrive at an inequality over the t only. We also use the obvious inequality, the ceiling test,

$$\sum_{K} p_k t_k > z^*, \qquad (11)$$

which is of the same form as the Benders inequalities.

Numerical results for eight methods

• *Algorithms*

In order to study the comparative merit of algorithms and the utility of approximations and inequalities, we implemented eight algorithms within a modular system of subroutines:

1) A branch-and-bound algorithm of multibranch type, i.e., having as many pending nodes as are generated, stored, and retrieved according to largest objective function value. The integer solution approximation i) is used. No "penalty" computation is made of the type customary in branch-bound codes.
2) The same as in 1) plus "group penalties", computed from a cyclic group of order seven at the suggestion of Ellis Johnson. Customary penalties could be expected to be weaker. This relates to only one pivot step in the rounding up or rounding down of a basic variable, whereas the group penalties give a good measure of the total cost of rendering that basic variable integral.
3) The state enumeration algorithm with approximation i) and corresponding Benders inequalities.
4) State enumeration with approximation i) and Gomory-Johnson inequalities (computed from the cyclic group of order two) as described earlier.
5) State enumeration with approximation i) and Benders as well as Gomory-Johnson inequalities.
6), 7), 8) Same as 3), 4), 5) except that all approximations i), ii), and iii) are used.

• *Numerical results*

In Table 1 we give, for each method, results for 12 knapsack problems of small size (10 zero-one variables only). A number of 75- and 100-variable problems were also run and did not cause any particular difficulties. The 12 problems consist of three different sets of constraint data, each of which was run with the four right-hand sides 74, 80, 139 and 250.

When there is one entry in a table column location, it denotes the number of linear programs that were executed during the solution process. A second entry situated just below the first gives the number of inequalities generated and utilized. Where two numbers are located below the first entry, they give the number of distinct Benders inequalities and the number of Gomory-Johnson inequalities, respectively.

Rows 1 and 2 in the table are a measure of the basic effectiveness of BB programming, primarily for comparison purposes. Results in rows 3, 4 and 5 are representative of state enumeration. We believe that this comparison is fair. Methods 3 to 5 use inequalities, whereas 1 and 2 do not. However, the use of inequalities is an essential

Table 1 Twelve knapsack problems, resolved by methods 1 to 8 (See text for explanation and interpretation of column entries).

Problem	*1*	*2*	*3*	*4*	*5*	*6*	*7*	*8*	*9*	*10*	*11*	*12*
Objective function	*77*	*80*	*127*	*215*	*79.654*	*84.015*	*147.42*	*260.97*	*73.55*	*73.55*	*145.65*	*219.20*
METHOD												
1. Branch-and-bound	26	40	54	30	16	32	30	96	18	18	18	16
2. Branch-and-bound + group penalties	6	14	18	10	14	32	30	78	16	18	14	16
3. State enumeration, (i)	11	27	34	14	10	11	12	62	15	24	8	10
Benders	3	5	5	5	3	3	2	6	2	2	2	2
4. State enumeration, (i)	13	16	20	16	7	6	12	26	2	2	1	2
Gomory-Johnson	20	22	26	22	7	9	13	40	3	3	3	3
5. State enumeration, (i)	10	14	11	8	5	4	7	21	2	2	1	2
Benders, Gomory-Johnson	5,16	5,19	5,18	5,13	4,7	4,6	3,8	5,35	2,2	2,2	2,2	2,2
6. State enumeration, (i) (ii) (iii)	2	8	5	3	2	10	7	43	6	13	8	18
Benders	4	4	5	3	4	4	4	11	3	3	2	3
7. State enumeration, (i) (ii) (iii)	7	9	9	4	4	4	8	23	1	1	1	1
Gomory-Johnson	8	11	11	7	5	5	7	37	3	3	3	3
8. State enumeration, (i) (ii) (iii)	2	6	4	2	1	3	3	16	1	1	1	1
Benders, Gomory-Johnson	4,3	5,10	5,6	3,4	4,2	6,6	4,6	13,32	3,2	3,2	2,2	3,2

feature of state enumeration, just as the use of a tree with pending nodes stored in a table and the selection of a best pending node for further processing is characteristic of branch-and-bound programming.

The relative expenditure of computational effort is difficult to assess, depending on the weight that one wishes to give to storage requirement versus purely computational requirement. Row 2 represents a rather good BB scheme. The results of Ref. 8 seem to suggest that inequalities, especially of the Benders type, are only modestly useful in BB programming if used as they are in state enumeration. (Used differently, in auxiliary linear programs over a set of inequalities, they can be quite effective). As a consequence we believe that the results of the table are not biased and that the good performance exhibited in row 5 is indicative of a considerable potential for state enumeration.

The rows 6 to 8 represent what can be done when one exploits the specific structure of a problem. In the present case, this exploitation is effected by construction of approximating integer solutions. The effect is drastic, even though no particular ingenuity was expended on this. Better approximations can undoubtedly be found at modest additional computational cost. The lesson is (and this is reinforced by similar experiences with special structures in other areas) that it is essential to take problem structure into account as much as is at all possible. Examples of these experiences may be found in studies of set covering and plant location [9], where additional references may be found.

Problem structure can, of course, be accounted for in many ways, and much expertise can be brought to bear upon it. Within the enumerative framework presented here, very simple devices were employed, and effectively so, as the last entries of the table indicate. In our opinion, the state enumeration approach with inequalities, once established in a program, makes the exploitation of special structures easier than might otherwise be the case.

References

1. M. M. Guignard and K. Spielberg, "Search Techniques with Adaptive Features for Certain Integer and Mixed Integer Programming Problems," *Proc. IFIPs Congress*, 1968.
2. M. M. Guignard and K. Spielberg, "The State Enumeration Method for Mixed Zero-One Programming," IBM Technical Report No. 320-3000, Philadelphia Scientific Center, Feb. 1971.

3. P. C. Gilmore, and R. E. Gomory, "The Theory and Computation of Knapsack Functions," *Operations Research* **14,** 1045 (1966).
4. D. Fayard and G. Plateau, "Une Méthode Enumerative pour les Problemes de Knapsack à Variables Bivalentes," Publication 30, Lab. de Calcul, Univ. de France in Lille, March 1971.
5. K. Spielberg, "Plant Location with Generalized Search Origin," *Management Science,* **16,** 165 (1969).
6. C. E. Lemke, H. M. Salkin, K. Spielberg, "Set Covering by Single-Branch Enumeration with Linear Programming Subproblems," *Operations Research* **19,** 998 (1971).
7. R. E. Gomory and E. L. Johnson, "Some Continuous Functions Related to Corner Polyhedra," IBM Research Report RC3311, Feb. 23, 1971.
8. E. L. Johnson and K. Spielberg, "Inequalities in Branch and Bound Programming," NATO conference, Helsingor, 1971. See IBM Research Report RC-3649, December 1971.
9. M. L. Balinski and K. Spielberg, "Methods for Integer Programming: Algebraic, Combinatorial and Enumerative," *Progress in Operations Research,* Vol. 3, Editor, J. Aronofsky, John Wiley & Sons, Inc., New York 1969.
10. J. F. Benders, "Partitioning Procedures for Solving Mixed-Variables Programming Problems," *Numerische Mathematik* **4,** 238 (1962).
11. R. E. Gomory, "Some Polyhedra Related to Combinatorial Problems," *Linear Algebra and its Applications* **2,** 451 (1969).

Received November 22, 1971

K. Spielberg is located at the IBM Data Processing Division Scientific Center, 3401 Market Street, Philadelphia, Pennsylvania 19104. M. M. Guignard is Maitre Assistante, Computing Laboratory, Department of Computer Science, Electronics, Electrotechnics and Automata, University of Lille. Dr. Guignard also served as consultant to the IBM Data Processing Division Scientific Center.

IBM PHILADELPHIA SCIENTIFIC CENTER TECHNICAL REPORT NO. 320-3025 JUNE 1973

A REALIZATION OF THE STATE ENUMERATION PROCEDURE

M. M. GUIGNARD
University of Pennsylvania, Philadelphia

K. SPIELBERG
IBM Scientific Center, Philadelphia

PHILADELPHIA SCIENTIFIC CENTER
IBM CORP.
3401 MARKET STREET, PHILADELPHIA, PENNSYLVANIA, 19104

Abstract

"State Enumeration" is a natural way of extending enumeration procedures for pure zero-one programming problems to the zero-one mixed integer problem. The main emphasis of this paper is to show what can be accomplished with a particularized version of it, using fully some of the techniques which have recently become available within integer programming. Numerical results for a number of standard test problems are given.

A Realization of the State Enumeration Procedure.

In earlier papers, such as <1>, we have outlined a "State Enumeration Method" as the natural way of extending enumeration procedures for pure zero-one programming problems to the zero-one mixed integer problem (see <2> and <3> for surveys). But the method is complex, allows many options, and is difficult both to describe and to implement. We still feel that it is the appropriate generalization of enumeration to mixed integer programming, but the main emphasis of this paper is to show what can be accomplished with a particularized version of it, using fully some of the techniques which have recently become available within integer programming.

The Problem

Consider the mixed integer programming problem (P):

$$\begin{aligned} \text{Min} \quad & z = f \cdot x + g \cdot y \\ & A \cdot x + B \cdot y = C \\ & x \geq 0, \; x \in R^p \\ & y \in S \subset R^n \end{aligned}$$

where A and B are (m,p) and (m,n) matrices, respectively, and where $S = \{0, 1\}$.

The General Algorithm

We solve (P) enumeratively. Initially, at level 0 of the search tree used to describe the enumeration of all possible values of y in S, all variables are "free". They are fixed along branches of the tree, which is visualized as growing downward, and altered to the complementary value on "backup" steps. An essential concept is that of the state associated with a node <1> . It is a partitioning of the set of the remaining free variables into three subsets: F0 of variables temporarily set at zero, F1 of variables temporarily set at 1, and F-(F0 ∪ F1). The state is essentially used as an attempt to find feasible solutions and to curtail the search as much as possible via generated Benders inequalities, <4>. Problem (P) at a given node (with the fixed variables substituted) corresponds to $F0=F1=\phi$, but one may derive useful information from nontrivial choices of F0 and F1.

The Auxiliary Problems.

At a given node of the tree, relaxation of the integrality constraints on the remaining free variables results in a "relaxed" auxiliary problem whose optimal value, if it exists, is a lower bound for the value of z over all successor nodes.

Constraining the relaxed problem by choosing F0 and F1 not both empty and letting the other bivalent variables vary between 0 and 1 gives the "state problem". Especially if F0 and F1 cover the whole set of still free variables, this problem may at some point in the search give a feasible solution of (P).

Use of Inequalities

From state or auxiliary problems, whether they are feasible or not, one may derive inequalities which are necessary conditions for a y to yield an improvement over previously found feasible solutions, if any. These conditions are valid over the whole tree, and may be used either locally, i.e. over successor nodes, or globally. They are equivalent to Benders conditions. One can use reduction over the set of generated inequalities of this nature to fix variables, since they are in terms of y only <5,6>. Also, one may add cuts to an optimal LP tableau in order to remove the integral infeasibility of basic bivalent variables and reoptimize over the augmented tableau. In particular one may use the minimal valid inequalities of Gomory and Johnson <7,8,9>.

Use of Bounds

The lower bound z_* furnished by relaxed problems and the upper bound z^* furnished by previously found feasible solutions of (P) give an interval on which the further search should be concentrated. In case of a purely bivalent problem, the two inequalities

$$g\ y \leq z^*$$

and $$g\ y \geq z_*$$

can be adjoined to the Benders inequalities and used for reduction. In all cases, the resulting inequalities have the form

$$\beta + z \geq \sum_j \alpha_j y_j = \alpha \cdot y \quad .$$

This implies that any value of z has to satisfy

$$z \geq -\beta + \alpha y \quad .$$

The weakest way in which this inequality can be satisfied is that of setting to 1 (0) the components of y with negative (non negative) coefficients α_j so that a lower bound for z will be

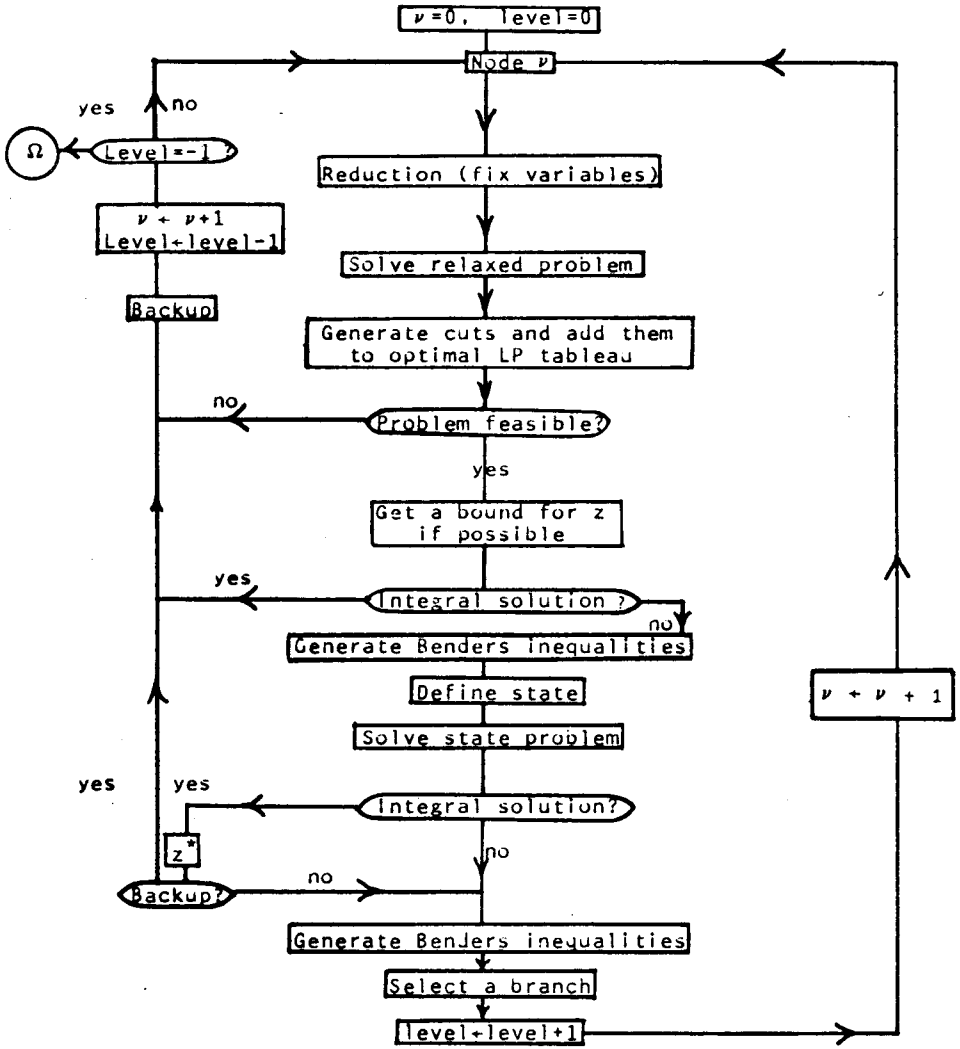
$$-\beta + \sum_{j:\alpha_j<0} \alpha_j + \sum_{j\in E} \alpha_j$$

where E is the set of variables definitively set at 1.

Choice of a Branch

If after having generated the Gomory-Johnson cuts for a relaxed LP, one reoptimizes over the augmented tableau, the optimal solution will most likely not be integral yet. One can choose to reoptimize several times, and eventually branch on the variable which turns out to be the most difficult to render integral in that fashion, namely the one which most persistently has a non-integral value during the reoptimizations. In ties, one may choose the branch variable from a preferred set of small size, so that setting it at its preferred value now, or on the way back, will allow the fixing of other variables at some point (see <6>).

Fig. 1 Flowchart for State Enumeration



Numerical Results

We solved several test problems taken from the literature ⟨10⟩, ⟨11⟩, ⟨12⟩, ⟨13⟩. The program was written in APL ⟨14⟩. We are indebted to Ellis Johnson for having made available to us an APL function which generates valid Gomory-Johnson inequalities, ⟨7, 8⟩. The program also uses APL functions which generate minimal preferred inequalities; these are described in ⟨6⟩. The problems were run both with and without reduction, ⟨6⟩. The following table compares the results which we obtained with similar results from a normal branch and bound technique or adaptive binary programming ⟨15⟩, when available. The combined use of Gomory-Johnson inequalities and reduction, together with branching on repeatedly fractional basic variables, and lower bounds obtained from Benders inequalities, seem promising.

	Normal B B	Adaptive binary programming DZLP <15>	State Enumeration without reduction	with <6>
6x12 <10>	15,55	8	5,66,1	2,36,2
10x20 <12>	41,542	76	5,116,4	5,83,4
10x28 <12>	---	85	33,435,31	21,295,14
15x30 <11>	---	---	22,503,22	4,91,4
31x31 <11>	---	73	7,406,7	6,404,6
5x39 <12>	---	208	25,649,7	35,811,33
5x50 <12>	---	347	---	106,2329,101
28x35 <5>	57,1246	69	10,359	10,175,1
12x44 <5>	45,399	72	5,132,3	7,137,5
1x40 <13> 1.	---	---	5,138,4	4,71,4
2.	---	---	1,24,1	1,23,1

For each entry, the numbers represent: No. of LP's, No. of pivots, Node at which opt. sol. found (in that sequence).

Example

In the (6x12) problem of Bouvier and Messoumian, one can immediately fix y1 at 0. The minimal number of preferred variables generated by reduction is then 2. With y1 =0, the relaxed problem ($0 \leq y \leq 1$) has an optimal objective function value of 6.85, 5 integer infeasible rows in the optimal tableau give 5 cuts, reoptimization lifts the optimum to 8.94, and a second reoptimization with new cuts added gives a lower bound for z of 9.72. The state problem obtained by rounding fractional components of y is infeasible and yields a Benders inequality which is nothing but one of the isolated constraints. Three variables y6, y8 and y10 appeared repeatedly in the basis in that process, y6 and y8 moreover appear twice in preferred inequalities. One chooses to branch on y6, rounding it down from .09 to 0. Node 2 at level 1 with two variables fixed, y1 definitively at 0 and y6 locally at 0, sees first of all three other variables fixed, y11 and y5 at 0, and y3 at 1. The relaxed objective function value, through the addition of cuts is successively 8.1, 11.6 and 11.8. Four variables are basic at the end, but only one, y12, is fractional. One generates a Benders inequality and the state problem obtained by rounding y12 up from .76 to 1 is feasible, with an integer solution and an objective function value of 13. One can fix

variables globally from the inequalities, y5, y7, y9, y11 at 0, y3 and y4 at 1. Locally the problem becomes infeasible (this result is obtained through reduction). Backup leads to level 0 at which 8 variables are fixed (among them y6 complemented at 1). This yields an empty constraint set, which is checked easily by reduction. Altogether, two LP's have been necessary, including postoptimization and an intensive use of reduction has considerably curtailed the search.

Sample Problem:

PROBLEM N. 6 12

MIN (5 1 3 2 6 4 7 2 4 1 1 5) × *Y*

SUBJECT TO

($^{-}$1 3 $^{-}$12 0 $^{-}$1 7 $^{-}$1 0 0 3 $^{-}$5 $^{-}$1) × *Y* ≤ $^{-}$6

(3 $^{-}$7 0 1 6 0 0 0 0 0 0 0) × *Y* ≤ 1

(11 0 1 $^{-}$7 0 $^{-}$1 2 1 $^{-}$5 0 9 0) × *Y* ≤ $^{-}$4

(0 5 6 0 12 7 0 3 1 $^{-}$8 0 5) × *Y* ≤ 8

($^{-}$7 $^{-}$1 $^{-}$5 3 1 $^{-}$8 0 $^{-}$2 7 1 0 $^{-}$7) × *Y* ≤ $^{-}$7

($^{-}$2 0 0 $^{-}$4 0 0 $^{-}$3 $^{-}$5 $^{-}$1 0 1 1) × *Y* ≤ $^{-}$4

AND Y[I]= 0 *OR* 1 , *I*=1,...,12

References

1. GUIGNARD, M., SPIELBERG, K., Search techniques with adaptive features for certain integer and mixed-integer programming problems. Proc. IFIPS Congress, Edinburgh, 1968, North-Holland Publ. Co., pp. 238-266.

2. GEOFFRION, A., MARSTEN, R. E., Integer Programming Algorithms: A framework and state-of-the-art survey. Perspectives on optimization, Ed.: A. Geoffrion, Addison Wesley Pub. Co. 1972, pp. 137 - 163.

3. BALINSKI, M.L., SPIELBERG, K., Methods for Integer Programming: Algebraic, Combinatorial and Enumerative. Progress in Operations Research, Ed. J. Aronofsky, John Wiley & Sons, 1969, pp. 195 - 292.

4. BENDERS, J.F., Partitioning procedures for solving mixed variables programming problems., Num. Math. Vol. 4, 1962, pp. 238 - 252.

5. LEMKE, C. E., SPIELBERG, K., Direct Search Algorithms for Zero-one and Mixed-integer Programming. Journal of ORSA, Vol. 15, No. 5, Sept. 1967, pp. 892 - 914.

6. SPIELBERG, K., Minimal preferred variable reduction for zero-one programming. Presented at the ORSA meeting, New Orleans, 1972, IBM Philadelphia Scientific Center Report No. 320 3013, July 1972.

7. GOMORY, R.E., JOHNSON, E. L., Some continuous functions related to corner polyhedra, Mathematical Programming 3, August 1972, pp. 23 - 86.

8. JOHNSON, E.L., Cyclic Groups, Cutting Planes, and Shortest Paths, IBM Research Report RC4010, Aug. 1972.

9. JOHNSON, E.L., SPIELBERG, K., "Inequalities in Branch and Bound Programming", IBM Research Report RC 3649, December 1971. To appear in Proceedings, NATO Conference, Copenhagen 1971.

10. BOUVIER, B., MESSOUMIAN, G., Programmes linéaires en variables bivalentes. Faculté des Sciences, University of France at Grenoble: Doctoral Thesis, 1965.

11. HALDI, J., 25 Integer Programming Test Problems, Stanford University, Working Paper no. 43, Dec. 1964.

12. PETERSEN, C., Computational Experience with Variants of the Balas Algorithm Applied to R+D Projects. Mngmt Sc., Vol. 13., No. 9, May 1967, pp. 736 - 750.

13. PLATEAU, G., FAYARD,D., Une methode enumerative pour les problemes de Knapsack a variables bivalentes. Publication No. 30, Laboratoire de Calcul, University of France at Lille, March 1971.

14. FALKOFF, A.D., IVERSON, K.E., APL/360 User's Manual, IBM, 1968, currently available as IBM publication GH 20 - 0683, 1970.

15. SALKIN, H. M., On the Merit of the Generalized Origin and Restarts in Implicit Enumeration., Opns. Res. 18, 1970, pp. 549 - 555.

TECHNICAL REPORT INDEXING INFORMATION

1. AUTHOR(S): M. M. Guignard and K. Spielberg	9. INDEX TERMS FOR THE IBM SUBJECT INDEX: State Enumeration Mixed Integer Programming 13 - Management Science 16 - Mathematics
2. TITLE: A Realization of the Stae Enumeration Procedure	
3. ORIGINATING DEPARTMENT: Philadelphia Scientific Center	
4. REPORT NUMBER: 320-3025	
5a. NO. OF PAGES: 13	5b. NO. OF REFERENCES: 15
6a. DATE COMPLETED: April 1973	6b. DATE OF INITIAL PRINTING: June 1973
6c. DATE OF LAST PRINTING:	

7. ABSTRACT:

"State Enumeration" is a natural way of extending enumeration procedures for pure zero-one programming problems to the zero-one mixed integer problem. The main emphasis of this paper is to show what can be accomplished with a particularized version of it, using fully some of the techniques which have recently become available within integer programming. Numerical results for a number of standard test problems are given.

8. REMARKS:

During an all-integer zero-one algorithm, at a general iteration, one has a matrix relation of the form (τ denoting transposition):

$$(z,s,y,\bar{y})^{\tau} = A \cdot (1,-t)^{\tau} , \qquad (2.6)$$

with

$$A = \begin{pmatrix} v & -c \\ b & C \\ \gamma & Q \\ \bar{\gamma} & -Q \end{pmatrix} \qquad (2.7)$$

The vector t has as elements the set of "current" independent variables, consisting of some composite of the structurals $y(j)$ and certain slack-variables $r(k)$ which are linear combinations of the original $y(j)$. A representation of the $r(k)$ in terms of the original $y(j)$ can be carried along easily:

$$r = \rho + R \cdot y . \qquad (2.8)$$

Gomory's well-known all-integer integer algorithm maintains dual feasibility ($c(j) \geq 0$) and integrality of the "logical part" of A (the relations involving Q) . With suitable precautions in the selection of pivot rows and columns it has finite, albeit usually poor, convergence. Until optimality, the "current solution" (determined by $t=0$) has some negative components. As explained in <1>, one may derive from each corresponding row an implied pivot row

$$r(i) = -h(i) + (-g(i)) \cdot (-t) \qquad (2.9)$$

with $h(i) > 0$ and $g(i,j) \geq 0$, determine the pivot column j to be the lexicographically smallest column with $g(i,j) > 0$, pivot $r(i)$ into the non-basis and discard the updated pivot row. ($r(i)$, $h(i)$ are scalars, $g(i)$ is an n-vector with entries $g(i,j)$). In the case of a zero-one program it is natural to take the parameter λ of Gomory, which determines the values of the pivot row coefficients, large enough to render $h(i)$ one and the $g(i,j)$ zero or one. The pivot step then becomes particularly simple, consisting in subtraction of the pivot column from all other columns with a minus one entry in the pivot row.

To obtain a stronger algorithm we suggest the use of the reduction procedures of <2>, as follows:

(a) Drop the objective function row and all feasible rows (those with nonnegative constant term).

(b) Drop all positive terms in the resulting matrix and multiply each column with the range (bound interval) of the corresponding variable (initially 1).

(c) Reduce the resulting matrix as in <2>, to obtain a set of minimal preferred inequalities.

These inequalities have the form (2.9) with all coefficients $(-h(i),-g(i,j))$ equal to 0 or -1, and it is clear that for any Gomory inequality as described above there exists a minimal preferred inequality which is either equal to it or better, in the sense of having fewer coefficients.

Reprinted from:
NONLINEAR PROGRAMMING, 2
© 1975
ACADEMIC PRESS, INC.
New York San Francisco London

ZERO-ONE ZERO-ONE PROGRAMMING AND ENUMERATION

by
Monique Guignard[1]
Kurt Spielberg[2]

1. Introduction

It is possible to devise a special version of the Gomory all-integer integer algorithm so that: the introduced cuts are especially tight, the matrix of added cuts can be carried in logical form (has entries -1, +1, 0) and the current solution is on a vertex of the hypercube. The algorithm can always start and is likely to fail after a number of iterations. The circumstances under which failure occurs can be characterized to some extent, and one can use heuristic and linear programming devices to delay failure.

Aside from raising questions of intrinsic interest, the algorithm may have some use in branch and bound and enumerative programming. When failure occurs, the existing tableau can be taken as starting point for resolution of a linear program, and the resultant overall procedure qualifies for providing bounds in a branch and bound search. In this paper, however, we explore an enumerative procedure in which the final tableau (after failure) is taken to define a set of auxiliary cuts to be used in the enumeration.

1) Dept. of Statistics and O.R., Univ. of Pennsylvania
2) Philadelphia Scientific Center, IBM

2 All-integer zero-one programming.

Consider the zero-one program:

$$\min \; c^o \cdot y$$
$$c^o \cdot y \le b^o \qquad (2.1)$$
$$y(j) \text{ in } \{0,1\},\ j = 1,2,\ldots,n\ ,$$

in which C^o is a matrix of m rows and n columns, and b^o and c^o are correspondingly dimensioned vectors. The coefficients need not necessarily be integral, but computer tests of the proposed algorithms have been performed on integer data only.

Let (2.1) be rewritten in the form

$$\min z = 0 + (-c^o)\cdot(-y) \qquad (2.2)$$
$$s = b^o + C^o\cdot(-y)$$
$$\text{and } y = 0 + (-1)\cdot(-y) \qquad (2.3)$$
$$\bar{y} = e + I\cdot(-y),$$

as suggested by R. E. Gomory in <1> (except for the last, redundant, constraint set).

I is an (n,n) identity matrix and e a vector of n entries 1. $\bar{y}$ is the vector of complemented structural variables:

$$\bar{y}(j) = 1 - y(j) \qquad (2.4)$$

(2.2) and (2.3) link the objective function z and the "dependent variables" $(s,y,\bar{y})$ with the "independent variables" $(-y)$ in terms of the matrix

$$A^o = \begin{pmatrix} 0 & -c^o \\ b^o & C^o \\ 0 & -I \\ e & I \end{pmatrix} \qquad (2.5)$$

During an all-integer zero-one algorithm, at a general iteration, one has a matrix relation of the form (τ denoting transposition):

$$(z,s,y,\bar{y})^{\tau} = A \cdot (1,-t)^{\tau} , \qquad (2.6)$$

with

$$A = \begin{pmatrix} v & -c \\ b & C \\ \gamma & Q \\ \bar{\gamma} & -Q \end{pmatrix} \qquad (2.7)$$

The vector t has as elements the set of "current" independent variables, consisting of some composite of the structurals $y(j)$ and certain slack-variables $r(k)$ which are linear combinations of the original $y(j)$. A representation of the $r(k)$ in terms of the original $y(j)$ can be carried along easily:

$$r = \rho + R \cdot y . \qquad (2.8)$$

Gomory's well-known all-integer integer algorithm maintains dual feasibility ($c(j) \geq 0$) and integrality of the "logical part" of A (the relations involving Q) . With suitable precautions in the selection of pivot rows and columns it has finite, albeit usually poor, convergence. Until optimality, the "current solution" (determined by $t=0$) has some negative components. As explained in <1>, one may derive from each corresponding row an implied pivot row

$$r(i) = -h(i) + (-g(i)) \cdot (-t) \qquad (2.9)$$

with $h(i) > 0$ and $g(i,j) \geq 0$, determine the pivot column j to be the lexicographically smallest column with $g(i,j) > 0$, pivot $r(i)$ into the non-basis and discard the updated pivot row. ($r(i)$, $h(i)$ are scalars, $g(i)$ is an n-vector with entries $g(i,j)$). In the case of a zero-one program it is natural to take the parameter λ of Gomory, which determines the values of the pivot row coefficients, large enough to render $h(i)$ one and the $g(i,j)$ zero or one. The pivot step then becomes particularly simple, consisting in subtraction of the pivot column from all other columns with a minus one entry in the pivot row.

To obtain a stronger algorithm we suggest the use of the reduction procedures of <2>, as follows:

(a) Drop the objective function row and all feasible rows (those with nonnegative constant term).

(b) Drop all positive terms in the resulting matrix and multiply each column with the range (bound interval) of the corresponding variable (initially 1).

(c) Reduce the resulting matrix as in <2>, to obtain a set of minimal preferred inequalities.

These inequalities have the form (2.9) with all coefficients $(-h(i),-g(i,j))$ equal to 0 or -1, and it is clear that for any Gomory inequality as described above there exists a minimal preferred inequality which is either equal to it or better, in the sense of having fewer coefficients.

To have fewer coefficients in the pivot rows than usual is important in that it obviously delays the onset of the well-known degeneracy situations which impede the Gomory all-integer algorithm in practice.

A simple example may be useful. Let all $t(j)$ have range 1 and let an infeasible inequality of (2.7) be:

$$s = -9 + 3\,t1 + 5\,t2 - 4\,t3 + t4 \text{ , or}$$
$$3\,t1 + 5\,t2 - 4\,t3 + t4 \geq 9 .$$

Then the Gomory cut with λ large is:

$$t1 + t2 + t4 \geq 1$$

and a minimal preferred inequality is:

$$t2 \geq 1$$

3. (Pseudo) zero-one zero-one representation.

The representation of the dependent variables in terms of the independent variables will be said to be "zero-one zero-one" as long as Q has entries 1, -1, 0 and γ has values 0 and 1 only. A computer code could then keep the logical part of A in the form of two bit matrices (one for the non-negative and one for the nonpositive entries) of dimension $(2n,1+n)$.

Let A' consist of the first $(1+m+n)$ rows of A, after replacement of any row in the (γ,Q) portion of A with $\gamma(i) = 1$ by the corresponding row in the $(\bar{\gamma},-Q)$ submatrix of A. The last n rows of A' thus have constant terms 0. Hence one has a representation of the form

$$\begin{aligned} z &= v + (-c) \cdot (-t) \\ s &= b + C \cdot (-t) \\ \tilde{y} &= 0 + L \cdot (-t) \text{ , or} \end{aligned} \tag{3.1}$$

$$A' = \begin{pmatrix} v & -c \\ b & C \\ 0 & L \end{pmatrix} \tag{3.2}$$

A given row i of $(0,L)$ represents either $\tilde{y}(i) = y(i)$ or $\tilde{y}(i) = 1 - y(i)$ in terms of the current non-basic variables t.

It will be necessary to consider L partitioned (after rearrangement of columns) into:

$$L = (P,Z,N) , \tag{3.3}$$

P having at least one +1 in each column and Z and N having only non-positive entries. The columns in Z are those with zero reduced costs (or with unacceptably small reduced costs). Within P, and also within (Z,N), let the columns be ordered in terms of increasing reduced costs.

As discussed in section two, a pivot row is generated from a row of (b,C) with $b(i) < 0$. It has a -1 in the constant column and 0 or -1 coefficients elsewhere. The pivot step alters A' by subtracting the pivot column from all other columns with corresponding -1 entries in the pivot row. The pivot row itself is auxiliary and is not retained.

The following properties are worth noting: (assume that $c>0$)

1. The initial tableau, corresponding to (2.2) and (2.3), is a zero-one zero-one representation,

with P void, and $(Z,N) = -1$. Moreover, any pivot step of the all-integer zero-one algorithm will lead to a second tableau which is a zero-one zero-one representation.

2. Given a zero-one zero-one representation at a general iteration, consider a pivot step based on a pivot row with non-zero entries in (Z,N) only. The pivot column and the columns to be modified have no positive entries in L, and the pivot step will lead to another zero-one zero-one representation.

3. A necessary and sufficient condition for a pivot step to yield another vertex of the hyper-cube (i.e., a relation with $\gamma(i) = 0$ or 1) is that the pivot column be not in P.

If a pivot row can be constructed with 0 entries outside of N, the pivot column will be in N, and the pivot step will not only lead to another zero-one zero-one representation, but will also result in an increase in the objective function (will not be degenerate).

It will be noted that necessary conditions for 2. can not be stated easily. While any deviation of the L matrix from the (1,-1,0) format will derive from a pivot step with a -1 pivot row coefficient in a column of P, some matrix entry changing to (+1)-(-1), such a change will only take place if there is a -1 pivot column coefficient in the row of the +1 entry about to change. Given a particular pivot row candidate, one can check for this case.

But that is not likely to help much in a general procedure for constructing "favorable" pivot rows, i.e. pivot rows which maintain zero-one zero-one representation.

4. <u>(Pseudo) zero-one zero-one algorithms</u>.

A zero-one zero-one algorithm would be characterized by a sequence of zero-one zero-one representations of the dependent variables. Geometrically one would move along edges of the polyhedron determined by (2.1) and the cuts of the all-integer algorithm from one (infeasible) vertex of the hypercube to the next. Attaining a feasible vertex would amount to termination of the algorithm.

The prefix (pseudo) is meant to make it clear from the outset that such a goal can hardly ever be attained. But one might hope to delay the failure of the zero-one zero-one procedure until a point at which the number v might have a usefully large value (say, a value larger than the objective function value of the relaxed linear program over (2.1)) by virtue of the following observation:

> All difficulties are related to the presence of a -1 in the pivot row in column j of the P matrix, and any such given -1 entry can be removed. One need merely replace the pivot row by the sum of the pivot row and one of the rows of L which have a +1 entry in column j. The resulting row has a zero in the column in question and retains the constant term of -1.

If necessary, reduction may then be used to give a new pivot row which is guaranteed not to have a -1 entry in the initially offending column j .

Of course this does not prevent the presence or creation of another negative pivot row entry in a column of P . What is needed is a systematic procedure for elimination of all such entries.

Such a procedure will involve the creation of a linear combination of given rows to be, eventually, added to a pivot row with undesirable entries of -1. If the linear combination can be made to have a negative constant term and positive coefficients everywhere else, or wherever needed, then the resulting final pivot row will have no undesirable entries.

We shall describe two procedures. One can not expect them to be fully successful. Typically, failure occurs when P (or some transformed version of P) contains a "cycle" such as:

$$\begin{array}{llll} +1 & & -1 & \\ -1 & +1 & & \\ & -1 & \ldots & +1 \end{array}$$

The zero-one zero-one algorithm will be similar to the all-integer zero-one algorithm given in section 2. What distinguishes it is an attempt at constructing a pivot row with no -1 entry in a column of P (or of P and Z).

Consider two "transformation" procedures with four typical steps:

1. Select a submatrix T of $D = \begin{pmatrix} C \\ L \end{pmatrix}$

   with rows indexed by $I(T)$:
   either (i) the infeasible rows of C and all rows of L
   or (ii) all rows of D .
   and with columns indexed by $J(T)$:
   either (i) all columns of P ,
   or (ii) all columns of (P,Z).

2. Order the columns of T , columns of P always preceding all other columns.

3. If possible, construct an (at least partial) non-negative linear combination of the rows of T , say $u \cdot T$.

4. Using the result of 3 , add $u \cdot [d(I(T)), D(I(T))]$ to the pivot row so as to produce a new pivot row with no undesirable -1 coefficients $(d = \begin{pmatrix} b \\ 0 \end{pmatrix})$.

Procedure 1:

Use linear programming. Determine a non-negative multiplier u subject to:

$$\begin{aligned} u \cdot T &\geq e \\ u \cdot d(I(T)) &\leq 0 \quad . \end{aligned} \tag{4.1}$$

Procedure 2:

Use a heuristic procedure in which a sequence of matrices

$$T(0) = T, T(1) = M(1)T(0), T(2), \ldots, T(k) = M(k)T(k-1) \tag{4.2}$$

is constructed by nonnegative linear transformations M(j), j < k , such that T(k) has all entries in columns 1 through k positive.

This can certainly be done for k = 1 , since column 1 of T contains a plus 1 entry from P . The procedure can be continued until a column k in T(k) (k ≤ no. of columns of T) contains no positive entry.

Details and possible modifications.

The above has been kept somewhat vague since it is not completely clear what constitutes a best approach.

First of all, it is not obvious in what order one should:

(i) transform

(ii) generate possible pivot rows by reduction (in the sense of <2>).

In our implementation we first executed the heuristic transformation procedure on T = (P,Z) with all rows of L and the infeasible rows of C , followed by reduction of the matrix product of M(k) by the I(T) rows of (d,D). When the resulting pivot rows were all unsuitable, we made another attempt (rarely successful) at removing -1 entries in a given row r(i) by addition of suitable rows in P .

We ordered T(0) only, keeping the columns of P corresponding to nonbasic r(i) leftmost, followed by the nonbasic y(i) and then by the r(i) and y(i) of Z .

A possibly better procedure might be:

(i) Generate a number of minimal preferred inequalities, i.e. a number of possible pivot rows r(i) = -h(i) + g(i). t . For every such candidate:

(ii) Form T as above, but with the corresponding entries of g(i) adjoined as an additional, say last, row.

(iii) At step j in the transformation leading to the sequence (4.2), reorder T(j) according to the last row, keeping essentially the negative entries at the left.

The generation of the sequence could then possibly terminate successfully with k < n when no negative entries remain in the last row of T(k).

Note that the procedure 1 can also be improved by a weakening of the constraints. I.e., if (4.1) is infeasible, one can relax the problem by changing to 0 entries of 1 in e corresponding to columns j of T with no negative coefficient (-g(i,j)) = -1 in the pivot row. Finally, for each r(i) in the modified procedure, with pivot column j not in P, one may check whether a pivot step to another zero-one zero-one representation is after all possible (see the remarks concerning the necessity of the conditions in section 3).

5. Applications, Enumeration.

In the following, we assume that the zero-one algorithm cannot be or is purposely not continued (terminates) at iteration p .

The utility of the algorithm depends almost entirely on the value of the objective function bound v at termination. If it is reasonably large compared with the objective function of the linear programming relaxation for (2.1), then a number of applications are possible and attractive.

Such applications would rest upon the fact that the solution $(z,y)^p$ at termination step p could be considered a solution of a different relaxed version of (2.1). As such, it could for example be used in branch and bound (BB) schemes.

At each node of the BB procedure one could take z^p as the objective function bound and could utilize the tableau for guidance in branching. The integrality and zero-one zero-one nature would be important (certainly on machines with large ratios of floating point to fixed point execution times).

Another possibility at termination is the dropping of rows in P which lead to difficulties. This has the effect of relaxing the problem further, with the compensating advantage of permitting the zero-one zero-one algorithm to proceed. For the 6 by 12 test problem we can attain the integer problem objective function value (z = 13) in such a fashion.

In this section we shall briefly concentrate on enumerative procedures using one termination tableau only. We use three sets of data:

1) The initial tableau: Y - representation.

2) The zero-one zero-one representation at termination: YR - representation.
(the independent variables t are a known mixture of y(j) and r(k)).
3) The relation between the introduced vector r = (r(k)) of slack variables and the structural variables y(j).

$$Y: \quad \min c^o \cdot y \quad C^o \cdot y \le b^o \qquad YR: \quad \min c \cdot t \quad C \cdot t \le b \quad L \cdot t \le 0 \quad t..\text{given vector of } y(j) \text{ and } r(k) \tag{5.1}$$

$$r = \rho + R \cdot y \tag{5.2}$$

To render the enumeration more effective, we also solve the original problem as a linear program followed by imposition of Gomory-Johnson cuts (see e.g. <3>,<4>) and retain the toprow of the final tableau. It is used throughout the enumeration in "ceiling tests", i.e. comparisons with an incumbent objective function bound z* for the purpose of fixing variables or shrinking bound sets. The use of this simple procedure is probably not sufficiently widespread. It provides an excellent and inexpensive vehicle for taking advantage of cutting plane techniques.

We implemented two types of enumeration:

(i) Enumeration over y (direct use of the Y-tableau, the YR - tableau being used as auxiliary tableau of cuts (similar to the cuts of Benders <5>)),

(ii) Enumeration over the YR - tableau, with the Y - tableau used as auxiliary constraints.

In both procedures the equations (5.2) were used to provide updated bounds on the r(k) and also, when indicated, to furnish additional cuts.

The enumerations made consistent use of reduction procedures proposed in <2> to fix variables and to shrink bounds whenever possible. For an alternate approach see <9>. Minimal preferred variables were then generated and utilized. It can be shown that this involves (for procedure (ii)) multiplying a column j of the YR - tableau, corresponding to a slack variable r(k), with the current bound interval for r(k) before reduction. (See Appendix 1).

A typical minimal preferred inequality might then be interpreted as a disjunctive logical constraint of the form: either ... lower the upper bound of t(j1), or ... raise the lower bound of t(j2), or .... .

A "branch" would be selected from such a disjunctive constraint. Priority would be given to possible branches with the property of "double contraction", i.e. to branches which would satisfy one disjunctive constraint and contradict another.

The enumerations were carried out so as to associate a forward branch always with a raising of a lower bound. Such a restriction is of course not necessary. (For details see the Appendix 2)

We feel that an enumeration scheme of the above type should be among the better possible ones, as long as the reduction procedure is not expensive (this would depend on efficiency of implementation, computer execution times and similar considerations). As pointed out elsewhere, e.g. in <2>, there are important classes of mixed integer problems, with (5.1) representing a set of Benders inequalities, for which the cost of reduction would be negligible compared with the other required data processing operations.

Finally, it should be remarked that "state enumeration procedures" (see <6>, <7>, <8>) would find natural application here. Our implementation, however, did not attempt to define states.

6. Experimentation and results.

We summarize a set of experiments on five test problems (described elsewhere, e.g. in <10>), in the TABLE.

The (10,20) and (10,28) problems are maximization problems treated as minimization problems with negative cost coefficients. However, our programs require dual feasibility, which is imposed by the introduction of the complementary variables $\bar{y}(j) = 1 - y(j)$. This explains the negative constant term in column 8 (also present in the initial tableau of the linear program).

The first five columns describe the problem and an initially performed linear program with a number of added Gomory-Johnson cuts (see column 5).

TABLE

(PSEUDO) ZERO-ONE ZERO-ONE PROGRAMMING PLUS ENUMERATION

1	2	3	4	5	6	7	8	9	10	11
PROBLEMS	OPTIMAL OBJ.F. OF INT. PROBLEM	OPTIMAL OBJ.F. OF LP	OPTIMAL OBJ.F. OF LP + CUTS	NO OF CUTS	Z* IMPOSED UPPER BOUND	NO OF ALL INTEGER PIVOTS z^P_{max}	ALL INTEGER OBJ.F. AT START OF PSEUDO	ALL INTEGER OBJ.F. AT TERM. OF PSEUDO	Y-ENUM. NODES TO TERMINATION WITHOUT USE OF PSEUDO TABLEAU	WITH
(6.12)	13	6.85	8.94	5	99	5	0	10	9	9
(10,20)	-6120	-6155.3	-6155.3	0	-6100*	10	-8655	-6590	75	11
(10,28)	-12400	-12462	-12433	4	-12300	12	-15495	-14095	43	27
(28,35)	550	521.05	542.6	2	575	22	0	425	43	21
(12,44)	73	56.7	62.9	3	77	6	0	33	439	299
(12,44)	73	56.7	69.8	7	77	2	0	24	85	45

For a description of such cuts the user may wish to consult <3> and references cited in <4>. Gomory-Johnson cuts are generated from all fractional rows associated with structural integer variables. In the generation process (due to Johnson), the slacks are treated as integer variables (of course only for pure integer programs).

As described before, the top row of the final tableau (column four) is retained for use by the final enumeration procedure. It allows the fixing of variables, or shrinking of variable bounds, both for non-basic integer and slack variables. We also extract a Benders inequality from the top row (i.e., an inequality over the nonbasic structural variables) and append it to the initial integer tableau to be used during the pseudo zero-one zero-one procedure. The Gomory-Johnson cuts therefore are of importance throughout the experimental runs.

Column 6 lists an upper bound z^*, imposed throughout all computations. One exception to this is marked by a *. For the (10,20) problem the tight upper bound of -6100 makes the problem too easy for the enumeration. It is therefore replaced for that purpose by -5500.

Column seven describes how many zero-one zero-one representation tableaux were generated by the algorithm, and column nine gives the value z^P_{max} (the last objective function value). Except for the first small problem, these values are disappointingly small compared to those in columns 3 and 4.

However, it should be mentioned that experiments without transformations 4.2 yielded results which were much worse. The transformation procedures do permit a substantial increase in the figures of columns 7 and 9.

To improve matters, one would have to implement some of the suggestions of chapter 4. We do not believe that one will be able to go very much further, but we do not really know. Among other things there is always the difficulty of erratic behavior of the best planned algorithms. The last two rows of the TABLE, e.g., show in column 7, that the use of stronger cuts (more Gomory-Johnson cuts ought to lead to a stronger Benders inequality) can influence the zero-one zero-one procedure unfavorably. One can always observe isolated instances of such behavior and must trust that the rational procedure is the good one in the large majority of cases.

The matrices R consist initially of (0,1) rows. But the algorithm eventually generates rows with large integer coefficients. The buildup in upper bound values for the r(i) is extraordinarily rapid. Large bounds are clearly a determining factor in the bad convergence of the dual all-integer algorithm. Convergence of that algorithm requires that feasible values of the t(j) be found, and these values will often be very large. From this point of view, the very fact of finite convergence is remarkable.

YR - enumeration will deteriorate with increasing bounds. Experimentation shows this behavior quite clearly. Even small values of p may already be too large. For some problems we found

that the efficiency of the YR - algorithm versus p was roughly a bell-shaped function with an optimum below the maximal value we were able to obtain (column 7).

For Y - enumeration, on the other hand, it appeared that the most important factor was the attained objective function value in column 9.

Our main result is the comparison in the last two columns, the first dealing with Y - enumeration without auxiliary cuts, the second with Y - enumeration in which reduction procedures were applied to auxiliary cuts represented by the last zero-one zero-one representation tableau. As can be seen, the improvement in terms of nodes to be evaluated can be substantial.

7. Appendix:

1. Reduction procedures for bounded variable inequalities.

Consider the inequality:

$$\sum_J c(i,j).t(j) \leq b(i), \qquad \text{(A1)}$$

with $$L(j) \leq t(j) \leq U(j). \qquad \text{(A2)}$$

The transformation:

$$\tau(j) = (t(j)-L(j))/(U(j)-L(j)) \qquad \text{(A3)}$$

leads to variables in the hypercube:

$$0 \leq \tau(j) \leq 1 \qquad \text{(A4)}.$$

After the transformation A3, a typical inequality A1 takes the form:

$$\sum_J d(i,j).\tau(j) \leq \beta(i) \qquad \text{(A5)}$$

with $d(i,j) = c(i,j).(U(j)-L(j))$

and $\beta(i) = b(i) - \sum_J L(j) \cdot c(i,j)$.

In the following we assume that these transformations have been carried out. A generalization of the reduction procedure of <2>, to be described more fully in <11>, can then be applied to the transformed data. It associates with each constraint (A5) a minimal preferred inequality of the form

$$\sum_{\pi(i)} \bar{\tau}(j) > 0 \; . \qquad \text{(A6)}$$

$\pi(i)$ is a set of "minimal preferred variables" and $p(i) = |\pi(i)|$ is called the "preferred weight" of row i (the term "minimal" refers to p(i); what is minimized by the reduction algorithm is the number of terms in $\pi(i)$).

When all $\tau(j)$, $j \in \pi(i)$, are bivalent variables, then (A6) yields the normal interpretation: at least one of the variables $\bar{\tau}(j)$ must be one if the original inequality is to be satisfied.

When the $\tau(j)$ are continuous variables, $0 \leq \tau(j) \leq 1$, $j \in \pi(i)$, then the interpretation of (A6) is the following:
not all $\bar{\tau}(j)$, $j \in \pi(i)$, can be zero, therefore not all t(j) , $j \in \pi(i)$, can simultaneously be at their lower (upper) bounds. One of the bounds at least must be raised (lowered) by no less than one unit.

If $p(i) = 0$, the problem is infeasible.

If $p(i) = 1$, then $\bar{\tau}(j)$, $j \in \pi(i)$, must be positive, and L(j) must be raised by one unit if $\bar{\tau}(j) = \tau(j)$, or U(j) must be decreased by one unit if $\bar{\tau}(j) = 1 - \tau(j)$.

In the general case $p = \min_i \{p(i)\} > 1$, one retains all minimal inequalities of size p. Suppose there are k such minimal preferred inequalities. They can be represented in a k by n matrix Q . Each row i of Q has p nonzero entries in the positions j of the associated preferred variables. A +1 entry identifies the case $\bar{\tau}(j) = \bar{\tau}(j)$, a -1 the case $\bar{\tau}(j) = \tau(j)$.

The matrix Q can be used to give direction to a search procedure. Examples of such use can be found in Appendix 2, also in references <2> and <12>.

2. The essentials of the enumerative procedure.

As explained before, the enumerative procedure was coded so as to permit enumeration either over the Y - tableau or over the YR - tableau. Let x correspondingly stand for either y or t . Appropriate comments will be made where the algorithm differs for y and t . Capitalized names refer to subroutines to be described further.

1. Set the x(j) to the given lower bounds L(j) for all j . Label all variables "free". Set the node count and the "level" of the search to 0 .

2. Node computation.

2.1 Increase the node count.

2.2 Compute current working bounds in WORKB.

2.3 Ceiling test in CEILING.

2.4 Reduction (fixing of variables, shrinking of bounds) in REDUCE.

In 2.2 to 2.4 the following conditions may obtain:

(i) The current node may be "fathomed". That means: the node (with its successor nodes) is established as infeasible, or

the node is associated with an integer solution (by virtue of the chosen origin of the search and dual feasibility of the tableaux no successor node can have an improved integer solution), or

the variables are all fixed (zero bound intervals).

In such a case, go to 4 ("backup").

(ii) Variables can be fixed or bound intervals can be shrunk. In such a case, go to 2.2.

3. Forward step ("Branch").

3.1 Establish a constraint set $A \cdot x \leq b$, either from the Y - tableau or from the YR - tableau, depending on the chosen type of enumeration. As explained before, columns of bounded variables must be multiplied by the appropriate (working-) bound interval.

3.2 Compute a "minimal preferred inequality matrix Q" from the constraint set (by the methods of <2>; see also Appendix 1).

Let J be a set of indices for possible branching.

If Q is empty (which is only rarely the case), set J equal to the set of indices of the free variables (the variables not yet fixed). Go to 3.4.

Otherwise, row i of Q may be interpreted as follows:

Aside from zero entries it has exactly $p > 1$ entries of +1 or -1. The smaller a value of p (the "preferred" weight), the more constrained is the problem.

$q(i,j) = +1$ (-1) means that constraint i can be satisfied (indeed must be satisfied for at least one j) by a lowering of the upper bound (raising of the lower bound) on $x(j)$.

3.3 If there are "double-contracting" variables $x(j)$, i.e. variables such that for j there exist i1 with $q(i1,j) = 1$ and i2 with $q(i2,j) = -1$, have J include all corresponding indices j. Go to 3.4.

If not, let I be the set of indices of rows of Q such that all nonzero entries are -1.

If $I \neq 0$, set $J = \{j | q(i,j) = -1,\ i \in I\}$. Go to 3.4.

If I is void, drop all positive coefficients in A. Go to 3.2.

(The purpose of this step is to force a matrix Q with I not void above, so that the preferred variable inequalities suggest a "Forward Step" corresponding to a raising of a lower bound. Of course, the algorithm could be, and probably should be, generalized to allow forward steps which consist of the lowering of some upper bounds).

3.4 For all $j \in J$ compute the new right hand side (of the Y tableau or the YR - tableau). Choose the branch variable j^* so as to minimize overall infeasibility.

3.5 Record the forward step:

Add j^* to an index list, record $L(j^*)$, the lower bound of $x(j^*)$ before the forward step, in a bound list. Increase the level ℓ by one.

Go to 2.

4. Backward step.

Retrieve $j^*(\ell)$ and $L(j^*(\ell))$ from the index and bound lists. Fix the upper bound of $x(j^*(\ell))$ at $L(j^*(\ell))$.

Replace ℓ by $\ell - 1$. Terminate if $\ell < 0$.

Otherwise, go to 2.

Subroutines:

WORKB:

1. After a forward step, go to 3.

2. Reconstruct bounds from the current index and bound list.

3. When $x = t$, special "state" vectors for y are maintained by the program (giving for each fixed $y(j)$ its value and the level at which it was fixed).

   This information is used together with equation (2.8) to compute bounds on those $t(j)$ which are slacks $r(k)$ of the pseudo zero-one zero-one procedure.

4. Compute the right hand sides of both tableaux (Y and YR), as well as of the saved top row of the initial linear program (+cuts), by substituting the lower bounds of y and t, respectively.

   If one of the tableaux yields a feasible solution, compare it to the incumbent (currently best) solution, and record it if it is an improvement.

REDUCE:

Given a set of working bounds on y and x, there exist simple procedures (e.g. <9> or <2>) for deriving possible further restrictions on the bounds. Using these procedures, REDUCE alters bounds and fixes variables when possible. It must be emphasized that REDUCE uses both tableaux, as well as the relations 4.2, regardless of the type of enumeration.

CEILING

The top row of the initial linear program (+cuts) is of the form $z = z^{\circ} + a \cdot \sigma$, with $a \geq 0$, σ a vector of integer and slack variables, and z° the objective function value given in column 4 of the TABLE. Given an upper bound z^* on the integer objective function and bounds on σ, the condition $z \leq z^*$ may permit the computation of improved bounds on the components of σ and y.

8. References

[1] Gomory, R. E. (1963) "All-Integer Integer Programming Algorithm," in Graves, Wolfe, "Recent Advances in Mathematical Programming," McGraw Hill.

[2] Spielberg, K. (1972) "Minimal Preferred Variable Reduction for Zero-One Programming", IBM Philadelphia Scientific Center Report 320 3013.

[3] Gomory, R. E., Johnson, E. L. (August 1972) "Some Continuous Functions Related to Corner Polyhedra", Mathematical Programming.

[4] Johnson, E. L., Spielberg, K. (December 1971) "Inequalities in Branch and Bound Programming," IBM Research Report RC 3649. Also in: "Optimisation Methods", Editors: Cottle, Krarup, English Univ. Press, London, 1974.

[5] Benders, J. F. (1962) "Partitioning Procedures for Solving Mixed-Variables Programming Problems," Numerische Mathematik.

[6] Guignard, M. M., Spielberg, K. (1968) "Search Techniques with adaptive Features for certain Integer and Mixed-Integer Programming Problems", Proc. IFIPS Congress, North-Holland Publ. Co.

[7] Guignard, M. M., Spielberg, K. (June 1973) "A Realization of the State Enumeration Procedure", IBM Philadelphia Scientific Center Report 320 3025.

[8] Guignard, M. M., Spielberg, K. (1972) "Mixed-Integer Algorithms for the (0,1) Knapsack Problem," IBM Journal of Research and Development.

[9] Zionts, S. (1972) "Generalized Implicit Enumeration Using Bounds on Variables for Solving Linear Programs with Zero-One Variables", Naval Research Logistics Quarterly.

[10] Lemke, C. E., Spielberg, K. (1967) "Direct Search Zero-One and Mixed Integer Programming", Operations Research.

[11] Guignard, M. M., Spielberg, K. (to appear) "Reduction Procedures for Bounded Variables Integer Programming".

[12] Spielberg, K. (June 1973) "A Minimal Inequality Branch- Bound Method", IBM Philadelphia Scientific Center Report 320 3024.

Computers and Mathematical Programming

Proceedings of the Bicentennial Conference on Mathematical Programming held at the National Bureau of Standards, Gaithersburg, Maryland, November 29-December 1, 1976

Edited by

William W. White

IBM Corporation
Poughkeepsie, New York 12602

Sponsored by:

Special Interest Group on Mathematical Programming
Association of Computing Machinery
1133 Avenue of Americas
New York, New York 10036

and

Applied Mathematics Division
Institute for Basic Standards
National Bureau of Standards
Washington, D.C. 20234

acm
sigmap

U.S. DEPARTMENT OF COMMERCE, Juanita M. Kreps, Secretary

Dr. Sidney Harman, Under Secretary
Jordan J. Baruch, Assistant Secretary for Science and Technology

NATIONAL BUREAU OF STANDARDS, Ernest Ambler, Acting Director

Issued February 1978

An Experimental Interactive System for Integer Programming

Monique Guignard

Wharton School

University of Pennsylvania

Kurt Spielberg

IBM Scientific Marketing

Abstract

The paper describes an experimental interactive system for the solution of integer programming problems (primarily of 0-1 nature, but not exclusively so)

The system includes most techniques which appear to offer hope of overcoming the combinatorial difficulties of all integer programming algorithms: in particular LP with cuts, BB with propagation, State enumeration, Interval reduction, Preferred variable reduction, Exploitation of Benders inequalities, Local search, Heuristics, etc..

There is emphasis on interaction by the analyst at the terminal. An example illustrates the use of various tools. A table summarizes results obtained by a number of different approaches to a problem of moderate difficulty.

1. Introduction

We describe an experimental interactive system for 0-1 programming. The programs are written in APL. Various techniques have been or are planned to be incorporated in this system. They are intended to be called by the user, independently from each other (subject to restrictions which we would like to make as little painful as possible), or sequentially, depending on the results of the experimentation.

No single technique can be expected to solve all types of integer problems. But our preliminary experimental results are encouraging and suggest that a truly flexible interactive system has potential for aiding in the understanding and solution of integer programs, beyond what can be expected from a standard preset program.

The paper is not intended to give all algorithmic details. In section 2 an overview of techniques and features is given, with references to expository papers. For convenience, certain key concepts are briefly summarized in an appendix.

What counts here is that there are certain techniques for "looking" at the problem, gleaning more information from it in simple form (e.g, in terms of simple logical relations among variables, simple inequalities, stronger bounds, etc.) by judicious action at a computer terminal, guided by the printout of the "current" state of the solution process. The user may find some help from the example presented in section 3.

Can interactivity play a major role in solving difficult problems? We increasingly believe that the answer is yes if the user of the system is well versed in integer programming. Whether one shall be able eventually to construct useful interactive system for the non-specialist is somewhat difficult to predict now. We are optimistic, but much work remains to be done.

2. Techniques of the System

At present the following features and techniques have been incorporated :

2.1 Linear Programming (LP),

with possible addition of Gomory-Johnson cuts and dual reoptimization until either no real progress is made any more (in terms of changing objective function), or until the number of reoptimizations or cuts reaches upper bounds imposed by the user. <1,2,3,4>

2.2 Branch and Bound Programming (BB),

with propagation on nonbasic (possibly preferred) variables. A single LP is solved at the current origin (or node) of the search tree and penalties are computed. A branch on a nonbasic variable set at its optimal LP value is called propagating. One will want to propagate as long as the "alternate"

penalty is large, <5,6,7>.

2.3 Benders inequalities,

which are generated at the end of an executed LP. They are intended to be used either

(I) after solution of a problem within a BB or Enumeration scheme (usually with some aid from cutting planes), or

(II) after guesses at partial integer solutions have been entered by the user in order to get logical conditions which might characterize the problem. After a guess is entered, the remaining linear program is resolved and the Benders inequality is taken from the final LP tableau (whether feasible or infeasible).

Note: it may not be permissible to use cuts in such an instance.

In general, Benders inequalities are collected and retained for exploitation in terms of reduction (see 2.4, 2.5 below). In contrast, Gomory-Johnson cuts (and we believe other cuts of similar genesis as well) are not well suited for reduction (i.e., they lead to logical inequalities of large degree). After use in an LP, they are therefore usually discarded. <8,9>

2.4 Reduction Procedures,

are invoked throughout to
(i) shrink bound intervals , and
(ii) to generate logical inequalities

The interval reduction procedure is used iteratively, screening all constraints so as to compute the tightest possible bounds on structural and slack variables. Great care must be taken to avoid round-off difficulties; i.e., one must use appropriate scaling and tolerances (as in LP), especially when Benders inequalities are used for reduction. <9,10,11,12,13>

2.5 Logical inequalities.

Given any inequality in (0,1) or integer variables, one can derive from it a set (possibly empty, but not usually so) of minimal preferred variable inequalities (m.p.i.'s). The degree of the system ("size" of the smallest inequalities; the degree of an inequality I will be called PI(I) in the printouts) is a good measure of the tightness of the problem. The bound interval of one of the preferred variables, at least, must be shrunk by one unit.

These logical inequalities are among the main tools for guiding both the BB scheme and the enumeration (see also "penalty improvement"). <9,10,14,15>

As the referee points out, there exist logical inequalities ("Boolean", "Canonical" inequalities; e.g., see <16,17>) which are more general than m.p.i.'s. They are especially important in the complete characterization of the underlying problem. However, their overabundance may present computational difficulties. We believe we have good reasons for preferring to work with the more special properties of m.p.i.'s (used computationally already in <9>).

2.6 Enumeration,

is based on the additive algorithm of Balas <18>, modified in a number of ways, with the search starting at the origin y=(0,0,...,0). Branches are restricted to minimal preferred variable sets. The actual set to be used is the union of those preferred variables (in minimal preferred sets) which have good contraction properties:
1 st priority: double contraction
2 nd priority: single contraction
3 rd priority (default): all free variables.
(a branch is "contracting" when the degree of the system is guaranteed to be decreased as a result. Contraction can be determined from the set of minimal preferred inequalities). Within the finally chosen set, the actual branch variable is selected so as to lead to minimal (maximal) overall infeasibility of the problem at the next node in phase 1 (phase 2) of the enumeration. <10>

In the above context, phase 1 is meant to be the period during which the search is directed towards finding improved feasible solutions, whereas phase 2 is to deal with the establishment of optimality (possibly within certain tolerances) once a good integer solution has been generated.

Note: These branching criteria can be rendered inoperative when the "local search" (see 2.10) identifies an improving direction.

2.7 State Enumeration,

differs from enumeration as follows. At any node, a state is determined (usually by rounding of an LP

solution). It is essentially a particular value for the integer vector y, say $\hat{y}$, believed to be close to feasible integer solutions. In the zero-one case, the search variables for an enumeration such as that of 2.6, are then taken to be y(j) if $\hat{y}(j) = 0$ and 1 - y(j) if $\hat{y}(j) = 1$. (See <12> for the generalization to the integer case with small bound intervals).

I.e., the final enumeration is carried out with transformed variables y(j) set at 0 initially and increased on forward steps of the algorithm (as in <18>). In dynamic State Enumeration, we can envisage that the search variables y(j) may be redefined over the free variables at each node, provided that reasonably good information is available for doing so.

We have found State Enumeration, with strategies roughly as outlined in 2.6, 2.7 (there are many other possibilities of sometimes highly complex nature), highly effective in finding good feasible integer solutions fast, especially when augmented by a simple "one-level" local search and by a "ceiling test" (see <2.8,2.9>). <3,4,14>

2.8 Ceiling Test

Most enumeration schemes have tests involving the objective function of the initial tableau (often with non-negative cost coefficients). In state enumeration, the original objective function becomes less interesting and can be taken care of, anyway, as a special Benders inequality. Instead of the original objective function, we have found it useful to concentrate on the final objective function row from the optimal tableau of an initial LP problem (with cuts added when deemed desirable). The row is retained (coefficients and all necessary information about the nature of the nonbasic variables) and permits the fixing of variables and/or shrinking of bound intervals for the nonbasic variables, among them original slack variables as well as structural variables.
It should be noted that the shrinking of slack bound intervals can be used as input to the various reduction procedures and can lead to further information about all the variables there.

2.9 Bounds on Slacks; Compression

Our linear programming system and all related integer programming procedures permit the imposition of upper and lower bounds on the slacks. Equality constraints, for example, are handled by imposition of zero bounds above and below. The minimal inequality reduction procedures work with inequalities only, and we generate two such inequalities as input to reduction whenever we know (or believe) that the upper bounds on the slacks are "true" (i.e., truly restrictive) bounds.

Some of our sample problems are known to have two inequalities representing actually only one equality constraint. The system therefore has a "compress" function detecting such situations and (re)creating the (original) equality constraints.

2.10 Local Search

Any BB or Enumeration algorithm may benefit from a search around the "current" point under consideration. In the present system, the "depth" of the local search is controlled by one parameter (LEV, or λ). The value 0 corresponds to no search (just one, the current, point considered), the value 1 corresponds to the alteration of all f "free" (not fixed) variables by one unit at a time. In general, one enumerates the $(f!)/(f-\lambda)!$ adjacent points. In view of the exponential increase in the number of such points, only the values 1 and 2 and conceivably 3 appear to be reasonable for λ.

2.11 Heuristics for Feasible Solutions

Several heuristic methods have been programmed which try to generate feasible solutions to a system of linear inequalities in 0-1 variables. One of them starts with a given 0-1 vector and modifies the component which maximally decreases the sum of the infeasibilities or the number of infeasibilities. Such steps are repeated as often as specified. One way of using such modifications is to have a complete "forward pass" over all the variables, followed by a complete "backward pass", i.e. all variables are altered in a pass, the sequence being determined by one of the criteria mentioned above, <19>.

Another heuristic concentrates on the logical inequalities of small degree and tries to generate a feasible solution to the current system of minimal preferred inequalities, which can then be used as a starting point for the first heuristic, or for a local search, or as a state for the state enumeration algorithm. <3,4,14>

2.12 Row Combination

It has been known for a long while that linear combinations of rows may be more interesting than the rows themselves, but there is little insight into how the inequalities are to be generated (possible exceptions are references <20> and <21>). Consistent with the main emphasis of this paper, we have chosen to take the degree of an inequality as its basic measure of strength. Hence we allow for "visual" combination of ($\leq$) inequalities (e.g., so as to have coefficients of opposite sign combine to produce small results in magnitude on the left and large negative right hand sides, if possible), and we test the resultant inequality by reduction. Alternatively, we also have heuristic algorithms of modest capability which utilize minimal preferred inequalities to give automatic row combinations, with reduction used again to check out the results and terminate the process.

2.13 Improvement of Penalties

Penalties can be improved by taking into account logical conditions on the integer variables, such as the minimal preferred inequalities. If some nonbasic variable must be modified, it may be the case that at least some of a set of other nonbasic variables must also be altered to achieve feasibility. This usually leads to the accrual of an additional penalty. One may utilize improved penalties a priori in a preprocessing of the penalty table for BB programming, or dynamically to store nodes with increased alternate penalties in BB with propagation. <6,7>

---

3. An Example of a Terminal Session

1) START1

BM1515 is a (16 by 16) tableau, of the form

$$\begin{bmatrix} Co & C \\ -B & A \end{bmatrix}$$

corresponding to the minimization of $z = Co + C \cdot y$, subject to:

$$A \cdot y \leq B$$

$$0 \leq y \leq 1, \text{ all } y(j) \text{ integer} .$$

It represents a 15 variable problem of moderate difficulty, with 15 constraints and a dense constraint matrix (taken from <22>, problem #13).

```
      START1
ENTER M,N
⎕:
      15 15
ENTER TABLEAU, EXPANDED FORM
⎕:
      ⎕←BM1515
  0   7   4   1   3   4   2   6   2   1   5   1   5   7   2   7
-36   2   6   1   0   3  -3  -2  -6  -2   2  -5   3  -3   7   9
 22  -6   5  -8  -3   0  -1  -3  -8  -9   3  -8   6  -3  -8  -6
  3   5  -6  -5  -3  -8   8  -9  -2   0   9  -1   7   9  -9   4
 -1  -9  -5   0   9  -1   8  -3   9   9   3   0  -7   5   4  -9
-10   8  -7   4   5   9  -1   7   1  -3   2   0  -3  -5   9   7
-12   7   5   2   0   6   6   7  -6  -7  -7  -1  -8  -3   9   1
  1  -1   3   3  -4   1   0  -4  -1  -6   0   8   0   1  -5  -4
  2  -1   2  -6  -9   0   7  -9   9   6  -4   5   3   1  -3  -9
  3   4  -6  -7  -2  -2   0  -6  -6   7  -4   0   2   8   0   4
 -2   0   0   0   0   0   0   0   0   0  -1  -1  -1  -1   1   1
-14  -1   3   3  -4   5   5   7   8  -8   0   0  -1  -2   4  -6
 -9   8   0   5   0   2   6   7   1  -1   1   4  -5  -7  -8   2
 25  -2  -7  -8   6  -2  -5  -9  -2   2  -8   1  -3   5   6  -8
 -3   7   9  -7  -5   1  -5  -5   4   9   3  -4   0   0   7  -7
-32   7   1   4  -3   0   6  -3   7   8   1  -6   6   8  -3   8
```

2) Initial LP. Preprocessing.

No variable can be fixed by reduction in the initial preprocessing. The continous LP optimum is 14.96 and 5 variables out of 15 are fractional. A Benders inequality is computed from the optimal LP tableau and printed out. (ZSTAR stands for the best objective function value found, or for an upper bound supplied by the user. We use a large value as the default).

One enters a "state" ST1, using a rounded LP solution with rounding parameter ρ = .5 , (entries 1..keep y(j), entries 0..use complemented variables 1 - y(j)) .

```
BENDERS INEQUALITY OF TYPE 1
( 5.844 0 ¯6.122 6.741 3.03 0 0 ¯1.121 0 0 0.7512 2.862 8.058
1.963 ¯0.7163 ) * Y ≤ ¯22.92 +ZSTAR

ST1=1 0 0 1 1 1 1 0 1 1 1 1 1 1 0
```

3) Guesses

One is prompted for guesses. One enters indices and corresponding values for a subset of the integer variables (unspecified variables being left unconstrained) and resolves one resulting LP.

The final LP tableau yields a Benders inequality of type 1 if feasible, of type 2 if infeasible.

The function GUESS works with the last Benders inequality.

GUESS 0 chooses the free variables with negative coefficients and sets them to 0. The guess is then likely to be infeasible and a new feasibility condition will be generated (B.I. of type 2).

GUESS 1 chooses the free variables with positive coefficients, and one sets these variables to 1. In this case also one constrains the current B.I. and tries to obtain tighter feasibility conditions on a subset of the variables.

Any given guess leads to the resolution of one linear program. While the Benders inequality is retained for further processing (possibly after a check as to whether it is strong enough, i.e. has degree low enough). For large problems, the guesses will most likely be generated automatically , according to a scheme which the user may have developed for a smaller problem of similar structure.

Having some familiarity with a given problem (and by that we do not mean knowing its solution), one can usually think of a number of other promising guesses. E.g., one may want to set to one some variable(s) with large positive coeffient(s) or at zero some variable(s) with large negative coefficients; or one makes guesses in consonance with one's knowledge of any possible special structure.

```
WANT TO GUESS AT SOLUTION?
YES
ENTER GUESS FOR Y, FIRST INDICES, THEN VALUES
□:
     GUESS 1
2 10 12
□:
     1
ACTUAL NUMBER OF ROWS FOR MAT :18
PROBLEM NOT FEASIBLE
DETERMINANT=   1.004E6
BENDERS INEQUALITY OF TYPE 2
( 36 ¯1.563 ¯10.11 0 11.02 5.74 0 0 ¯6.753 25.63 0 6.149 0 0 0 )
* Y ≤ 12.12
```

```
WANT TO GUESS AT SOLUTION?
YES
ENTER GUESS FOR Y, FIRST INDICES, THEN VALUES
□:
      7
□:
      0


11.47  SEC
OBJECTIVE FUNCTION =  15.12
NO. OF PIVOT STEPS = 29
DETERMINANT=   1.045E6
STRUCTURALS: 0.01126 0.7627 1 0 0 0.0874 0 1 0.3813 0.2613 0.1264
 0 0 0 1
BENDERS INEQUALITY OF TYPE 1
( 0 0 ¯9.671 8.708 2.266 0 ¯3.909 ¯0.6645 0 0 0 3.872 11.21 2.857
 ¯6.548 ) × Y ≤ ¯32 +ZSTAR
```

4) After all guesses have been entered, and the corresponding LP's have been solved, with a final Benders inequality always adjoined to the tableau, PREPROC checks for interval reduction and fixing of variables. In this problem variables y1, y9 and y15 can be fixed at 0, 1 and 1, respectively.

```
N. VARS. FIXED 3
      LU2[1;]      (lower bounds)
0 0 0 0 0 0 0 0 1 0 0 0 0 0 1
      LU2[2;]      (upper bounds)
0 1 1 1 1 1 1 1 1 1 1 1 1 1 1
↑                 ↑             ↑
y1                y9            y15
```

5) The starting LP for the BB (or an Enumeration) procedure is then solved with these values substituted, and the initial objective function value is thus raised to 15.7 . Dual reoptimization after the generation and temporary addition of Gomory-Johnson cuts finally brings the initial BB objective function to 18.39 .

```
OBJECTIVE FUNCTION =  15.7
NO. OF PIVOT STEPS = 25
DETERMINANT=   4.967E5
STRUCTURALS: 0 0.8451 1 0 0 0.1926 0.1758 0.3951 1 0.2186 0 0 0 0
 1
BENDERS INEQUALITY OF TYPE 1
( 7.362 0 ¯4.894 5.722 1.515 0 0 0 3.848 0 0 1.463 8.021 3.042
1.645 ) × Y ≤ ¯15.1 +ZSTAR
5  CUT(S) GENERATED



OBJECTIVE FUNCTION =  18.39
NO. OF PIVOT STEPS = 87
BENDERS INEQUALITY OF TYPE 1
( 10.25 0 ¯3.911 3.535 0.09996 0.06115 0 0 4.343 0 0 0 5.384
0.004794 1.39 ) × Y ≤ ¯16.57 +ZSTAR
STRUCTURALS: 0 0.1975 1 0 0 0 0.6106 0.4917 1 0.1856 0.101 0.5849
 0 0 1
```

6) BB starts. The system computes penalties (added to the current objective function value to give a maximal lower bound on the final objective function value).

In the normal BB mode, the generated node (i.e., the current bounds and z + penalty) is stored in the node table, and the next node would be brought in.

In the propagation mode one looks for variables to be fixed at their current value, with an alternate node stored in the table, and the current node being processed further.

In this example a propagating variable y(13) is identified and the alternate (y(13)=1; often other conditions can be imposed from the minimal preferred inequalities) is stored with penalty 5.384.

The user is asked whether he accepts the propagation. If the answer is yes, the propagation proceeds; if not, the user has the choice of proceeding in the normal BB mode or of trying to finish the current problem (in core) by state enumeration. In the present case we accept the first propagation (storing an alternate node with penalty 5.384) and reject the second possibility with alternate penalty 3.911 (the system, of course, generates the penalties in descending order of magnitude).

```
PRP.PEN.CTR 13 5.384 0   (CTR=0 ↔ NO CONTRACTION)
                         ( PRP=13 ↔ PROPAGATE WITH Y13)
ACCEPT      ?            ( PEN=5.384 ↔ WITH ALTERNATE
YES                                   PENALTY PEN)

OUT:        (STORE NODE)
**********************
1 23.77 5.384 -13  (NODE NUMBER, OBJ.F. VALUE, PEN., BRANCH)


PRP.PEN.CTR -3 3.911 0

ACCEPT      ?
NO
FOR ALTERNATE      CALL STRAT. BP. HERE
SAVE NOW. BEFORE ENUM.
```

7) Enumeration starts (from the state ST1). Variable 13 is at 0 (via propagation), and y1, y9, y13 have been fixed globally. One branches on y8=1, and a feasible solution is found immediately (undoubtedly due to the imposition of the state), with obj function z=26.

```
PI 4 4 4 2 2 4 3 2 3 10 5 3 4 3 4 10 4 2 2 2 10 10 2 10 3 2 10 5 (row degrees R(i)
K IS 1   (local search level)
Z     26
 FEASIBLE SOLUTION 26
SOLUTION FOUND AT NODE  2 LEVEL 1 PHASE 1
************************************************************
OBJECTIVE FUNCTION 26
NEW ZSTAR =26
SOLUTION 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1
INT. SOL. FEASIBLE  , Z= 26
```

8) Enumeration is completed after examination of a total of 17 nodes.

```
L/BND= -1  INF. IN BOUNDRD  (IN BOUND REDUCTION)
END OF ENUMERATION
******************
OPTIMAL SOLUTION 0 1 1 0 0 0 1 1 1 0 0 1 0 0 1
OPT. OBJ. FUNCTION 26
```

9) Return to BB.

The stored alternate node is brought back from the tree into high speed storage. After imposition of the bounds, the inequality system (including Benders inequalities) with $z \leq z^{*}-1 = 26-1 = 25$, is now infeasible. End of BB.

Termination of integer program.

```
IN OF NODE 1
*****************
1 23.77 5.384 ¯13  (NODE NUMBER, OBJ.F.VALUE, PEN., BRANCH)

RED     (REDUCTION)
NOPV 0  (π OF SYSTEM)
pPLUS 20 15
INFEASIBLE;NOPV      0
```

4. Computational Results

There are a number of problems which can be entirely resolved by the use of one or the other technique. For example, a (28,35) problem of the tanker scheduling variety (see <9>) yields an integer solution after imposition of a total of 6 Gomory-Johnson cuts in 2 reoptimizations.

The same problem yields well to state enumeration techniques (11 BB nodes of the propagation type, i.e. with only one true linear program resolved, followed by between 9 and 33 enumeration nodes depending on the initial state; at the end none of the alternate BB nodes need to be processed further).

On the other hand it requires a large number of LP optimizations in a straightforward BB optimization run (between 40 and 109 linear programs depending on strategy).

In the following table we describe various approaches to the (15 by 15) problem used as example in the text. In some respects it is rather difficult: the gap between LP obj. function and integer solution objective function is large; the penalties are rather small ; the response to Gomory-Johnson cuts is only moderate; the degree of the initial system is 3. It behaves much better when Benders inequalities are added, permitting the reduction of the degree to 1 (fixing of three variables); etc..

In the table we summarize a number of runs using some or all of the available tools. One does or does not add Gomory-Johnson cuts. One does or does not enumerate at a selected pending node of the BB tree. The state for enumeration can be chosen arbitrarily: we ran the program with the state determined from the rounded LP solution, with all fractional variables less than or equal to .5 fixed at 0, and all variables larger than .5 fixed at 1.

We entered some guesses at a solution and appended the resulting Benders inequalities to the original tableau.

Which of the results are considered to be best will depend on the relative computational efforts, and therefore on the computer and on the system used (APL in our case). In general, we feel it most essential to reduce the number of linear programs, but even more fundamentally, to avoid large trees (many BB nodes, large enumeration levels), i.e. situations in which the combinatorics overwhelms the problem solver.

The interactive system appears to give the insight and often the tools for avoiding exponentially intractable situations; i.e., even for more difficult problems (e.g., the (37 by 74 problem of <9>), which we did not run to conclusion because of slow APL response at the terminal, it was clear that the search was "well-behaved", the tree remaining a "low-level tree" , with steady progress being made .

Initial LP value	14.96	14.96	14.96	14.96	14.96	14.96
Total n. of cuts	0	0	0	47	35	41
reoptimizations	0	0	0	7	7	7
Final value of LP	14.96	14.96	14.96	16.83	16.53	18.33
Integer optimum	26	26	26	26	25	25
Use B-S?	no	yes	yes	yes	yes	yes
N. of nodes generated		13	2	2	2	2
..... called		11	2	2	2	2
Optimum found at node		12				
Use enumeration?	yes	no	yes	yes	yes	yes
State for enumeration from LP?			yes	yes	yes	yes
1?	yes					
0?						
N. of nodes	105		63	37	13	17
Optimum found at node	73		31	25	10	2
Use guesses?	no	no	no	yes	yes	yes
Type of guesses				random		
N. of guesses				10	6	10
N. of Benders ineq.				12	13	12
N. of variables fixed				0	1	3
N. of LP solved	0	25	2	9	7	10

Table

Six Approaches to Solving the (15,15) Sample Problem

5. Appendix: Definition of Some Terms (10)

- A logical inequality in bivalent variables is meant to be an inequality with 0, 1, -1 coefficients only.
- A preferred variable inequality (abbreviated p.i.) is a logical inequality of the form $\sum \tilde{y}(j) \geq 1$, $\tilde{y}(j)$ being either $y(j)$ or $1 - y(j)$, which expresses the condition that at least one of the $\tilde{y}(j)$, $j \in J$, must be one.
- The values $\tilde{y}(j) = 1$ (i.e., either $y(j)=0$ or $y(j)=1$, depending on the nature of the $\tilde{y}(j)$) are termed the suggested or indicated values of the preferred variables.
- Given some procedure for generating a set of preferred inequalities, we call minimal (relative to the procedure) those p.i.'s with minimal number of non-zero coefficients.
- The degree d of a m.p.i. system (and by extension the degree of the initial inequality or system of inequalities from which the m.p.i. system was derived) is the number of non-zero coefficients of one of the minimal preferred inequalities.

6. References

<1> Gomory, R.E., and E.L. Johnson, "Some Continuous Functions Related to Corner Polyhedra" , 1. Math. Progr., Vol. 3, #1, 1972; 2. Math. Progr., Vol 3, #3, 1972

<2> Guignard, M., "Inegalites Valides de Gomory-Johnson", Journees de Combinatoire de l'AFCET, Dec. 1971

<3> Guignard, M., and K. Spielberg, "The State Enumeration Method for Mixed Zero-One Programming", IBM Phila. Sci. C. Rep. 320-3000, Feb. 1971

<4> Guignard, M., and K. Spielberg, "A Realization of the State Enumeration Procedure", IBM Phila. Sci. C. Report 320-3025, June 1973

<5> Land, A.H., and A.G. Doig, "An Automatic Method for Solving Discrete Programming Problems", Econometrica, Vol. 28, 1960

<6> Spielberg, K., "A Minimal Inequality Branch-Bound Method", IBM Phila. Sci. C. Report 320-3024, June 1973

<7> Guignard, M., "Preferred Shadow Prices in 0-1 Programming", Res. Report, Dept. of Stat & OR, Wharton School, Univ. of PA, 1974

<8> Benders, J.F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, 1962

<9> Lemke, C.E., and K. Spielberg, "Direct Search Zero-One and Mixed-Integer Programming", J. ORSA, 1967

<10> Spielberg, K., "Minimal Preferred Variable Reduction for 0-1 Programming" , IBM Phila. Sci. C. Rep. 320-3013, 1972

<11> Zionts, S., "Generalized Implicit Enumeration using Bounds on Variables for Solving Linear Programs with Zero-One Variables", Naval Res. Logistics Quarterly, 1972

<12> Guignard, M., and K. Spielberg, "Reduction Methods for State Enumeration Integer Programming", Workshop on Integer Programming, Bonn 1975 (to appear in Discrete Mathematics)

<13> Guignard, M., and K. Spielberg, "Zero-One Zero-One Programming and Enumeration" , in "Nonlinear Programming Vol. 2", ed: O. Mangasarian, R.R. Meyer, S.M. Robinson , Acad. Press, 1975

<14> Guignard, M., and K. Spielberg, "Search Techniques with Adaptive Features for Certain Integer and Mixed-Integer Programming Problems", Proceedings of the V th IFIPS Conference, Edinburgh, 1968

<15> Guignard, M., "Contraintes Additionelles en Variables Bivalentes", Pub. Labo. Calcul, Univ. of Lille, Report #20, 1970

<16> Hammer, P.L., "Boolean Procedures for Bivalent Programming", Univ. of Waterloo, Dept. of Combinatorics and Optimization Reports, CORR #73-1, Oct. 1973 (with numerous references to earlier related work going back to the 60's, some of it jointly with authors such as S. Rudeanu and F. Granot)

<17> Balas, E., and R. Jeroslow, "Canonical Cuts of the Unit Hypercube", SIAM Journal of Applied Mathematics, Vol. 23 (1), July 1972 (work dating back to Carnegie Mellon University Report #198 of 1969)

<18> Balas, E., "An Additive Algorithm for Solving LP with Zero-One Variables", J. ORSA, Vol. 13, 1965

<19> Guignard, M., "Methodes Heuristiques de Resolution d'un Systeme d'Inegalites en Variables Entieres ou Bivalentes", Pub. Labo. de Calcul, Univ. of Lille, Report #32, 1972

<20> Bradley, G., "Equivalent Integer Programs and Canonical Problems", Management Science, 1971

<21> Hammer, P.L., Padberg, M.W., and U.N. Peled, "Constraint Pairing in Integer Programming", Univ. of Waterloo, Dept. of Combinatorics and Optimization Res. Reports, CORR #73-7, Aug. 1973

<22> Bouvier, B., and G. Messoumian, "Programmes Lineaires en Variables Bivalentes", Thesis, Faculte des Sciences, Univ. of Grenoble, 1965

MAINTENANCE SCHEDULING

MONIQUE GUIGNARD and K. SPIELBERG

(Stamford, U.S.A.)

1. INTRODUCTION

Maintenance scheduling problems have been formulated as mixed integer problems for quite a while (see ⟨1, 2⟩ for examples and for other references), but the practical solution of such problems is still not attempted widely, undoubtedly due to their large dimensionality and difficulty. Important exceptions appear to be in the utility industry, as suggested by references ⟨3, 4, 5⟩.

One may well wish to take a new look at the situation. Computers and production mixed integer codes are more powerful than ever, and there have also been some advances in the technique of solving mixed integer problems.

We report on work in two areas. They are both, to some extent, related to opportunities opening up with the new "extended control language" of IBM's MPSX-MIP/370 ⟨6, 7, 8⟩, which permits an unprecedented level of *user interaction* with a robust mixed integer programming system. Within such a framework, some large practical problems of special structure may become amenable to solution for the first time.

We first consider certain relaxations of the problem, such as "wraparound" or abandonment of integrality for the start of the maintenance period. We were encouraged to try the latter approach by the success of the work on investment planning reported in ⟨9⟩. It became apparent later that for our problem this approach amounts to an application of type 2 (S2) "special ordered sets", as described in ⟨10⟩. However, the absence of an *explicit* S2 feature in our system led us to use equivalent constraint formulations which may have their own utility.

Secondly, we considered the possibility of inserting an enumerative module into the mixed integer production code (via the extended control language of MPSX/370), possibly as a *heuristic aid*. In the course of this investigation, we found that our enumerative code, written and tested as an experimental program in APL, could handle rather large problems quite effectively, at least for the case of ready feasibility (problem not too constrained; this appears to be the most likely case in practice), or

for the case of problem infeasibility (which could arise often within a branch and code approach to the overall problem).

The enumeration module could be a heuristic within a larger production code treating a somewhat more complex practical situation. It might serve as an efficient tool for producing good feasible solutions (or solution candidates), while the production code would provide all the flexibility, generality and robustness of a well-designed large system.

2. SOME MAINTENANCE PROBLEMS

Consider a set of
machines i, $i=1, 2, \ldots, M$
which are to be removed from service (for maintenance) at the start of *one time period* t, $t=1, 2, \ldots, T$
over a time horizon of T periods.
Let a *(starting) variable* $x(i, t)=1(0)$
if and only if maintenance of machine i starts at the beginning of period t.

It will be understood that maintenance for a given machine i requires a given number (integral, unless stated otherwise) of contiguous time periods, denoted by $D(i)$.

We shall consider two scheduling situations:

2.1. Scheduling of Labor Requirements

Let $L(i)$ be a vector of length $D(i)$, specifying the labor requirement in each subunit of the maintenance period. If one denotes by $R(t)$ the vector of total labor requirements in periods t, $t=1, 2, \ldots, T$, then one may wish to consider the problems:

P1:
$$\min R$$
$$R \geqq R(t), \quad t=1, 2, \ldots, T$$
$$R(t)=\Sigma_i L(i) \cdot \xi(i, t), \quad t=1, 2, \ldots, T$$

where the *(removal) variable* $\xi(i, t)=1(0)$
if and only if machine i is removed from service in period t.

Note: (i, t) does not appear to be useful for the solution of the problem. The variable $x(i, t)$ is used instead and the constraints are enforced in terms of entries corresponding to the $L(i)$ in the constraint matrix (see example below).

P2:
$$\min \sum_{t=1}^{T} R(t) - \sum_{i=1}^{M} L(i) \cdot \xi(i, t)$$
$$\sum L(i) \cdot \xi(i, t) \leqq R(t) \quad \text{for all } t.$$

Note: We have used P2 only for given $R(t)$, in which case the objective function is a constant and the algorithm becomes one of a search for feasibility. Actually, this renders the problem difficult (dually degenerate) and the imposition of a more realistic objective function would appear to be helpful.

The constraints of problems P1 and P2 can be represented in terms of the $x(i, t)$, as seen from the following small.

Example: $M=5$, $T=8$

i	$D(i)$	$L(i)$
1	3	4 6 3
2	3	3 2 5
3	3	7 1 1
4	3	1 3 6
5	3	8 9 2

	$i=1$			$i=2$			$i=3$			$i=4$			$i=5$				
11	12	13	14	21	22	23	32	33	34	35	42	43	44	45	46	53	54	55	
1	1	1	1																$=1$
				1	1	1													$=1$
							1	1	1	1									$=1$
											1	1	1	1	1				$=1$
																1	1		$1=1$
4				3															$\leqq R1$
6	4			2	3		7				1								$\leqq R2$
3	6	4		5	2	3	1	7			3	1				8			$\leqq R3$
	3	6	4		5	2	1	1	7		6	3	1			9	8		$\leqq R4$
		3	6			5		1	1	7		6	3	1		2	9		$8\leqq R5$
			3						1	1			6	3	1		2		$9\leqq R6$
										1				6	3				$2\leqq R7$
															6				$\leqq R8$

2.2. Scheduling to Maximize Reserve

We now consider a problem for which the *output* of the machines is of paramount importance, such as in the utility industry. Let $a(i)$ stand for the *capacity* of machine i, and let the $L(i)$ be vectors of length $D(i)$ having equal coefficients $a(i)$. Let $R(t)$ then be an upper bound on the amount of capacity which may be removed in time period t.

The objective is to stay away from the upper bounds as much as possible, i.e. to leave the system with as much "reserve" as possible (in some sense).

Denote a constraint set in terms of the $x(i, t)$ as in the example but with all nonzero entries of the $L(i)$ equal to $a(i)$,

$$A \cdot x = b$$

or in structured form

$$M \cdot x = e$$

$$Q \cdot x = R$$

with M representing the multiple choice constraints, so that e is a vector of 1's.

Then the integer program becomes:

P3: max

$$M \cdot x = e$$

$$Q \cdot x + \lambda = R.$$

It is possible to restrict the problem further, in a number of meaningful ways. Certain of the $x(i, t)$ can be set to 0 (the corresponding starting points are "prohibited") by deletion of the column, others can be set to 1 (starting point "fixed"), again be deletion of the column and modification of the right hand side.

Less obviously, one may enforce *precedence* requirements of various types, or *conflict* restrictions which prohibit the simultaneous removal of machines belonging to prespecified classes of machines (e.g., the machines within a certain geographical configuration may constitute a conflict class).

For instance, one may avoid the concurrent outage of machines $i=1$ and $i=2$ by adjoining to the matrix $A=(M, Q)$ another matrix of the same size as Q, with coefficients 1 in the place of the $a(i)$ for $i=1$ and $i=2$, all other coefficients being 0 and the associated right hand side having coefficients of 1.

3. RELAXED VERSIONS OF THE PROBLEMS

Problem P2, for the data of the example is not feasible with values of the $R(t)$ uniformly $\leqq 13$. A first attempt at relaxing the problem might be to permit

- all starting points for machine such that completion is guaranteed within the time horizon T.

In addition to the above one might permit *wraparound*, i.e. one would allow

- all starting points within the time horizon, even if that requires the completion of the maintenance period at the "beginning" of the overall period, i.e. at $t=1$, etc. (the interpretation being that one permits maintenance to be finished "next year").

In the example at hand, one might consider all the columns listed below:

11	12	13	14	15	16	17	18
4						3	6
6	4						3
3	6	4					
	3	6	4				
		3	6	4			
			3	6	4		
				3	6	4	
					3	6	4

Even with wraparound, the example has no solution for $R(t) \leqq 13$.

More interesting is the consideration of *adjacent fractions* $x(i, t)$:

$$x(i, t) = \varepsilon$$
$$x(i, t+1) = 1 - \varepsilon$$

Interpretation:

Maintenance starts in period $t\bar{x}$, ε units from the end of the period. (e.g.: 5 day week, 52 weeks, $\varepsilon = .2$ . .service starts on Friday of week t).

We impose uniform values $R = R(1) = \ldots = R(t)$. Problem P2 then has many solutions with $R \geqq 8.125$. Two representative solutions are:

$$R = 8.5$$

$x(1, 4) = 1$, $x(2, 1) = 1$, $x(3, 3) = .45833$, $x(3, 4) = .54167$, $x(4, 4) = .25$, $x(4, 5) = .75$, $x(5, 7) = .5$, $x(5, 8) = .5$

$$R = 8.23867$$

$x(1, 6) = .47697$, $x(1, 7) = .52303$, $x(2, 1) = .14803$, $x(2, 8) = .85197$, $x(3, 6) = .63816$, $x(3, 7) = .36184$, $x(4, 1) = 1$, $x(5, 3) = .125$, $x(5, 4) = .875$ (wraparound)

3.2. Implementation in MPSX/370

We first implemented the adjacent fractional value formulation (S2 sets) by introducing a new set of integer variables $y(i, t)$, designating the $y(i, t)$ as special zero-one variables (see property below), by relaxing the $x(i, t)$ to be continuous, and by imposing the constraints:

Y-Formulation

$$x(i, 1) \leqq y(i, 1) \leqq [x(i, T)] + x(i, 1) + x(i, 2)$$
$$x(i, 2) \leqq y(i, 2) \leqq x(i, 1) + x(i, 2) + x(i, 3)$$
$$\ldots\ldots\ldots\ldots\ldots\ldots$$
$$x(i, T-1) \leqq y(i, T-1) \leqq x(i, T-2) + x(i, T-1) + x(i, T)$$
$$x(i, T) \leqq y(i, T) \leqq x(i, T-1) + x(i, T) + [x(i, 1)]$$

The square brackets are applicable to the case with wraparound.

Property. The $y(i, t)$ have exactly two 1's in adjacent positions.

Example. (1, 1, 0, . . ., 0), (0, 1, 1, 0, . . ., 0), . . ., (0, . . ., 0, 1, 1).

Also: The constraints have the (advantageous) form typical of "weakly linked" problems (see ⟨11⟩), or of "generalized variable upper bounds" ⟨12⟩.

After developing the y-formulation, we found the following, simpler z-formulation in ⟨1⟩.

Z-Formulation

$$x(i, 1) \leqq [z(i, T)] + z(i, 1)$$
$$x(i, 2) \leqq z(i, 1) + z(i, 2)$$
$$\ldots\ldots\ldots\ldots$$
$$x(i, T) \leqq z(i, T-1) + z(i, T)$$
$$\Sigma_t z(i, t) = 1 \quad \text{for all } t \text{ (multiple choice)}$$

$x(i, t)$.. continuous, $z(i, t)$..(0, 1) integer

We used both formulations with MPSX-MIP/370, and found them of similar utility for relatively small problems.

3.3. A small Sample Problem

Consider a problem P3 for maximizing reserve, with the data:

$M = 4$, $T = 8$

$D(i) = 2 \quad 4 \quad 1 \quad 2$

$A(i) = 200 \quad 200 \quad 90 \quad 300$

$R(t) = 541 \quad 225 \quad 514 \quad 511 \quad 234 \quad 483 \quad 386 \quad 495$

In Fig. 1 we give the machine output (of an experimental APL program; see next section). Its value (maximal value for minimal reserve) is 34.

BEST POSS. SLACK, BEST SLACK, CURR. SLACK 34 34 34
SCHEDULED NO. OF MACHINES IS 4 OUT OF 4
SCHEDULE, OUTAGE, RESERVE-SLACK, RESERVE

	0	507	541
	0	191	225
* O	400	80	114
* O	400	77	111
O	200	0	[34]
O	200	249	283
□	300	52	86
▽ □	390	61	95

LAMBDA=[34]

The printout gives a plot of the schedules for the 4 machines, with the time-axis pointing downward and the machine-axis pointing to the right. For example, the second column represents the outage of machine 2 in the periods 3, 4, 5 and 6, with its capacity of 200 removed in each of these periods.

The first numerical column represents the sum of the capacities removed for the printed schedule. The second numerical column is the difference between the $R(t)$ and the first column. The last column, finally, gives the remaining reserves for each t, the minimal value corresponding to the objective function of the problem.

It is clear that this problem has many degrees of freedom. One could find many alternative, equivalent solutions. The defaults of the program have been set such as to move the schedules as much as possible to the end of the time-horizon, and this is reflected in the fact that the printed solution has the first two time periods free (which may be an advantage: i.e., the slackeness of the system can be exploited for secondary benefits).

In Fig. 2 we give an optimal schedule corresponding to relaxation via wraparound. It can be seen that the schedules for the first two machines occupy the first and last periods of the overall interval. The objective function value is drastically increased. It is, of course, inadvisable to extrapolate such a favorable result to larger problems. But for realistic problems, even small improvements in the minimal reserve may be significant.

```
BEST POSS. SLACK, BEST SLACK, CURR. SLACK 186 85 85
SCHEDULED NO. OF MACHINES IS 4 OUT OF 4
SCHEDULE, OUTAGE, RESERVE-SLACK, RESERVE

 * O        400   56  141
            0  140  225
      □     300  129  214
      □     300  126  211
            0  149  234
   O ▽      290  108  193
   O        200  101  186
 * O        400    0  [85]

LAMBDA=[85]
```

Finally, we present the results for integrality relaxations, obtained with MPSX-MIP/370, in Fig. 3. There is another substantial improvement in objective function value. It may well justify the inconvenience of having to go to split-schedules (represented by the ϵ).

FRACTIONAL (MPSX-MIP/370)

K	1	2	3	4	R	Σ	λ SLACK
$D(K)$	2	4	1	2			
$A(K)$	200	200	90	300			
1					541	200	341
2					[225]	.356×300=107	[118]
3					514	300+200×.42=384	130
4					511	200+300×.644=393	[118]
5					[234]	200×.58=116	[118]
6					483	200+90=290	193
7					386	200	186
8					495	200	295
ϵ	.42..	—	—	.356			
$1-\epsilon$	.58..	—	—	.644		$\lambda_{max}=118$	

4. AN ENUMERATIVE SCHEDULING SCHEME

A maintenance scheduling problem such as P3, and more complex variants of it, can today be solved with general purpose mixed-integer codes. An industrial user is probably well advised to go that route, to avoid the necessity of redesigning his code whenever the model changes. But the problems become rather large. A 40 machine, 52 time-period problem expands into an integer program with 2081 integer variables and more than 100 constraints. We have therefore developed an enumerative procedure which handles moderately sized problems very well and could be used as an effective heuristic aid (built into an all-purpose code as an add-on; e.g., into MPSX-MIP/370 via the extended control language) for larger models.

4.1. The Enumeration

We have an experimental APL enumeration program (not available outside of IBM) with the following features:

1. All constraints are treated *implicitly*, i.e. are generated as needed from the data: $M, T, A(i), D(i), R(t)$, and a P by M *conflict matrix* which represents P conflicts, one per row.
2. The method is a modified additive algorithm (⟨13⟩, also ⟨14⟩), starting at the 0 origin and scheduling outages one by one. The machines are either preordered

absolutely in terms of descending $A(i).D(i)$, or ordered in terms of conflict classes, the above ordering being respected within each class. A dynamic ordering could be imposed in the sense of *minimal preferred variable* inequalities ⟨15⟩, but has not been found necessary so far.

3. The book-keeping is implemented through binary matrices of sizes M by T, and therefore is modest in terms of storage requirement.

4. A row of the conflict matrix has entries of 1 for all machines in potential conflict with each other (the conflicts are assumed to impose a partitioning; this could easily be relaxed). If machine k', of maintenance duration d' has been scheduled ("fixed") to start maintenance at time t' in the search, and another machine (k, d, t) is considered subsequently (at larger "levels") of the search, then an entire string of starting points is prohibited for machine k (over successor nodes of the enumeration tree).
The prohibited string is determined by (wraparound not being considered here):

starting point: $ts0 = t' + 1 - d$
$ts = \max\{1, ts0\}$, and

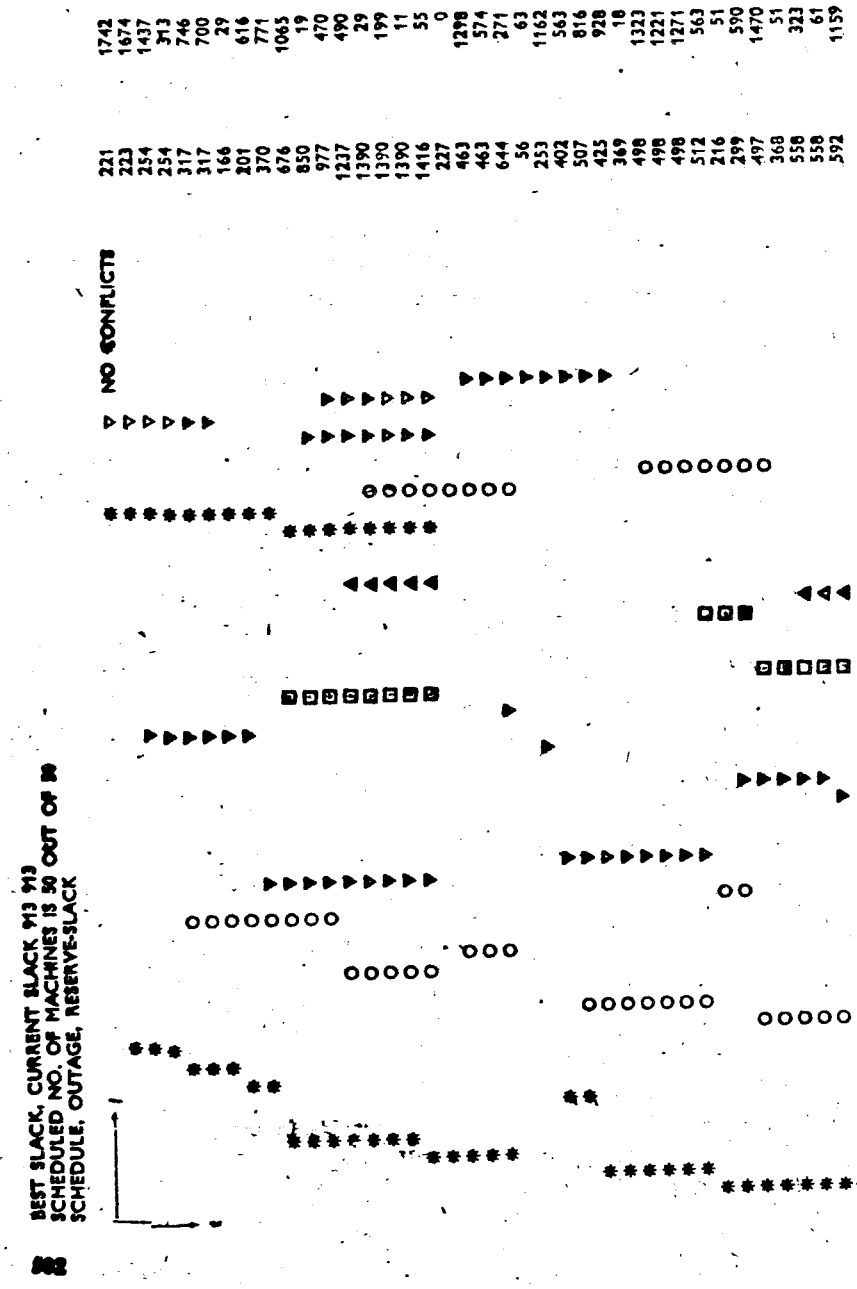
length: $L = d' + d - 2$ if $ts0 > 0$
$= (d' + d - 2) - (1 - ts0)$ if $ts0 \leq 0$.

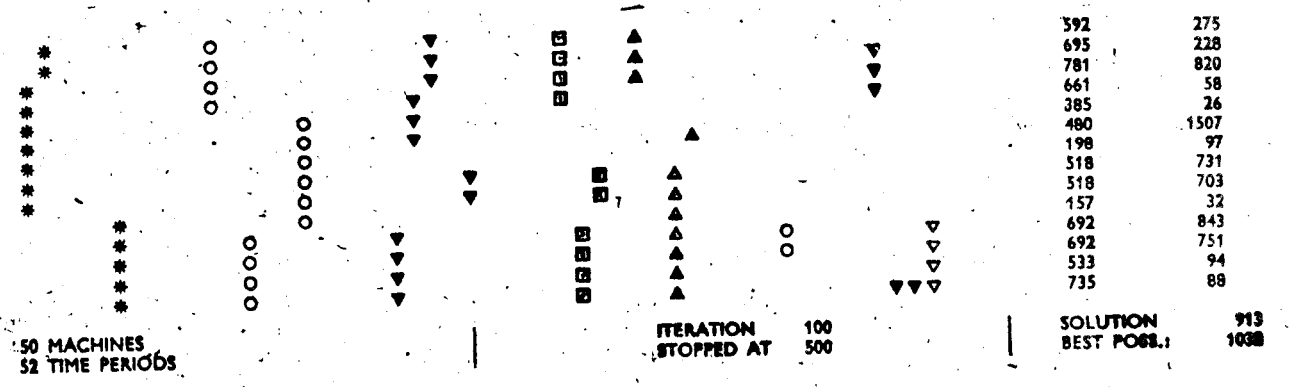
5. Upper bounds on λ are easily computable and may be used to stop the search when one has an incumbent solution close enough to the bound.

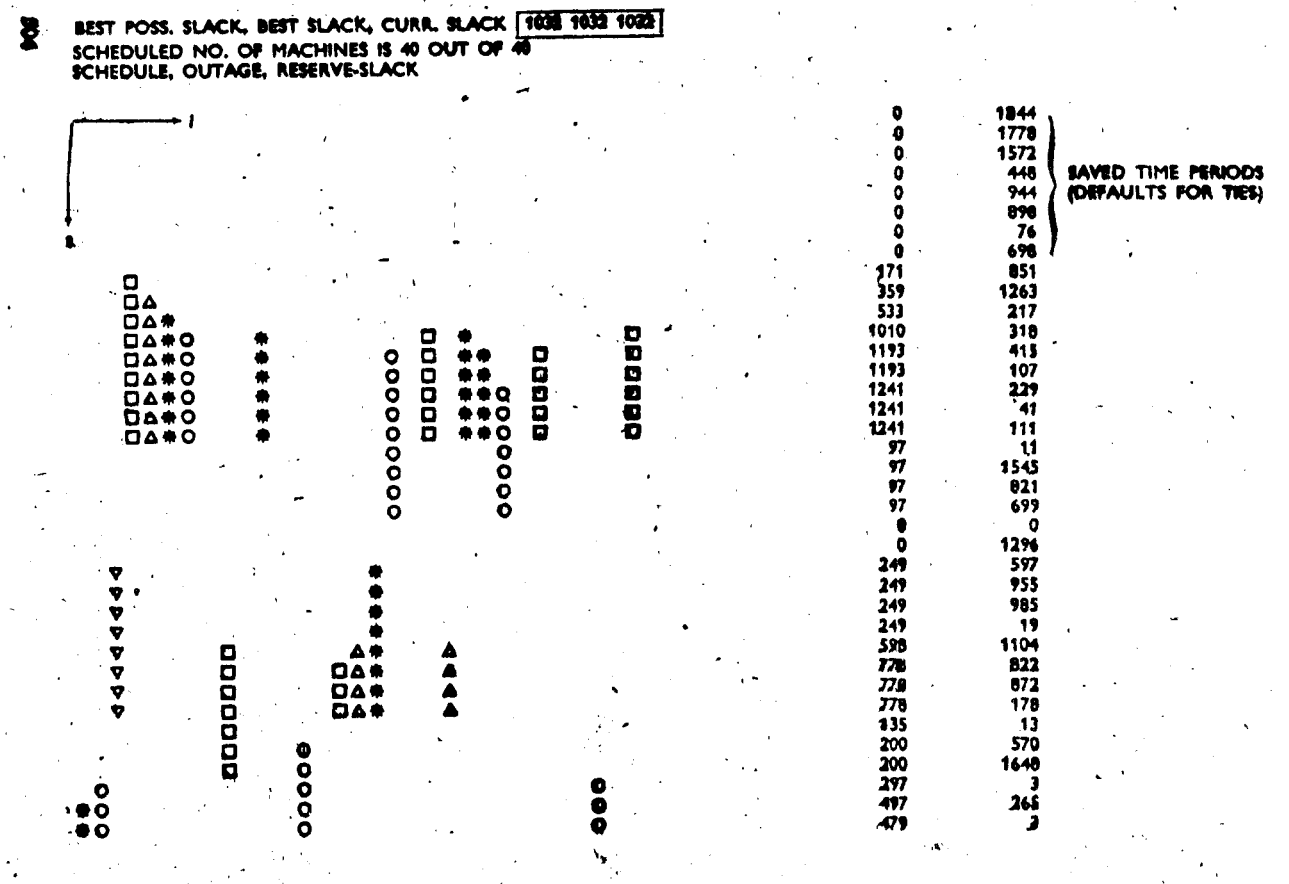
4.2. Results for Some Large Problems

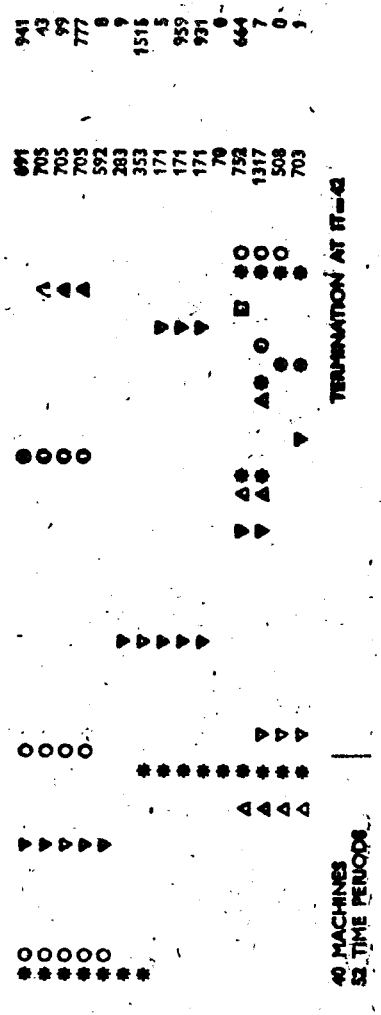
We give two experimental results. In the first case we consider 50 machines partitioned into 6 conflict classes plusone non-conflict class (indicated in the plot by changing character representation from class to class). It can be seen that the problem is quite tightly constrained by virtue of the conflicts. In the first conflict all but three time periods must be used if there is to be no conflict. As a consequence, the search for feasibility and then optimality is not too easy, and optimality was not attained after about 5 minutes (on an APL time sharing system). But there were many solutions and the plotted schedule is within 119 units of optimality.

In the second plot we have a 40 machine, 52 time period problem with no conflicts, it is clearly not very tightly constrained, and the depicted solution is found and attested optimal within a few minutes. It can again be seen, that the defaults of the system tend to create a schedule which leaves initial periods unused. In this case the entire maintenance requirement could be compressed into 44 time periods only.




50 MACHINES
52 TIME PERIODS
ITERATION 100
STOPPED AT 500
392 275
695 228
781 820
661 58
385 26
480 1507
198 97
518 731
518 703
157 32
692 843
692 751
533 94
735 88
SOLUTION 913
BEST POSS.: 1038


BEST POSS. SLACK, BEST SLACK, CURR. SLACK 1038 1032 1022
SCHEDULED NO. OF MACHINES IS 40 OUT OF 40
SCHEDULE, OUTAGE, RESERVE-SLACK
0 1844
0 1778
0 1572
0 448
0 944
0 898
0 76
0 698
171 851
359 1263
533 217
1010 318
1193 413
1193 107
1241 229
1241 41
1241 111
97 11
97 1545
97 821
97 699
0 0
0 1296
249 597
249 955
249 985
249 19
598 1104
778 822
778 872
778 178
835 13
200 570
200 1648
297 3
497 266
479 3
SAVED TIME PERIODS
(DEFAULTS FOR TIES)



REFERENCES

[1] Wagner, H. M.: *Principles of Operations Research*, Prentice Hall, Englewood Cliffs, N. J., 1969.

[2] Wagner, H. M., Giglio, R. J., Glaser, R. G.: Preventive Maintenance Scheduling by Math. Programming. *Mgmt. Sc.*, 10 (1964), 2, Jan.

[3] Kingston, P. L., Lipton, S. L., Stojka, J.: Mixed Integer Programming, Models for Generator Maitenance Scheduling, Working Paper, Niagara Mohawk Power Co. (to be presented at the ORSA/TIMS Meeting, Miami, No. 1976).

[4] Dopaz, J. F., Merrill, H. M.: Optimal Generator Maintenance Scheduling using Integer Programming, IEEE Trans. Power Appl. & Systems, Vol PAS-94, No. 5, Sept. 1975.

[5] Merrill, H. M.: Power Plant Maintenance Scheduling with Integer Programming, IEEE Tutorial Course, 76 CH1107-2-PWR, 1976.

[6] IBM Math. Progr. System Extended/370, (MPSX/370), Program Ref. Manual SH19-1095, Mixed Integer Progr./370 (MIP/370), Program Ref. Manual SH19-1099, Dec. 1974.

[7] Benichou, M., Gauthier, J. M., Hentges, G., Ribiere, G.: The Efficient Solution of Large Scale Linear-Programming Problems. Some Algorithmic Techniques and Computational Results., 9th Int. Symp. on Math. Progr., Budapest, 1976.

[8] Slate, L., Spielberg, K.: The Extended Control Language of MPSX/370 (and possible applications), IBM Scientific Marketing, White Plains, N. Y., 1976.

[9] Chen, R., Crowder, H., Johnson, E. L.: An Integer Programming Formulation of the Installment Scheduling Problem, IBM Th. J. Watson Research Center, Yorktown Heights, N. Y., 1976.

[10] Beale, E. M. L., Tomlin, J. A.: Special Facilities in a General Math. Progr. System for Nonconvex Problems using Ordered Sets of Variables, *Proc. 5th Int. Conf. on Operational Research* (ed. J. Lawrence), Tavistock Publ., London 1970.

[11] Balinski, M. L., Spielberg, K.: *Methods for Integer Programming: Algebraic, Combinatorial and Enumerative*, in *Progr. in OR*, Vol III (ed. J. Aronofsky), J. Wiley & Sons, N. Y. 1969.

[12] Schrage, L.: Implicit Representation of Generalized Upper Bunds in Linear Programs, Report 7543, Dept. Econ. Grad. School of Bus., Univ. of Chicago, Oct. 1975.

[13] Balas, E.: An Additive Algorithm for Solving Linear Programs with Zero-One Variables, *J. ORSA*, 13 (1965).

[14] Lemke, C. E., Spielberg, K.: Direct Search Zero-One and Mixed Integer Programming, *J. ORSA*, 15 (1967).

[15] Guignard, M., Spielberg, K.: The State Enumeration Method for Mixed Zero-One Programming, IBM Phil. Sc. Center Report, #320 3024, 1971.

Annals of Discrete Mathematics 1 (1977) 273–285

REDUCTION METHODS FOR STATE ENUMERATION INTEGER PROGRAMMING

Monique GUIGNARD

Department of Statistics, Wharton School, University of Pennsylvania, Philadelphia, PA 19174, U.S.A.

Kurt SPIELBERG

Scientific Marketing, IBM, White Plains, NY 10604, U.S.A.

Integer programs with small bound intervals can often be dealt with effectively, by a state-enumeration procedure with reduction methods. Our approach features the consistent use of logical inequalities, derived during the computation, especially for influencing the choice of appropriate directions for the search effort.

1. Introduction

In spite of much good research and a host of proposed algorithms, the all-integer program P

$$\begin{aligned} &\min\ c \cdot y = z \\ &\quad C \cdot y \leqslant b \qquad (1.1) \\ &\quad y(j) \text{ in } [L(j), U(j)], \quad j = 1, 2, \ldots, n \\ &\quad y(j) \cdots \text{integer}, \end{aligned}$$

is far from being solved successfully even for small problems.

Both branch-and-bound (BB) programming (see [4] for a recent survey) and enumerative programming meet success for some problems (usually those with which the analyst is familiar) and fail badly elsewhere.

There is a need for a flexible integer programming system, possibly with user intervention on some kind of interactive level. In this paper we discuss an experimental enumerative system which is meant to incorporate a family of techniques which we have shown, or which we believe, to have substantial promise.

Among the practical problems which may require such techniques, we cite large scale integer problems with substantial logical structure. Many of these are scheduling problems with time-dependent (0, 1) decision variables, say $y(i, t)$. (E.g., $y(i, t)$ might be 1 (0) if a certain choice is made (not made) in time period t.)

A production code of the BB type (such as MPSX/MIP of IBM) takes a good deal

of time solving linear programs. A promising alternative, then, is to solve only one linear program (at level 0 of the search), or conceivably a linear program whenever the search returns to level 0, and to finish by state enumeration.

MPSX/MIP 370 has a control language which would allow the writing of an enumeration program (using procedures of MPSX) in PL/I. Knowledge about special structure can probably be incorporated best in such an enumerative code.

Another area of interest for the techniques of this paper is to be found *within* a production mixed-integer code, especially for problems with a large number of continuous variables. The reduction and state enumeration procedures over Benders inequalities would be executed entirely in core storage and would consume negligible effort compared to the other I/O-bound solution procedures.

2. State enumeration

2.1 Scheme of search

The search is organized as follows. It starts at "level $l = 0$" and "node $\nu = 1$", with all components "free", i.e., with all components $y(i)$ only constrained by the initial bounds $L(j)^1$ and $U(j)^1$.

At a general iteration, one is at *level* l which measures the number of explicit bound changes ("forward branches") which have been imposed from the last time the search was at level 0.

At level l (and node ν; ν is a running counter, increased by one at each iteration), one basically takes one of two actions:

(i) A *Forward Step* from level l to level $l + 1$ (setting the bound of a branch variable to a new value).

(ii) A *Backward Step* from level l to level $l - 1$ (with the search terminating when $l - 1$ is -1).

The explicit forward steps from level 0 to level l are recorded in two lists of l numbers:

List 1 consists of signed component indices, the sign of an index reserved for indicating whether the associated variable was constrained by a raising (lowering) of its lower (upper) bound.

List 2 contains the value to which the upper (lower) bound of the related component from list 1 is to be lowered (raised) on *return* to a level.

It is clear that such a scheme suffices to record the history of the search and to control the search on backward steps. Further details can be skipped.

2.2 The state

At node ν (level l) one easily computes a set of "Working Bounds", $(L(j), U(j))^\nu$, i.e., a set of bounds determined by the explicit branches of the search,

as well as by subsequent applications of the reduction procedures. (We shall usually drop the superscript ν on the working bounds.)

The *State* S^ν is essentially meant to be a *conjectured value* y^ν, for which we permit (to keep the search simple) the choice of setting a given $y^\nu(j)$ either to its lower bound $L(j)$ or to its upper bound $U(j)$.

It is the state value y^ν which is substituted as a trial solution. Forward branches are taken so as to lead away from the state, and are chosen among some index set J (see Section 4.2) so as to reduce total infeasibility.

In the numerical experimentation we determined an initial state at node $\nu = 1$ from the initial LP solution y^λ:

$$\begin{aligned} y^1(j) &= L(j) \quad \text{if} \quad y^\lambda(j) \leqslant L(j) + r \cdot (U(j) - L(j)) \\ &= U(j) \quad \text{if} \quad y^\lambda(j) > L(j) + r \cdot (U(j) - L(j)) \end{aligned} \tag{2.1}$$

(r being an arbitrary rounding parameter). At subsequenct levels the state is carried along, i.e. one uses the transformed state given by:

$$\begin{aligned} y^\nu(j) &= L(j) \quad \text{if} \quad y^1(j) \quad \text{was} \quad L(j)^1 \\ &= U(j) \quad \text{if} \quad y^1(j) \quad \text{was} \quad U(j)^1. \end{aligned}$$

Alternatively, we also considered the options of setting $y^\nu(j)$ always to the lower working bound (in the tables indicated by "ALWL") or always to the upper working bound (in the tables: "ALWU").

3. Techniques for integer state enumeration

No one technique can be expected to solve all problems. A modular collection of techniques, possibly controlled in an interactive fashion, may eventually prove to be the best vehicle for studying and resolving general and special integer programs.

In Section 3.1 below, we outline those techniques which will be stressed in this paper, and for which some numerical results will be given. In Section 3.2 we outline other methods which we have tried and for which results have been given elsewhere. Some of these methods need to be generalized from the 0–1 to the integer programming case.

The results of this paper demonstrate (for small problems; but we believe that there is no reason to assume drastically different behavior for larger problems) the importance of *state enumeration* (good starting points for the enumeration) and of some form of *reduction* (i.e., systematic tightening of bounds).

The generation of logical ("preferred") inequalities does not yield, in these experiments, much additional improvement. We believe that this shows the necessity of combining such techniques with the use of penalties and propagation (see Section 3.2, items 7 and 8).

3.1 Techniques used in current experimentation

(1) *Solution of only one linear program* at the start (possibly followed by judicious use of cutting planes to get optimal but non-integral tableaux with a relatively good value for z, i.e. a large value for the objective function of the relaxed problem).

(2) *Retention of the top row* (or the related Benders inequality, [2]) for purposes of bound reduction or fixing of variables. For some purposes one may wish to retain the entire tableau. Suitably updated they are referred to as "*current*" top row or tableau.

(3) Definition and use of a *state*, i.e., a suitable origin for the search (see [8] or [9]) (either *permanently* after solution of the initial LP, or *dynamically* according to some (heuristic) criterion at each *level* of the search).

(4) "*Reduction*" *of system* (*1.1*) at *every level* of the search.

(a) Reduction of the *bound intervals* for the $y(j)$ and the slacks $s(i)$ (as proposed by Zionts [17]).

(b) *Construction of logical relations* ("minimal preferred inequalities") which are to guide the search so as to: (i) find feasible solutions, (ii) "minimize" the search effort.

Each preferred inequality specifies d (degree) *preferred* or *indicated* bound changes. One indicated bound change, at least, must be implemented if the problem is to have a solution.

The main emphasis is on "*contraction*", i.e., on guiding the search into (locally) increasingly constrained directions [6, 14, 16].

(5) *Local search* of lattice points close to a given point y. A simple procedure for looking at all points which differ from y in exactly "lev" (level: 1-level, 2-level search) components by exactly one unit. The search is also used as a strategic device, to select branches for getting to new points with decreased overall infeasibility. As can be seen, there is some overlap and conflict between (4) and (5) (see also Section 5).

3.2 Techniques to be incorporated in a full system

(6) *Cutting plane* techniques. Our experimental system includes the ability of adding cuts, followed by reoptimization (see [5, 10, 11]). The (0, 1) test problems of this paper can be solved by such cutting plane methods, with only little enumeration. More difficult problems (with larger gap between LP and IP objective function) may prove intractable.

(7) *Penalties* and preferred variable inequalities. Preferred variable inequalities (as this paper shows in conjunction with [7, 15, 16]) are best invoked together with penalties. One rules out certain branches of a suitable preferred inequality due to large associated penalties and pursues alternatives when they are favorable from a "contraction" point of view.

(8) *Propagation.* An "indicated branch (bound change)" of a minimal preferred inequality (see Section 4.2) may often be implemented by "propagation", tantamount to fixing a variable (or altering a bound) at its current values, i.e. at insignificant computational cost. See [15] for some excellent computational results.

(9) *Mixed* integer problems. For a mixed problem, one may work with Benders inequalities generated during the search. Techniques (1)–(8) can then be applied to a system of Benders inequalities in the integer variables, which in our experience lend themselves well to reduction.

4. Reduction for integer variables

4.1 *Reduction of bound intervals* [17]

Consider the constraint set of problem (1.1) in equality form:

$$A \cdot t = b \tag{4.1}$$
$$L(j) \leqslant t(j) \leqslant U(j), \quad j = 1, 2, \ldots, n+m$$

with t a composite of the structural variables $y(j)$ $(j = 1, 2, \ldots, n)$ and the slacks $s(i)$ $(i = 1, 2, \ldots, m)$.

From (4.1) a new set of bounds can be computed in accordance with the formulas:

$$L(j)' = \max \begin{cases} L(j);\ U(j) + (b(i)/a(i,j)) - (1/a(i,j))(\mathrm{APU}(i) + \mathrm{AML}(i)) \\ \quad \text{for } i\text{: } a(i,j) > 0, \\ U(j) + (b(i)/a(i,j)) - (1/a(i,j))(\mathrm{APL}(i) + \mathrm{AMU}(i)) \\ \quad \text{for } i\text{: } a(i,j) < 0; \end{cases}$$

$$U(j)' = \min \begin{cases} U(j);\ L(j) + (b(i)/a(i,j)) - (1/a(i,j))(\mathrm{APL}(i) + \mathrm{AMU}(i)) \\ \quad \text{for } i\text{: } a(i,j) > 0, \\ L(j) + (b(i)/a(i,j)) - (1/a(i,j))(\mathrm{APU}(i) + \mathrm{AML}(i)) \\ \quad \text{for } i\text{: } a(i,j) < 0; \end{cases}$$

$$j = 1, 2, \ldots, n+m. \tag{4.2}$$

$$\begin{aligned} \mathrm{APU}(i) &= \sum a^{+}(i,j) \cdot U(j) \\ \mathrm{APL}(i) &= \sum a^{+}(i,j) \cdot L(j) \\ \mathrm{AMU}(i) &= \sum a^{-}(i,j) \cdot U(j) \\ \mathrm{AML}(i) &= \sum a^{-}(i,j) \cdot L(j). \end{aligned} \tag{4.3}$$

Summation is from 1 to $n+m$,

$$\begin{aligned} a^{+}(i,j) &= \max(0, a(i,j)), \\ a^{-}(i,j) &= \min(0, a(i,j)). \end{aligned} \tag{4.4}$$

These formulas are easily derived. They are slightly altered from some of the formulas found in [17].

At any node of the enumeration one applies (4.2) iteratively until there is no more alteration of bounds. It appears to us (after some experimentation) that this procedure is somewhat preferable to the equivalent one of Section 4.2 below (which gives the same results by iterative application of minimal inequalities of degree 1).

In the numerical experiments which we conducted, little use was made of the resulting bounds on the slacks. They could be exploited, for example, in Section 4.2.

4.2. Logical inequalities for integer variables

(i) *The* $(0,1)$ *case* [1, 6, 14, 16]. Let (1.1) be a system in zero-one variables. One can then associate with it a "minimal preferred variable" system

$$Q \cdot y \leqslant q \tag{4.5}$$

of degree d. (The unusual case that (4.5) is empty, i.e. that (1.1) does not imply any logical relations for the $y(i)$, can be taken care of by simple default procedures. In the following it is always assumed that (4.5) is not empty.)

Each row k of Q has d non-zero entries and row k of (4.5) represents one logical condition implied by the system and the zero-one conditions. Let $q(k,j)$ be the entries in row k of Q.

$q(k,j) = -1$ $(+1)$ implies that the possibility $y(j) = 1$ (respectively $y(j) = 0$) should be considered as a logical alternative (i.e., as *preferred* or *indicated* value) in an either-or partitioning. E.g., $d = 3$, and $q(k,j1) = -1$, $q(k,j2) = 1$, $q(k,j3) = -1$, $q(k) = 0$, represents the logical implication:

$$\text{either } y(j1) = 1, \text{ or } y(j2) = 0, \text{ or } y(j3) = 1.$$

(ii) *The integer case*

(a) *Reduction.* Starting with (1.1), one multiplies the columns of C by the bound intervals $U(j)$–$L(j)$, and correspondingly subtracts $\sum c(i,j) \cdot L(j)$ from $b(i)$, for each i. In this fashion one effectively changes to a system with variables $t(j)$ in the unit hypercube, i.e. to

$$\sum d(i,j) \cdot t(j) \leqslant b'(i), \tag{4.6}$$
$$t(j) = (y(j) - L(j))/(U(j) - L(j)),$$
$$d(i,j) = c(i,j) \cdot (U(j) - L(j)),$$
$$b'(i) = b(i) - \sum L(j) \cdot c(i,j),$$
$$0 \leqslant t(j) \leqslant 1.$$

The procedures for generating all minimal preferred inequalities for zero-one variables [16] are then applied to (4.6) as if (4.6) were a system in (0, 1) variables.

When the $t(j)$ are true zero-one variables, then an inequality such as

$$-2 \cdot t(j1) - 3 \cdot \bar{t}(j2) \geqslant -1$$

(with $\bar{t}(j) = 1 - t(j)$) is interpreted as

$$t(j1) + \bar{t}(j2) \geqslant 1,$$

i.e., either $t(j1) = 1$, or $t(j2) = 0$.

In the case of (4.6), however, all one can imply is that either the lower bound of $y(j1)$ must be raised, or the upper bound of $y(j2)$ must be lowered.

The minimal preferred inequalities (obtained by exactly the same procedures as for zero-one variables) are best written as:

$$\sum_{\pi} \tilde{t}(j) > 0,$$

$$\tilde{t}(j) = t(j) \quad \text{or} \quad \tilde{t}(j) = 1 - t(j). \tag{4.7}$$

The degree d is equal to $|\pi|$, the cardinality of the preferred set under consideration.

It should be noted that a partitioning relation such as (4.7) is usually much stronger than conventional branch-bound dichotomies. In a branch-bound code, the partitioning of the above example, for instance, could be exploited by the successive solution of the two problems P[$t(j1) = 1$] and P[$t(j2) = 0$ *and* $t(j1) = 0$]. Clearly, analogous conjunctive conditions are imposed in integer branch-bound programming. See [15, revised] for some details.

In enumerative programming, such conjunctive conditions are taken care of automatically by the book-keeping.

One may summarize the situation more generally. Let

$$v(j) = L(j) \quad \text{if} \quad \tilde{t}(j) = t(j)$$
$$U(j) \quad \text{if} \quad \tilde{t}(j) = 1 - t(j), \quad \text{all} \quad j \in \pi.$$

Theorem 1. *In order that y be an integer solution of* (1.1), *it is necessary that* $I(\pi) \neq 0$, *i.e. that* $y(\pi) \neq v(\pi)$.

Let $\tilde{y}(j) = y(j) - L(j)$ (resp. $U(j) - y(j)$) if $\tilde{t}(j) = t(j)$ (resp. $1 - t(j)$).

Corollary 1. *y can be an integer solution of* (1.1) *only if* $\tilde{y}(\pi) \neq 0$.

Corollary 2. *If $\pi = \emptyset$, the condition is vacuous. If $d = 1$, one may reduce the bound interval of $y(\pi)$ by* 1. *If $d > 1$, one may reduce one of the d bound intervals of $y(\pi)$ by* 1.

When the reduction procedures of this section are preceded by those of Section 4.1, however, one assures that $d \neq 1$.

All techniques based on contraction, penalties, propagation, etc., clearly remain valid in some modified form.

(b) *Implementation.* In the experimentation for this paper we only implemented simple procedures based on contraction, as are described in what follows.

The entries of Q give information about the effect of an enumeration branch from a node ν to its successor node $\nu + 1$. Let

$$m1(j) = \#\{q(k,j) \mid k : q(k,j) < 0\}$$
$$m2(j) = \#\{q(k,j) \mid k : q(k,j) > 0\}. \tag{4.8}$$

It is clear that:

$m1(j) > 0$ implies $d^{\nu+1} < d^{\nu}$ if one branches with $y(j) = 0$,

$m2(j) > 0$ implies $d^{\nu+1} < d^{\nu}$ if one branches with $y(j) = 1$.

Such branches are called *contracting* branches, for they lead the search to a successor point in the integer lattice at which the problem is more constrained than before the branch.

The most favorable case is that of *double-contraction*, which arises for branch j when:

$$m1(j) > 0 \quad \textit{and} \quad m2(j) > 0.$$

The enumerative code described here isolates a set, J, of candidates for branching according to the priorities:

(i) $J = \{j \mid m1(j) > 0 \text{ and } m2(j) > 0\} \cdots$ double contraction,
(ii) $J = \{j^* \mid m1(j^*) \text{ or } m2(j^*) > 1 \text{ and equal to } (\max_j (m1(j), m2(j))\}$ (4.9)
(iii) $J = \{\text{all } j : L(j) \neq U(j)\} \cdots$ "free" variables.

In all cases, a branch with variable j is chosen such that $y^{\nu+1}(j) = 1$ (0) if $y^{\nu}(j) = 0$ (1).

As explained in [16], this requirement may necessitate a replacement of (4.5) (which represents "free" reduction with no state imposed) by a reduction after imposition of a state on (1.1), if the original preferred inequality system has no row for which all indicated branches lead away from the state.

Whether a procedure which ensures branching *away* from the state is indeed desirable, is not entirely clear. Some of our results in [10] seem to go against such a conjecture.

Our experimental system has been designed to admit the use of a truly "*dynamic state*", i.e. a state which can be recomputed (most likely so as to satisfy as closely as possible, in some sense, a set of minimal preferred inequalities) at each iteration. Such a feature, however, has not yet been tested.

5. Experiments

5.1 The experimental algorithm

The experimental algorithm was that of Section 2 with the techniques of Sections 3 and 4, stressed as follows:
one linear program was resolved and retained as per Sections 3.1 and 3.2;
the reduction procedures of Section 3.4 were used after any bound change, for whatever reason (possibly leading to some redundant work), in the order a and b;
states were determined as per Section 2.2 (see below for details);
a local search was conducted at every iteration (see Section 3.5), except for LEV = 0 (see below);
forward branches in that enumeration were chosen so as to lead to a successor with improved feasibility. Preferred inequalities were invoked for branching only if no improved feasibility was attained in the local search.

5.2 Experimental results

In Table 1 we describe a few test problems for which experimental results are described in Tables 2, 3, 4 and 5. The LP (Linear Programming) Objective Functions given are those obtained after an initial preprocessing reduction phase.

Table 1. Description of test problems

(m, n) Source	LP solutions		Integer solutions	
	in (0, 1)	in (0, 2)	in (0, 1)	in (0, 1, 2)
(1) 6,12* [3]	6.85	6.58	13	11
(2) 10,20 [3]	−6155.3	−6623.5	−6120	−6570
(3) 28,35 [12]	521.05	356.23	550	400
(4) 12,44 [12]	56.68	56.68	73	—
(5) 5,39 [3]	−10672.	−10737.	−10620	—
(6) 20,28*,b	31.34	27.71	47	—

* Problem differs slightly from source problem.
b Seems to have originated as test problem in IBM Paris.

The problems are small. They are from either [3] or [12] (with some coefficients possibly altered by transcription errors), and are for the most part easily resolved as (0, 1) problems. However, we solved some of them also as (0, 1, 2) problems. Our experimental work was on an APL system time-shared with some 100 users. Hence even small problems require substantial on-line time, and in a sense our environment was not much different from that of a user with larger problems and greater computing power via a dedicated machine.

Table 2. Problem (6,12)

	BND	STA	LEV	NR	R1	Q1	Q2
(1)	1	LP, .5	1	999 (155)	3 5	3 5	3 5
(2)	1	LP, .5	2	13 (140)	2 3	2 3	2 3
(3)	2	LP, .5	1	···	15 21	17 23	—
(4)	2	LP, .5	2	···	—	4 11	—
(5)	1	ALWL	1	···	4 7	4 7	4 7
(6)	1	ALWL	2	···	—	4 7	—
(7)	2	ALWL	1	···	8 13	8 13	—
(8)	2	ALWL	2	···	—	—	—
(9)	1	ALWU	1	···	15 17	12 13	—
(10)	1	ALWU	2	···	11 15	11 15	—
(11)	2	ALWU	1	···	44 53	38 49	38 49
(12)	2	ALWU	2	···	41 51	34 45	—

— stands for "not run".
··· stands for "not run, believed to be a bad strategy".

Table 3. Problem (10,20)

	BND	STA	LEV	NR	R1	Q1	Q2
(1)	1	LP, .5	1	···	57 57	51 51	1 11
(2)	1	LP, .5	2	−6100 (225)	6 9	6 9	7 9
(3)	2	LP, 1	1	···	1 21	1 19	—
(4)	2	LP, 1	2	−6570 (125)	1 25	1 23	1 19
(5)	1	ALWL	1	···	···	···	···
(6)	1	ALWL	2	···	−4158 (60)	−3960 (62)	···
(7)	2	ALWL	1	···	···	−3960 (140)	···
(8)	2	ALWL	2	···	···	···	···
(9)	1	ALWU	1	···	···	···	···
(10)	1	ALWU	2	−6120 (75)	9 15	9 15	−5880 (63)

Table 4. Problem (28,35)

	BND	STA	LEV	NR	R1	Q1	Q2
(1)	1	LP, .5	1	—	1 9	1 9	1 11
(2)	1	LP, .5	2	—	1 15	1 15	1 11
(3)	2	LP, 1	1	—	47 67	6 23	675 (35)
(4)	2	LP, 1	2	—	5 25	5 23	—
(5)	1	ALWL	1	···	···	1075 (43)	···
(9)	1	ALWU	1	—	—	30 45	—
(11)	2	ALWU	1	···	···	1300 (57)	···

Table 5. Diverse problems run with Q1

Type	Problem	BND	STA	LEV	Q1	
2.	(12,44)	1	LP, .5	2	1	357
2.	(5,39)	1	LP, .5	2	− 10618	(60)
2.	(20,28)	1	LP, .5	2	47	(800)

Tables 2, 3 and 4 are devoted to one sample problem each, run in a number of different ways. Table 5 gives a few results for somewhat more difficult problems. The (20, 28) problem, for example, requires on the order of one hundred LP programs, even when cutting plane techniques are used in branch-and-bound programming.

The columns headed BND and STA describe what bounds were used (on all variables) and what state was utilized. In all cases the state was computed at node 1 and updated in the obvious manner. The entry (LP, r) signifies that the state was obtained by rounding with the rounding parameter r. Column LEV refers to the search, which was used in practically all runs with a search level of 1 or 2. Search level 0 would correspond to no search.

The last columns are devoted to comparisons among four possible methods:

NR — no reduction method used at all,

R1 — reduction used as in Section 4.1,

Q1 — full reduction, Sections 4.1 and 4.2,

Q2 — full reduction; reduction (not search) used for strategy.

Under each of these 4 column headings there are two entries, NS, NT:

NS — iteration number ν at which optimal solution found;

NT — iteration number at which optimality ascertained.

However, when the second entry is in parentheses, this is meant to signify that the run did not terminate properly but was interrupted. In that case, (NT) is the iteration number at interruption and NS is the best objective function value found during the run (with some default value such as 9999, when no solution found).

The difference between Q1 and Q2 needs to be explained a little further. Having several features in an enumerative system often makes comparisons of results quite difficult. Accumulated counts of the successes of a particular technique depend strongly on the use of other techniques and on the order in which these techniques were deployed. In this particular instance, the fact that a search was used in all runs makes the subsequent use of minimal preferred inequalities somewhat ineffective. Column Q2 refers to runs in which the use of the search (to indicate indices of variables which lead to improved solutions or to points of reduced infeasibility) was suppressed.

It is not difficult to interpret the results, even though it is a little disappointing that there is so little difference among R1, Q1, Q2:

(i) The state has a very great influence on the enumeration, even if only used in a

simple manner, as here. (Dynamic state determination is clearly of interest.) This is not new (compare [13]), but often forgotten. We believe this to be on account of the fact that standard branch and bound methods work in the neighborhood of LP solutions and are therefore often successful. This does not, however, negate the great importance of codes which permit the "*imposition*" of a state (not easily possible in BB programming) for full enumeration or for heuristic programming.

(ii) Using a local search can, but need not, make a big difference. Occasionally a 1-level search misses a good solution which can be found by a 2-level search, and then the search procedure can meander uselessly for quite a while before leading back to a lattice area of interest.

(iii) Search method NR, no reduction, was no good at all. To mitigate this result somewhat, we must consider, however, that NR was obtained by cutting out all reduction. Any decent enumerative code has some provision for making inferences about fixing variables or reducing bounds. NR is, therefore, not representative of a reasonable enumerative code, in spite of containing state and search features.

(iv) There appears to be only a slight improvement due to Q1 or Q2 over R1. Our conclusion is that using a local search feature for R1 renders somewhat ineffective the use of logical inequalities for strategic purposes. Other evidence makes it fairly clear that penalties ought to be invoked in conjunction with logical inequalities.

(v) It need hardly be emphasized that integer programming remains always unpredictable. Comparing row 1 and row 10 of Table 3, for example, one sees that in one case method Q2 is much better than Q1, in the other case much worse. This only confirms the well-known impossibility of finding *one* suitable algorithm for all problems.

Finally, one might be able to make a genuine case for the use of interactivity in integer programming. Watching the behaviour of the search, e.g. by printing out List 1 and List 2 of Section 3.1, plus some selected data on obj. function, bounds and infeasibilities, ($s(i) < 0$), does give a feeling as to whether one is doing well or not.

For example, when one follows the behaviour of a search carefully, one will almost always notice that the "depth" of the *overall* search tree (namely the maximal l attained, before reduction procedures lead to backward steps) is a good indicator of whether things go well or not. Large values of l should perhaps be taken as poor behaviour and lead to redefinition of state, local search level or other parameters of the enumerative code. (In our past experience this behaviour was most evident with large plant location problems [13], where the use of a good state led otherwise quite difficult problems with 100 plants to have a search tree with level practically always below three. Given the relatively small machine at our disposal at that time, we did not resolve the problem but still believe that the search was "well behaved".)

References

[1] E. Balas and R. Jeroslow, Canonical cuts on the unit hypercube, SIAM J. Appl. Math. 23 (1972) 61–69.
[2] J.F. Benders, Partitioning procedures for solving mixed-variables programming problems, Numerische Math. 4 (1962) 238–252.
[3] B. Bouvier and G. Messoumian, Programmes lineaires en variables bivalentes, Thesis, Faculte des Sciences, Univ. Grenoble, 1965.
[4] R. Breu and C.-A. Burdet, Branch and bound experiments in zero-one programming, Math. Programming Study 2, (1974) 1–50.
[5] R.E. Gomory and E.L. Johnson, Some continuous functions related to corner polyhedra, IBM Res. Report R03311, Feb. 1971 (see also two recent papers in Math. Programming).
[6] F. Granot and P.L. Hammer, On the use of boolean functions in 0–1 programming, Methods of Operations Research, Vol. 12 (1972).
[7] M. Guignard, Preferred shadow prices in 0–1 programming, Res. Report, Dept. Stat. & OR, Wharton School, Univ. of PA, 1974.
[8] M. Guignard and K. Spielberg, Search techniques with adaptive features for certain integer and mixed-integer programming problems, Proc. IFIPS Congress, Edinburgh (North Holland, 1968).
[9] M. Guignard and K. Spielberg, The state enumeration method for mixed zero-one programming, IBM Phil. Sc. Center Rep. 320–3000, Feb. 1971.
[10] M. Guignard and K. Spielberg, A realization of the state enumeration procedure, IBM Phil. Sc. Center Rep. 320-3025, June 1973.
[11] E.L. Johnson and K. Spielberg, Inequalities in branch and bound programming, in: eds., R.W. Cottle, J. Krarup, Optimisation Methods (English Univ. Press, London, 1974).
[12] C. Lemke and K. Spielberg, Direct search algorithm for zero-one and mixed-integer programming, J. ORSA, 15 (1967) 892–914.
[13] K. Spielberg, Plant location with generalized search origin, Management Sci. 16 (1969) 165–178.
[14] K. Spielberg, Minimal Preferred Variable Reduction for Zero-One Programming, IBM Phil. Sc. Center Rep. 320 3013, July, 1972.
[15] K. Spielberg, A minimal inequality branch-bound method, IBM Phil. Sc. Center Rep. 320-3024, June 1973.
[16] K. Spielberg, Minimal preferred variable methods in zero-one programming, Working Paper, IBM Scientific Marketing, Dec. 1974.
[17] S. Zionts, Generalized implicit enumeration using bounds on variables for solving linear programs with zero-one variables, Naval Res. Logistics Quarterly 19 (1972) 165–181.

Annals of Discrete Mathematics 1 (1977) 247–271

ALGORITHMS FOR EXPLOITING THE STRUCTURE OF THE SIMPLE PLANT LOCATION PROBLEM

Monique GUIGNARD

Department of Statictics, Wharton School, University of Pennsylvania, Philadelphia, PA 19174, U.S.A.

Kurt SPIELBERG

Scientific Marketing, IBM, White Plains, NY 10604, USA

The paper is concerned with a number of approaches to the important simple plant location problem. In addition to describing several decomposition approaches, the paper focuses on modified simplex methods which exploit triangular bases.

1. Introduction

Simple (Uncapacitated) Plant Location Problems (we shall abbreviate Simple Plant Location by SPL, and SPL problem by SPLP) are of great significance both practically and theoretically. There exist telecommunication network problems which could use algorithms handling problems with thousands of "plants" and "destinations". These can only be tackled by heuristics at present.

The SPLP is one of the simplest mixed integer problems which exhibit all the typical combinatorial difficulties of mixed (0, 1) programming and at the same time have a structure that invites the application of various specialized techniques.

1.1. Brief survey

The referee's comments about the literature on SPL and related problems, for which we express our appreciation, indicate that a brief survey of some of the literature is necessary, incomplete as it must be for such a big subject.

Exact formulations appear to go back to Balinski [4]. A first enumerative algorithm of the branch-bound type, based on the "*aggregated*" constraints $\sum x(i,j) \leq m(i) \cdot y(i)$, was developed by Efroymson, Ray in [13]. It was later refined by a number of authors.

But the current state of the art must almost certainly rest squarely on the resolution of the SPLP with "*disaggregated*" constraints $x(i,j) \leq y(i)$, because the "relaxed" problem with $0 \leq y(i) \leq 1$ is very strong for the disaggregated and very weak for the aggregated form. This notion appears to have been observed and exploited independently by three groups of researchers.

Bilde and Krarup, in a paper published in Danish (in 1967), and therefore unfortunately largely unread (available now in [7]), devised excellent heuristic techniques for producing strong lower bounds on the objective function of the strong relaxed problem, exploited with good effect in a branch and bound algorithm.

A class of enumerative algorithms by Spielberg [24, 25], based on widely distributed IBM reports of 1967 and 1968, exploited the disaggregated form in terms of dual variable analysis leading to strong Benders inequalities and "*gain functions*". The work was extended to more general problems in Guignard, Spielberg [20]. A recent paper by Cornuejols, Fischer, Nemhauser [10] analyzes nicely a "greedy algorithm" which is based on one of the algorithms of [25] and has the additional merit of establishing clearly (by way of Lagrangean Techniques, due to Held and Karp and extended and summarized by Geoffrion [15]) that the disaggregated form of the constraints is indeed fully exploited in this fashion.

The third important approach (expressed in terms of the capacitated problem) is due to Davis, Ray [12], who solved the linear program by decomposition in 1967. This work established the practicability and desirability of solving the disaggregated LP directly.

What lends special interest to the above is that there has been steadily increasing recognition of the importance of disaggregation for large scale problems in the entire class of location and distribution problems, an area whose practical importance can hardly be overstated.

Without being in any sense complete, we can cite work on the M-Median Plant Location Problem by Garfinkel, Neebe, Rao [18], a successful application of Benders' algorithm to a large distribution problem by Geoffrion, Graves [16], and a general account of formulation techniques by Williams [26].

Finally we have recently seen the resolution of quite large distribution problems, with several thousand integer varables, by the general purpose code MPSX-MIP of IBM, after suitable introduction of disaggregated constraints (e.g., E.L. Johnson, private communication).

1.2. Approach of current paper

The following paper focuses first on decomposition and then on new possibilities for exploitation of the fully disaggregated linear program. In the latter area one might also consult the work of Marsten [22] and Graves, McBride [19] on specialized Simplex Methods.

Recent papers of Schrage [23] on implicit representation of generalized variable upper bounds, and Glover [17] on compact LP bases provide general techniques for problems which we called "weakly linked" in [5] and [20], a class of problem which encompasses location and more general fixed charge problems.

Finally, it may be of interest that there is a link to the Russian literature via the two references Frieze [14] and Babayev [1].

The first paper demonstrates a property for the gain functions of [24, 25], and the second relates this property to a "method of successive calculation" of Cherenin [9].

It is always enticing to start by decomposition techniques in order to get good bounds on the objective function. Equally interesting is the construction of specialized simplex algorithms which attempt to adhere to the great abundance of all-integer vertices as much as is at all possible. We have been able to solve a (20, 35) SPLP by a linear programming triangularization method, carefully bypassing all fractional vertices which would naturally lie in the path of an unmodified primal algorithm.

Such attempts have been given new impetus by the results of Balas and Padberg, given in [2, 3], to the effect that there is always a path of integer vertices leading to the integer optimum of a SPLP. This is a nice result, but an algorithm such as suggested in [3] runs into formidable difficulties which appear to be very much of an enumerative nature.

As opposed to the "usual" set-packing problem treated in [2, 3], the SPLP is unusual in the sense that as linear programming (LP) problem it is enormously large for problems which must be considered small in practice.

To tackle the SPLP successfully, then, one must have highly specialized tools for treating everything within the LP problem implicitly. In Section 3 we discuss a certain special basis representation, which we believe must play a role (possibly in a yet somewhat more modified form) in any efficient direct linear programming treatment of the SPLP.

Actually, we believe that Section 3 is important in several respects. The possibility of constructing triangular bases which lead to easily obtainable updated tableaux can be exploited for writing *computationally efficient* codes for problem sizes which would otherwise be intractable. What may be just as important, the latitude in constructing such bases can apparently be exploited to render them "*good*", in the sense of minimizing the number of negative reduced costs (related to gain functions which have been found to be important elsewhere).

Finally, these triangularization procedures are such that they can be applied to *any integer feasible solution*, no matter how it was found. This opens the way to a class of algorithms, dependent on the actual triangularization process adopted, consisting of steps such as:

(1) Heuristics, enumeration, etc., to give a feasible integer solution.

(2) Construction of a "good" triangular basis, and therefore a simply structured (implicit) updated tableau.

(3) Exploration of neighbor vertices. Pivot or block pivot to neighbor vertex.

(4) Return to (1).

I.e., depending on the actual basis choices, one has a class of true hybrid algorithms, which are LP intermittently, but then also permit *jumps* from one lattice point to completely different lattice points without loss in efficiency of LP

computation (e.g., given a solution point (x, y) arrived at by a LP step, it is possible that x can be improved for given y by inspection, or the jump might correspond to one of the simple heuristics which are easily available for SPLP). Notice that tableaux are never updated, since bases and inverses are easily constructed from the solution.

2. Decomposition and partitioning methods

There are many possibilities of decomposition and partitioning. On balance they are by now quite well known. We believe that the "*reverse partitioning*" of 2.2.3 is new, somewhat unusual, and therefore interesting.

What is most important, however, is the potential utilization of the special problem characteristics. It is clear that the difference between success and failure lies here, and we have tried to present ideas which might form a start for a real algorithm (stand-alone or auxiliary algorithm within enumeration).

$$\begin{aligned} &\min z = \sum f(i)\,.\,y(i) + \sum c(i,j)\,.\,x(i,j), \\ &\sum x(i,j) = 1, \quad \text{all } j, \\ &x(i,j) \leqslant y(i), \quad \text{all } i,j, \\ &y(i) \text{ 0 or 1}, \; x(i,j) \geqslant 0. \end{aligned} \tag{2.1}$$

The indices i and j range from 1 to m, and 1 to n, respectively. We admit only $f(i) \geqslant 0$ and $c(i,j) \geqslant 0$. Whether we consider (2.1) or its relaxed LP form (all $y(i)$ between 0 and 1) will usually be clear from the context.

2.1. *Dantzig and Wolfe decomposition* [11]

Consider the relaxed problem:

$$\begin{aligned} &\min fy + cx \quad (= gt) \\ &\text{s.t.} \sum_t x_{tj} = 1 \quad (= At), \\ &\left.\begin{aligned} -y_i + x_{ij} &\leqslant 0 \\ 1 \geqslant x_{ij}, \quad y_i &\geqslant 0 \end{aligned}\right\} (Bt \geqslant 0), \end{aligned}$$

where $g = (f, c)$, $t = \binom{y}{x}$. Let. $t^1, t^2, \ldots, t^k, \ldots$, be the extreme points of $\mathscr{B} = \{t \in \mathbb{R}^{m+mn} \mid Bt \geqslant 0\}$ (a compact set), and let K be the set of their indices. Then, for all $t \in \mathscr{B}$, there exists $\lambda = (\lambda_1, \ldots, \lambda_k, \ldots)$ such that

$$\lambda \geq 0,$$

$$\sum_{k \in K'} \lambda_k = 1,$$

$$t = \sum_{k \in K'} \lambda_k t^k.$$

Then the decomposition algorithm will consider the following two problems:

(λ-P)

$$\min_{\lambda} \sum_{k \in K'} \lambda_k (g t^k) = \sum_{k \in K'} \lambda_k z^k,$$

$$\text{s.t.} \sum_{k \in K'} \lambda_k (A_j t^k) = 1, \quad j = 1, \ldots, m,$$

$$\sum_{k \in K'} \lambda_k = 1, \quad \lambda_k \geq 0, \forall k \in K',$$

where K' is the index set of currently known vertices of $\mathfrak{B}$, with the dual

(λ-D)

$$\hat{z} = \max_{u, v} \sum_{j=1}^{m} u^j + v$$

$$-g t^k + \sum_{j=1}^{m} u^j (A_j t^k) + v \leq 0$$

and the problem

(B-P)

$$\hat{d} = \min_{t} \left(g t - \sum_{j=1}^{m} u^j A_j t - v = d \right)$$

$$\text{s.t. } t \in \mathfrak{B}, \text{ i.e. } \begin{cases} 0 \leq x_{ij} \leq y_i, & \text{all } i, j. \\ 0 \leq y_i, & \text{all } i. \end{cases}$$

In fact, (λ-P) and (B-P) can be rewritten

(λ-P)

$$\hat{z} = \min_{\lambda} \sum_{k \in K'} \lambda_k \left(\sum_i f_i y_i^k + \sum_{i,j} c_{ij} x_{ij}^k \right) = \sum \lambda_k z^k$$

$$\text{s.t.} \sum_{k \in K'} \lambda_k \left(\sum_i x_{ij}^k \right) = 1, \quad j = 1, \ldots, n$$

$$\sum_{k \in K'} \lambda_k = 1, \quad \lambda_k \geq 0 \text{ all } k \in K'.$$

(B-P)

$$\hat{d} = \min_{x, y} \left(\sum_i f_i y_i + \sum_{i,j} c_{ij} x_{ij} - \sum_j u^j \left(\sum_i x_{ij} \right) - v = d \right)$$

$$= \sum_i f_i y_i + \sum_j \sum_i (c_{ij} - u_j) x_{ij} - v,$$

$$0 \leq x_{ij} \leq y_i,$$

$$0 \leq y_i \leq 1.$$

The continuous objective function of SPLP is in the bracket:

$$\hat{z} + \hat{d} \leq \bar{z} \leq \hat{z}.$$

It is well known [8] that the algorithm converges even when (B-P) is not optimized, but suboptimized, i.e. as long as the solution $t = (y, x)$ is chosen so as to render d negative and to be an extreme point of $\mathfrak{B}$. Also, as long as there are feasible integer solutions to SPLP whose objective function values are between z and the current best value $z^* = \min z$, every such solution is not yet included in the set of generators of $\mathfrak{B}$ and would yield an improvement over the current $\sum \lambda_k t^k$, i.e. yield a negative value for d. If the optimal solution is not integral, the integer optimum is among those feasible solutions that render the last d negative.

Also, if an improving ($d < 0$) feasible solution to SLPL is the only generator added at that iteration, it will be the optimum of the next (λ-P) problem. Yet it might be better to add both the optimum and a feasible solution of SPLP simultaneously.

Remark 1. Every time a (λ-P) problem is solved, its solution yields a new feasible solution to the original LP. It is of the form (y, x). Keeping in mind the original problem, one may be able to find a better solution by taking y' defined by

$$y'(i) = \max_j x(i, j),$$

the new cost $f.y' + c.x$ being no larger than $f.y + c.x$. This is important, since $f.y + c.x$ (or $f.y' + c.x$) is an upper bound for the optimal value of the original problem. If (y', x) is an extreme point of $\mathfrak{B}$, one can add it to the current set of generators.

Remark 2. The constraints of a (B-P) problem are such that the problem is separable, as the constraints which enforce the presence of exactly one $x(i, j) = 1$ per column have disappeared from its formulation. Each (B-P) yields m subproblems of the form

$$\min f_i y_i + \sum_j (c_{ij} - u_j) x_{ij} - v/m,$$

$$\text{s.t. } 0 \leq x_{ij} \leq y_i \leq 1,$$

whose solution is obvious: if $c(i, j) - u(j) \geq 0$, set $x(i, j)$ to 0. Then, if there are some $c(i, j) - u(j) < 0$, set $x(i, j)$ equal to, say, a. Thus, $y(i)$ must be at least equal to a. The objective function is then equal to $(f(i) + \sum_{c_{ij} < u_j} c(i, j) - u(j)) . a$. If the coefficient of a is negative, set $a = 1$, otherwise set $a = 0$. One can follow this by an attempt to find a suboptimal, feasible solution to SPLP such that $d < 0$.

Remark 3. An initial set of extreme points of $\mathfrak{B}$ should be carefully chosen to allow generation of meaningful points from the outset. For instance, one might choose:

$t_i^0 = 0$ all l, $t_i^1 = 1$ all l, $t^2, \ldots, t^{k+1}, \ldots, t^{n+1}$

such that there is one 1 in column k of x, corresponding to the smallest $c(i,j)$, the corresponding $y(i)$ being set equal to 1, all others to 0.

Example. The following data

$$f = \begin{bmatrix} 1 \\ 1 \\ 1 \end{bmatrix}, \qquad c = \begin{bmatrix} 1 & 1 & 10 \\ 1 & 10 & 1 \\ 10 & 1 & 1 \end{bmatrix}$$

yield a continuous optimal solution

$$y = \begin{bmatrix} \frac{1}{2} \\ \frac{1}{2} \\ \frac{1}{2} \end{bmatrix}, \qquad x = \begin{bmatrix} \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & 0 & \frac{1}{2} \\ 0 & \frac{1}{2} & \frac{1}{2} \end{bmatrix}, \quad \bar{z} = 4.5$$

and several integer optimal solutions among them:

$$y = \begin{bmatrix} 1 \\ 1 \\ 0 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 & 1 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix}, \qquad z = 5.$$

Choosing

$$t^0 = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right], \quad t^1 = \left[\begin{array}{c|ccc} 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 \end{array}\right], \quad t^2 = \left[\begin{array}{c|ccc} 1 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right],$$

$$t^3 = \left[\begin{array}{c|ccc} 1 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \end{array}\right], \quad t^4 = \left[\begin{array}{c|ccc} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 \end{array}\right],$$

$z^0 = 0, \quad z^1 = 39, \quad z^2 = 2, \quad z^3 = 2, \quad z^4 = 2,$

one solves four linear programs of type (λ–P), arriving at a last (B–P) problem of the form:

$$\min \sum \left[\begin{array}{c|ccc} 1 & 1-\frac{1}{2} & 1-\frac{1}{2} & 10-\frac{1}{2} \\ 1 & 1-\frac{1}{2} & 10-\frac{1}{2} & 1-\frac{1}{2} \\ 1 & 10-\frac{1}{2} & 1-\frac{1}{2} & 1-\frac{1}{2} \end{array}\right] (y,x) + 0.$$

No solution yields $d < 0$. Hence the last solution of (λ–P) is optimal:

$$\bar{t} = \left[\begin{array}{c|ccc} \frac{1}{2} & \frac{1}{2} & \frac{1}{2} & 0 \\ \frac{1}{2} & \frac{1}{2} & 0 & \frac{1}{2} \\ \frac{1}{2} & 0 & \frac{1}{2} & \frac{1}{2} \end{array}\right].$$

The integer optimum satisfies:

$$\frac{5}{2} + \sum \left[\begin{array}{c|ccc} 1 & -\frac{1}{2} & -\frac{1}{2} & \frac{1}{2} \\ 1 & -\frac{1}{2} & \frac{1}{2} & -\frac{1}{2} \\ 1 & \frac{1}{2} & -\frac{1}{2} & -\frac{1}{2} \end{array}\right] (y,x) < 0. \qquad \text{(C)}$$

Conclusion. Instead of solving a LP with 12 rows and 21 variables, one solves 4 LP's with 4 rows and between 5 and 9 columns, each of them being identical with the previous one with one new column added (so that relatively few pivot steps are required). A condition (C) has been found which the integer solution must satisfy, and a bracket for the optimal value has been obtained:

$$4.5 \leqslant z \leqslant 5.$$

2.2. *Benders partitioning* [6]

2.2.1. *The general scheme*

Consider the problem P:

$$\min\{z = \delta\tau + \gamma\chi \mid D\tau + C\chi \leqslant \beta, \tau \geqslant 0, \chi \in S\}.$$

It can be rewritten as

$$\min_{\chi \in S} \left\{\gamma\chi + \min_{\tau} \{\delta\tau \mid D\tau \leqslant \beta - C\chi, \tau \geqslant 0\}\right\}$$

or

$$\min_{\chi \in S} \left\{\gamma\chi + \max_{u} \{u(-\beta + C\chi) \mid uD + \delta \geqslant 0, u \geqslant 0\}\right\}.$$

Let R be $\{u \mid u.D + \delta \geqslant 0,\ u \geqslant 0\}$. R is independent of χ. If $R = \emptyset$, P has no solution.

Otherwise, if for some χ there exists u^k, an extreme ray of R, such that $u^k.(-\beta + C.\chi) > 0$, $D = \max_{u \in R} u.(-\beta + C.\chi)$ is unbounded and $L = \min_{\tau \geqslant 0}\{\delta\tau \mid D\tau \leqslant \beta - C.\chi, \tau \geqslant 0\}$ has no solution.

P can therefore be solved as G:

$$\min_{\chi \in S} \left\{z \mid z \geqslant \gamma\chi + \max_{u \in R} \{-u(\beta - C.\chi) \mid u \text{ either extreme point of } R \text{ or extreme ray of } R \text{ satisfying } u(\beta - C.\chi) \geqslant 0\}\right\}.$$

An algorithm would proceed as follows:

Step 0. Set $k = 0$.

Step 1. Replace k by $k+1$. Form Q, the set of indices of known extreme points u^l and extreme rays u^h of R. For $k = 1$, choose any $\chi^1 \in S$, go to 3.

Step 2. Solve G^k over Q. If G^k has no feasible solution, the same is true for P. In that case terminate. Otherwise, let χ^k, z^k be an optimal solution of G^k.

Step 3. Solve L^k and D^k with $\chi = \chi^k$. Let $f(\lambda^k)$ be their optimal values. If $f(\chi^k)$ is $-\infty$, terminate with no feasible solution. If $f(\chi^k)$ is $+\infty$, there is no feasible τ associated with χ^k; let u_*^k be the optimal extreme ray of D^k. Go to 1. If $f(\chi^k)$ is finite, let (τ^k, u^k) be the optimal pair. It is a feasible solution for P. If

$$z^k \geqslant \gamma\chi^k + f(\chi^k),$$

then (χ^k, t^k, z^k) is optimal for P; terminate. Otherwise go to 1 with a new extreme point u^k of R.

We shall call the "normal" case, in which one identifies χ with y, "*direct partitioning*". By contrast we shall use the term "*reverse partioning*" for the identification $\chi = x$. (It is interesting that one can, at least for some problems, bring back the integrality conditions, which in the direct case are taken care of (formally) by $\chi \in S$, in an indirect fashion.)

2.2.2. Direct partioning

Starting with an arbitrary y (often $y' = (1, 1, \ldots, 1)$, all plants open), one solves alternatively the following two problems. Firstly L^k

$$\min \left\{ \sum c(i,j) . x(i,j) \,|\, x(i,j) \leqslant y^k(i),\ \sum_i x(i,j) = 1,\ x(i,j) \geqslant 0 \right\}$$

and its dual D^k

$$\max \left\{ \sum v(j) - \sum w(i,j) . y^k(i) \,|\, w(i,j) + c(i,j) \geqslant v(j),\ w(i,j) \geqslant 0 \right\}$$

with solutions $x^k(i,j)$, $v^k(j)$, $w^k(i,j)$; and secondly G^{k+1}

$$\min_{y \in S} \left\{ z \,|\, z \geqslant \sum_i \left[f(i) - \sum_j w^h(i,j) \right] . y(i) + \sum_j v^h(j),\ h = 1, \ldots, k \right.$$

and (w^h, v^h) extreme point of R;

$$\sum w^h(i,j) . y(i) \geqslant \sum_j v^h(j),\ h = 1, \ldots, k$$

$$\left. \text{and } (w^h, v^h) \text{ extreme ray of } R \right\} .$$

Example. The 3 by 3 problem used before yields the following sequence of problems:

L: $z = 6$, inequality for G: $\Sigma y(i) + 3 \leqslant z$;
G: $z = 3$, $3 \leqslant \bar{z} \leqslant 6$, $y = (0,0,0)$;
L: infeasible, inequality for G: $\Sigma_i y(i) \geqslant 1$;
G: $z = 4$, $4 \leqslant \bar{z} \leqslant 6$, $y = (1,0,0)$;
L: $z = 13$, inequality for G: $(1, -8, -8)y + 12 \leqslant z$;
G: $z = 4$, $y = (0,1,0)$, $4 \leqslant z \leqslant 6$;
L: $z = 13$, inequality for G: $(-8, 1, -8)y + 12 \leqslant z$;
G: $z = 4$, $y = (0,0,1)$, $4 \leqslant z \leqslant 6$;
L: $z = 13$, inequality for G: $(-8, -8, 1)y + 12 \leqslant z$;
G: $z = 4.5$, $y = (.5, .5, .5)$;
L: $z = 4.5$, optimal.

We have run bigger problems and have experienced the normal difficulties towards the end, as the number of constraints in G increases. We have tried two versions of the algorithm, the one described above, and another one in which some

heuristics are used to render y integer if it turns out fractional. In the second case we noticed much faster convergence. E.g., a 20 by 35 problem shows the following behavior: after 20 iterations, the first algorithm gives the interval $209.0 \leqslant z \leqslant 344.8$, whereas the second has arrived at $235.7 \leqslant z \leqslant 245$. The optimal value is 243.

2.2.3. Reverse partitioning

One now has to solve, for $S = \{x \,|\, \Sigma_i x(i,j) = 1, \text{ all } j\}$,

$$\min_{x \in S} \left\{ c . x + \min_y \{ f . y \,|\, 0 \leqslant x(i,j) \leqslant y(i), \forall j \} \right\} ,$$

or

$$\min_{x \in S} \left\{ c . x + \max_t \left\{ \sum_{i,j} t(i,j) . x(i,j) \,\middle|\, t(i,j) \geqslant 0,\ \sum_j t(i,j) \leqslant f(i) \right\} \right\}$$

i.e., L^k is $\min_y f . y$

$$y(i) \geqslant x^k(i,j), \quad \forall i,j$$

$$y(i) \geqslant 0, \quad \forall i$$

whose solution is clearly $y(i) = \max_j x^k(i,j)$, all i. D^k is

$$\max_t \sum t(i,j) x^k(i,j)$$

$$t(i,j) \geqslant 0, \quad \forall i,j$$

$$\sum_j t(i,j) \leqslant f(i), \quad \forall i$$

and G^k is

$$\min_{x \in S} \left\{ z \,|\, z \geqslant c . x + \max_t \left\{ t . x \,|\, t = t^1, \ldots, t^k, \right. \right.$$

$$\left. \left. t \text{ extreme point of the set } \left\{ t \geqslant 0,\ \sum_j t(i,j) \leqslant f(i) \right\} \right\} \right\}$$

whose dual reads

$$\max \left\{ \sum_j v(j) \,\middle|\, v(j) - \sum_p w^p c_p(i,j) \leqslant 0,\ w^p \geqslant 0,\ \sum_p w^p = 1 \right\}$$

with $c_p = c + t^p$. Let $d(i,j) = \Sigma_p w^p c_p(i,j) - v(j)$. Sum p from 1 to k. Given w, one can determine v and d via

$$v(j) = \min_i \sum_p w^p . c_p(i,j) \quad \text{and} \quad d(i,j) = \sum_p w^p c_p(i,j) - v(j) \geqslant 0.$$

(1) The reduced cost of a $d(i,j)$ is $-x(i,j)$, $0 \leqslant x(i,j) \leqslant 1$, so that the only candidates to enter the basis are the w's.

(2) If w^{p_1} comes in and w^{p_2} goes out, the pivot row is $\sum_p w^p = 1$.
(3) If w^{p_1} comes in and $d(i,j)$ goes out, the "pivot row" is

$$v(j) - \sum_p w^p c_p(i,j) + d(i,j) = 0.$$

(4) Consider the constraint

$$v(j) - \sum_p w^p c_p(i,j) + d(i,j) = 0.$$

(a) Either $v(j) < \sum w^p c_p(i,j)$, then $d(i,j) > 0$ is basic and the constraint gives the $d(i,j)$, or
(b) $v(j) = \sum w^p c_p(i,j)$, then
(i) either $d(i,j) = 0$ nonbasic ... row gives v or basic w,
(ii) or $d(i,j) = 0$ basic ... row gives $d(i,j)$.

For the 3 by 3 problem we obtain the continuous optimum after 10 iterations, i.e. 10 LP's with 10 rows and between 4 and 10 variables (exclusive of the slacks) of type GD. In fact one does not really need to use an LP code to solve GD, but rather one uses a specialized technique involving much less computation.

The optimal value is immediately in the interval (3, 5), then at the 2nd iteration in (4, 5), then (4th iteration) (4.167, 4.83), finally in (4.5, 4.83) at the 5th iteration. The next iterations leave the bracket unchanged, until the 10th iteration gives the optimum 4.5.

3. Modified simplex methods

3.1. Simple plant location and the simplex method

The SPLP lends itself rather well to solution by LP techniques, in the sense that the LP solutions are often integral. There are many bases which are unimodular (vertices which are integral). A standard simplex algorithm, however, will encounter fractional vertices.

Also, the LP tableau of the SPLP is large. E.g., a 10 plant, 20 customer problem corresponds to a LP with $20 + 10.20 = 220$ rows and $10.20.2 + 10 = 410$ variables, including slacks.

An efficient implementation of the simplex method, then, requires that:
(i) The structure of the LP be carried along implicitly, all relevant elements of the updated tableau being generated as needed, and
(ii) efforts be made to avoid fractional vertices.

3.2. A triangularization algorithm [21]

We have implemented a simple "*triangularization*" algorithm, which tries to accomplish these objectives. In outline, it functions as follows.

(1) Consider (2.1) as an equality system with slacks $s(i,j)$, i.e. with the constraints:

$$x(i,j) + s(i,j) = y(i), \quad \text{all } i,j.$$

An initial triangular basis is easily found. To simplify matters, we always included (and maintained) the $y(i)$ in the basis. (In retrospect, we believe that this may be too restrictive.)

(2) At a typical iteration, given the triangularity of the basis, we can compute the dual variables by recursive scanning of the dual constraints and substitution. If the problem is not optimal (dually feasible) we select an incoming variable $t(i^*, j^*)$, which represents either $x(i^*, j^*)$ or $s(i^*, j^*)$.

(3) We generate the pivot column by scanning the primal constraints and expressing the basic variables $y^B(i)$ and $x^B(i,j)$ in terms of $t(i^*, j^*)$. This is possible on account of triangularity, and the scanning can be used to exhibit the sequence of variables which shows the triangularity of the basis explicitly. It is clear, that the basic $s^B(i,j)$ can be generated afterwards from the constraints (2.1), so that only the x^B, y^B computations require iterative scanning.

(4) Given the constant column (the values of the basic variables) and the pivot column developed in (3), we can perform the standard ratio tests and decide on an outgoing variable. Let $t^B = b + t^* . p + \cdots$ (p... pivot column) represent the basic variable vector t^B in terms of its current value b and the incoming variable t^*. The $b(i)$ are either 0 or 1, but the pivot column may, in general, contain integer entries other than 0, 1, -1.

In the ratio test one searches for an outgoing variable $t(i^{**}, j^{**})$ which corresponds to a maximal $b(i)/p(i)$, over $p(i) < 0$. When the maximal ratio is zero, the pivot step is *degenerate* (does not change the value of the solution; one remains at the same vertex of the polytope). It can be seen that one can then find an eligible i^{**} for which the $p(i)$ is -1, so that the new basis remains unimodular. When the maximal ratio is -1, we have a non-degenerate pivot step which leads to a new unimodular basis. When the maximal ratio is fractional, i.e., when the $p(i)$ is negative other than -1, we abandon the incoming variable t^* because the new basis would have to be non-triangular. In effect, one abandons motion along one edge of the polytope from the current vertex to what would most likely (apparently there are exceptions) be a fractional neighbor.

Comments. (i) In our code all array representations are kept in binary form. We do not generate $p(i)$ which are other than 0, 1, -1, but carry along a fourth type (represented by a code of two bits) which we designate as "*polluted*". Linear combinations of polluted entries are also designated as polluted. We abandon incoming variables (candidate edges) which lead to a polluted $p(i^{**})$. This means that our code is somewhat too restrictive (pessimistic).

(ii) The code will *fail in two cases*:

(a) There are no candidate edges leading to a unimodular new basis.

(b) A new unimodular basis has been found and is yet non-triangular.

(iii) For very small problems (we have run a large number of problems with $m = 4$ and $n = 6$) we have not been able to get one of the two conditions mentioned above, no matter what data we tried.

Failure (a) apparently is unlikely for "easy" fixed charges. It can be "induced" most readily by using uniformly large fixed charges (rendering the problem almost fully combinatorial).

We have only one example for failure (b), for a fairly large problem of 20 plants and 35 customers. The unimodular basis which appears to be non-triangular is of size (735 by 735).

(iv) The real flaw of the method, however, lies in two other circumstances. One is the well-known problem of *degeneracy*, which leads to large numbers of apparently useless pivot steps. The other is that a method which treats the $x(i, j)$ and $s(i, j)$ as the important variables is probably doomed to failure because of *dimensionality*. We are now convinced that a direct LP technique will have to concentrate on the $y(i)$, just as is done by enumerative methods. Our choice of taking all $y(i)$ always basic and preventing them from leaving the basis was probably unwise, and the methods of the next section are probably more appropriate.

Table 1 exhibits selected computational results for small problems. The code permits slight changes in initialization and selection of incoming variables. We do not attribute any significance to such changes and only use an asterisk to distinguish between two similar yet different runs.

Table 1

Dimensions	Problem #	completed		Number of iterations	Number of fractional vertices discarded	Optimal value		Value at termination
		yes	no reason for failure			Int.	Cont.	
4×6	(1) ($f_i = 10^4$)	✓		11	0	20024	id	id
	(1) (f_i small)	✓		6	0	44	—	—
	(2) ($f_i = 10^4$)	✓		16	4	10026	—	—
	(2) (f_i small)	✓		11	1	25	—	—
	(1*) ($f_i = 10^4$)	✓						
	(1*) (f_i small)	✓		6	0	44	—	—
10×10	(1) (f_i small)	✓		14	0	80	?	80
	(2) (f_i large)		✓ a	34	37	?	?	30058
	(1*)	✓		14	0	80		80
	(2*)		✓ a	35	43	?		30058
20×35	(1)		✓ b	54	large	243	243	252
	(1*)	✓		84	moderate	243	343	243

3.3. Some special triangular bases

The SPLP can be (as was discussed among A. Hoffman, E.L. Johnson and M. Padberg, and suggested to us by A. Hoffman) reformulated as follows in terms of variables $\bar{y}(i) = 1 - y(i)$:

$$\begin{aligned} &\min \sum_i f(i).(1-\bar{y}(i)) + \sum_{i,j} c(i,j).x(i,j), \\ &\sum_i x(i,j) = 1, && \forall j, \\ &x(i,j) - 1 + \bar{y}(i) \leqslant 0, && \forall i,j, \\ &0 \leqslant x(i,j) && \forall i,j, \\ &y(i) \text{ in } \{0,1\}, && \forall i, \end{aligned} \tag{3.1}$$

or

$$\begin{aligned} &\min \left[\sum_{i,j} c(i,j).x(i,j) - \sum_i f(i).\bar{y}(i) \right] + \sum_i f(i), \\ &\sum_i x(i,j) = 1, && \forall j, \\ &x(i,j) + \bar{y}(i) + s(i,j) = 1, && \forall i,j, \\ &1 \geqslant x(i,j), \quad \bar{y}(i), \; s(i,j) \geqslant 0, && \forall i,j, \\ &y(i) \text{ in } \{0,1\}, && \forall i. \end{aligned} \tag{3.2}$$

(3.2) is a highly structured and generally very large *set partitioning* problem. Therefore, the interesting results of [2] and [3] apply, even though their practical applicability is uncertain in view of the large problem size.

Note that:

(1) a given $x(i, j)$ appears in exactly two explicit equations, one of which ($\Sigma_i x(i, j) = 1$) we shall term the Σj (*sigma j*) *equation*, while the other ($x(i, j) + s(i, j) + \bar{y}(i) = 1$) shall be referred to as $* ij$ (*cross ij*) equation;

(2) a given $\bar{y}(i)$ appears in n equations $* i1, \ldots, * in$;

(3) a slack $s(i, j)$ appears only in one $* ij$ equation. These observations are important in pointing out how basic variables can be computed. One may establish a number of useful properties:

Property 1. A basic $s(i, j)$ can be determined only from $* ij$, so that when the involved $x(i, j)$ and $y(i)$ have been determined, the basic $s(i, j)$ is known. I.e., once the basic $x(i, j)$ and $y(i)$ have been computed, the basic $s(i, j)$ can be determined in triangular fashion (one at a time). Therefore, one need only be concerned with the subbasis $B^{n,1}$, the subbasis whose columns correspond to x and $\bar{y}$.

Property 2. Given j, there must be at least one $x(i, j)$ basic expressed from Σj. All other basic $x(i, j)$ must come from $* ij$.

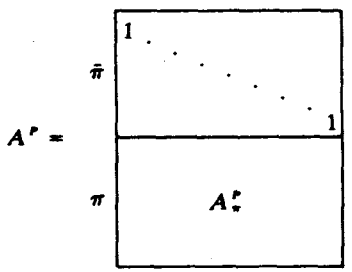
Property 3. Given i, a basic $\bar{y}(i)$ must be determined from one of the $*ij$. Therefore, if all $x(i,j)$ are basic, at least one of them must come from a Σj, so that $\bar{y}(i)$ can be computed.

Property 4. The subbasis corresponding to basic $\bar{y}(i)$ and $x(i,j)$ equal to 1 can be rearranged so as to be triangular.

The constraint matrix has only coefficients 0 and 1, the right hand side contains only 1's; there must therefore be exactly one 1 per row in the submatrix. There is no zero column. Hence there must exist a permutation of the rows and columns which brings an identity matrix to the upper part of the submatrix.

Property 5. It is always possible to complete the basis in a triangular fashion by choice of basis columns which correspond to variables at zero.

Let A be the constraint matrix; let P be the set of indices of variables at 1 and let π and $\bar{\pi}$ be suitable index sets. Then the subbasis of Property 4 is



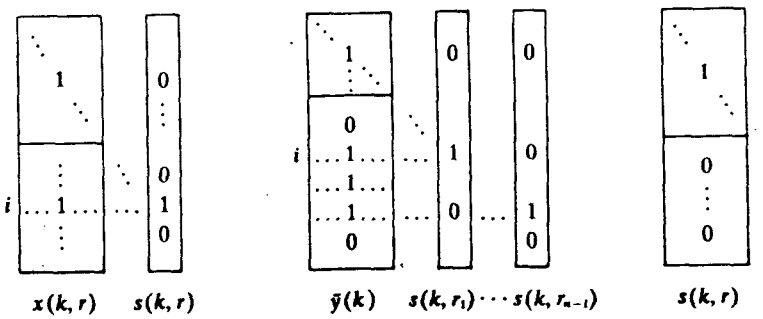
(a subscript is used for row indexing, a superscript for column indexing).

Consider A^P_π. It also has exactly one 1 per row. Consider row i, $i \in \pi$. It has one 1 in column $j(i)$, which can correspond to an x, a y or an s. We shall give one possible way of completing the basis:

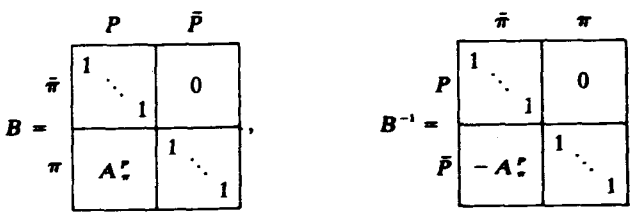
(1) if the entry corresponds to an x, say $x(k,r)$, $x(k,r) = 1$ in the current solution, so that $\bar{y}(k)$ and $s(k,r)$ are 0 and not yet in A^P. Since $s(k,r)$ occurs only in one equation $(*kr)$, one can append column $s(k,r)$ to A^P.

(2) if the entry corresponds to a $\bar{y}$, say $\bar{y}(k)$, $\bar{y}(k) = 1$ means $x(k,j) = 0\ \forall j$, and $s(k,j) = 0\ \forall j$. The columns $x(k,j)$ contain two 1's and one of these might be above the main diagonal, whereas the columns $s(k,j)$ contain only one 1 and $(n-1)$ of them are adjoined to A with their 1's on the diagonal. One of the $s(k,r)$ will be nonbasic (its choice is arbitrary).

(3) the entry can not correspond to an $s(k,j)$ at 1, since an $s(k,j)$ column has only one 1 which is in A^P_π.

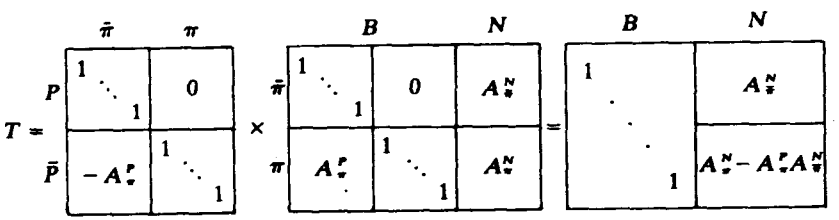


Property 6. The basis thus constructed (which we shall call the *s-canonical* basis, since only s columns were added), has the *anti-involutive* property:



Property 7. The top rows $\bar{\pi}$ of the *updated tableau* are unchanged, whereas the "*bottom*" rows π are equal to the original rows minus one of the top rows.

Let $T = B^{-1}.A$, then



and since A^P_π has only one nonzero element per row, one subtracts one row of the top from one row of the bottom.

More precisely, let us use the following notation:

(a) in each column j, let $i(j)$ be the route on which x is 1:

$$x(i(j), j) = 1.$$

(b) if $\bar{y}(i) = 1$, $(n-1)$ of the $s(i,j)$ are basic. Let ij_0 be the index of the nonbasic s (notice that $x(i,j)$ is also nonbasic); we shall refer to $(x(i,j_0), s(i,j_0))$ as the ***nonbasic pair*** associated with a $\bar{y}(i)$ at 1 (i.e., with a closed plant), then

$$T^{f_i}_{s_{ij}} = 1,$$

$$T^{x_{ij_0}}_{x_{i(j_0)j_0}} = 1, \quad T^{x_{ij_0}}_{\bar{y}_i} = 1, \quad T^{x_{ij_0}}_{s_{i(j_0)j_0}} = -1, \quad T^{x_{ij_0}}_{s_{ij_0}} = -1, \quad j \neq j_0,$$

$$T^{x_{ij}}_{x_{i(j)j}} = 1, \quad T^{x_{ij}}_{s_{ij}} = 1, \quad T^{x_{ij}}_{x_{i(j)j}} = -1 \quad \text{if } s_{ij} \in B,$$

$$T^{x_{ij}}_{\bar{y}_i} = 1, \quad T^{x_{ij}}_{s_{ij'}} = -1, \quad j' = j_0,$$

are the nonzero nonbasic entries of the updated tableau.

In the *s-canonical* basis no $\bar{y}(i)$ or $x(i,j)$ at 0 is basic.

Property 8. The reduced costs of the nonbasic variables are:

$$d^N = c^N - c^B A^N_B$$

in particular

$$d(\bar{y}(k)) = -f(k),$$

$$d(x(i,j)) = c(i,j) - c(i(j),j) + \langle s(i,j) \in N \rangle . f(i),$$

$$d(s(i,j)) = f(i),$$

where

$$\langle s(i,j) \in N \rangle = \begin{cases} 1 & \text{if } s(i,j) \in N, \\ 0 & \text{if } s(i,j) \in B. \end{cases}$$

Every nonbasic $\bar{y}(k)$ is therefore a candidate for entering the basis; a nonbasic $x(i,j)$ is a candidate if its cost plus possibly the ith fixed charge is smaller than the cost of the route currently used in column j. All these pivot steps are degenerate, since the bottom part of the right hand side consists of 0's, and each candidate column has positive entries in the bottom part. Any move to a better neighbor integer vertex therefore involves a block pivot (see [2] and [3]).

Example. Take f and c as in 2.1. Consider the solution

$$y = \begin{bmatrix} 1 \\ 0 \\ 0 \end{bmatrix}, \qquad x = \begin{bmatrix} 1 & 1 & 1 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}.$$

The cost is 13. One can construct the s-canonical basis displayed in Table 2, making $s(2,2)$ and $s(3,1)$ nonbasic on account of the large associated costs of x.

Table 2. s-canonical basis: (B, T^N)

	x_{11} x_{12} x_{13}	$\bar{y}_2$ $\bar{y}_3$	s_{11} s_{12} s_{13} s_{21} s_{23} s_{32} s_{33}	x_{21} x_{22} x_{23} x_{31} x_{32} x_{33}	s_{22} s_{31}	$\bar{y}_1$
$\Sigma 1$	1			1 1		
2	1			1 1		
3	1			1 1		
•22				1	1	
31		1		1	1	
11	1		1	−1 −1		1
12	1		1	−1 −1		1
13	1		1	−1 −1		1
21		1	1	1 −1	−1	
23		1	1	−1 1	−1	
32		1	1	−1 1	−1	
33		1	1	−1 1	−1	

$d^N = (0, 10, -9, 10, 0, -9, 1, 1, -1)$

The negative entries in the nonbasic tableau belong to the upated tableau, the positive ones are the original entries which are preserved in the transformation to the updated tableau. $x(2,3)$, $x(3,3)$, $\bar{y}(1)$ are candidates to enter the basis, but all yield degenerate steps if taken alone.

Block pivot [3]. One looks for a set K of nonbasic columns to bring into the basis at level 1, such that:

$$\sum_{k \in K} T^k_l = 0 \text{ or } 1 \quad \text{if } l \in P$$

(giving the lth basic variable value 1 or 0),

$$\sum_{k \in K} T^k_l = 0 \text{ or } -1 \quad \text{if } l \in \bar{P},$$

(rendering the lth basic variable 0 or 1).

For instance, bringing in $x(2,3)$ at level 1 saves 9, renders $s(2,3)$ infeasible $(= -1)$, which has to be corrected by bringing in at 1 either $x(2,2)$ (*costs* $10 - 9 = 1$) or $s(2,2)$ (*saves* $9 - 1 = 8$). Both changes render the problem feasible, therefore yield neighbor vertices. Choosing the improving vertex, we get $x(2,3) = s(2,2) = 1$ and

$$y = \begin{bmatrix}1\\1\\0\end{bmatrix}, \quad x = \begin{bmatrix}1 & 1 & 0\\0 & 0 & 1\\0 & 0 & 0\end{bmatrix}, \quad s = \begin{bmatrix}0 & 0 & 1\\1 & 1 & 0\\0 & 0 & 0\end{bmatrix}, \quad \bar{y} = \begin{bmatrix}0\\0\\1\end{bmatrix}$$

with a cost of $13 - 8 = 5$.

Property 9. Given the set P of variables at 1, one can also complete the basis by first adding columns corresponding to $\bar{y}(i)$ at zero, then only adding s columns as needed.

We shall can this basis the $y - s$ *canonical basis.* Essentially, the procedure is the following: bring the identitiy matrix to the top of the basis as before, which corresponds to placing first the rows $\Sigma_1, \ldots, \Sigma_m$ then for each $\bar{y}(i)$ at 1 ($y(i) = 0$, $x(i,j) = s(i,j) = 0\ \forall j$) choose a nonbasic pair $(i, j_0(i))$ (or ij_0 when not ambiguous) as before. Then, for each $\bar{y}(i)$ at 0 ($y(i) = 1$, the plant is open) some $x(i,j)$ must be 1 in a basic solution since not all $s(i,j)$ can be simultaneously positive and thus basic, as $\bar{y}(i)$ also must come from one $*ij$. Choose one index j for which $x(i,j)$ is 1 and render $s(i,j)$ nonbasic. Then complete with columns corresponding to $s(i,j) = 1$ (possible only with $\bar{y}(i) = 0$ basic and $x(i,j)$ nonbasic). This is still a triangular procedure. Finally complete with $s(i,j) = 0$.

Table 3: (B, T^N).

	x_{11} 12 23	$\bar{y}_3$	1 2	s_{13} 21 22 12 32 33	x_{31}	s_{31} s_{11} 23	x_{13} 21 22 32 33
$\Sigma 1$	1				1		1
2	1						1 1
3.	1						1 1
*31		1			1	1	
11	1		1		−1*	1	−1*
23	1		1			1	−1* −1*
13			1	1	1*	−1*	1 1*
21			1	1		−1*	1* 1 1*
22			1	1		−1*	1* 1 1*
12	1		1	1	1*	−1*	1* −1* −1*
32		1		1	−1*	−1*	1
33		1		1	−1*	−1*	1

Example (cont.). The starred entries in Table 3 belong to the updated tableau.

$$d^N = [9\ 1\ 1\ 1\ 8\ -1\ 9\ 0\ -1].$$

Two columns are candidates to enter the basis: $x21$ and $x33$ (in slightly simplified notation) at a saving of 1, but both pivot steps would be degenerate.

If one brings in $x21$ at level 1, $s12$ becomes -1, which can be corrected by bringing in at 1:

$s11$ at no improvement, but this is a feasible neighbor vertex.

$x22$ at a cost of $9 - 1 = 8$, which is also feasible, so that there is no need to pursue this combination further.

$x32$ at a saving of 1, but this renders $s32$ infeasible, which can be corrected by setting $x31$ or $s31$ at 1, with no improvement left. $s31$ would yield a feasible point, $x31$ would create an infeasibility which could not be corrected at a saving.

If one brings $x33$ in at 1, $s33$ becomes -1, $s31$ would completely correct it at no saving, $x31$ would cost $9 - 1 = 8$ and no further saving is possible. The solution is therefore *optimal* in integer variables.

Property 10. Given an integer feasible solution to SPLP, one can also define a triangular basis (so-called $y - x - y - s$ *canonical basis*) having the following columns: all $\bar{y}(i)$'s at 1, some $x(i,j)$'s at 0 for closed plants, all $x(i,j)$'s at 1, all $\bar{y}(i)$'s at 0, all $s(i,j)$'s at 1, some $s(i,j)$'s at 0.

We suggest that constructing such a basis as follows, one may achieve the goal of arriving at a relatively small number of negative reduced costs (i.e., to position oneself in a sense close to an optimal solution, in order to finish via relatively few and simple block pivots).

(1) Place first the columns corresponding to $\bar{y}(i) = 1$ ($x(i,j) = s(i,j) = 0\ \forall j$) and associate with each i a row $*(i, j_0)$, where $(x(i, j_0(i)), s(i, j_0(i)))$ will be a nonbasic pair such that $f(i) + c(i,j) - f(i(j)) - c(i(j), j)$ is maximal for $j = j_0(i)$.

(2) For each i with $\bar{y}(i) = 1$, for each pair $(x(i,j), s(i,j))$, $j = j_0$, if $c(i,j) \geq f(i(j)) + c(i(j), j)$ make $x(i,j)$ nonbasic, otherwise make $s(i,j)$ nonbasic and add row $*ij$ and column $x(i,j)$ to the subbasis.

(3) add rows Σj and columns $x(i(j), j)$.

(4) For $\bar{y}(i) = 0$ ($y(i) = 1$, one $x(i,j)$ at least is 1), choose one $j_1(i)$ such that (a) $x(i, j_1) = 1$ and (b) the increase in cost for shipping from another plant is maximal over $\{j \mid x(i,j) = 1\}$ for $j = j_1$. Note that for different i, we'll get different $j_1(i)$ as there is only one $x(i,j)$ at 1 per column. We can therefore talk of $j_1^{-1}(j)$ with the convention that $f(j_1^{-1}(j))$ is $f(i)$ if $j = j_1(i)$ and is 0 if there is no i such that $j = j_1(i)$. Then add row $*ij(i)$ and column $\bar{y}(i)$ to the submatrix.

(5) Complete with the slacks at 1, and then some slacks at 0. Then, if we call $B_0(i) = \{j \mid x(i,j)$ is basic, $x(i,j) = 0\}$, we have the following properties:

$$d(s_{ij_0(i)}) = f_i - c_{i(j_0),j_0} - \sum_{j \in B_0(i)} f_{i(j)} + \sum_{j \in B_0(i)} (c_{ij} - c_{i(j),j}),$$

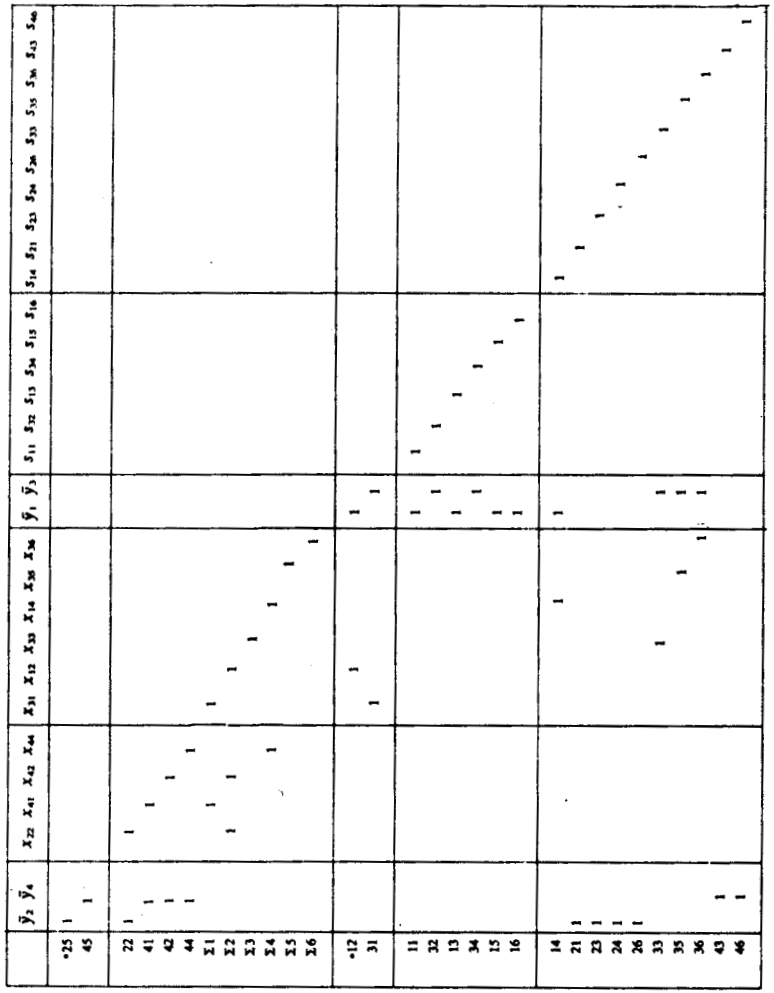
$$d(x_{ij_0(i)}) = c_{ij_0} + d(s_{ij_0(i)}),$$

for $j \in B_0(i)$, $d(s_{ij}) = f_{j_1^{-1}(j)} + c_{ij} - c_{i(j),j}$,

for $x(i,j)$ nonbasic with $s(i,j) = 1$, $d(x_{ij}) = c_{ij} - [c_{i(j),j} + f_{j_1(j)}]$,

$$d(s_{ij_1(i)}) = f_i \text{ (all other } s(i,j) \text{ are basic).}$$

Table 4. Original basis (in triangular form).



Example.

$$f = \begin{bmatrix} 5 \\ 10 \\ 10 \\ 10 \end{bmatrix}, \quad c = \begin{bmatrix} 3 & 2 & 4 & 2 & 20 & 5 \\ L & 1 & 3 & L & 1 & L \\ 1 & L & 1 & L & 2 & 2 \\ 8 & 5 & 5 & 1 & 3 & 4 \end{bmatrix}, \quad x = \begin{bmatrix} 0 & 1 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 1 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix}, \quad y = \begin{bmatrix} 1 \\ 0 \\ 1 \\ 0 \end{bmatrix},$$

L stands for a very large number. Tables 4 and 5 display the original basis and updated nonbasis.

Table 5. Updated nonbasis (asterisked entries introduced in inversion process).

Basis	x_{25}	s_{25}	x_{45}	s_{45}	x_{22}	41	42	44	x_{11}	21	x_{32}	x_{13}	23	43	x_{24}	34	x_{15}	x_{16}	26	46	s_{12}	31
$\bar{y}_2$	1	1																				
$\bar{y}_4$			1	1																		
x_{22}	−1*	−1*			1																	
41			−1*	−1*		1																
42			−1*	−1*			1															
44			−1*	−1*				1														
x_{31}			1*	1*		−1*			1	1												
12	1*	1*	1*	1*	−1*		−1*				1											
33												1	1	1								
14			1*	1*				−1*							1	1						
35	1		1														1					
36																		1	1	1		
$\bar{y}_1$	−1*	−1*	−1*	−1*	1*		1*				−1*										1	
$\bar{y}_3$			−1*	−1*		1*			−1*	−1*												1
s_{11}	1*	1*	1*	1*	−1*		−1*		1		1*										−1*	
32			1*	1*		−1*			1*	1*	1											−1*
13	1*	1*	1*	1*	−1*		−1*				1*	1									−1*	
34			1*	1*		−1*			1*	1*					1							−1*
15	1*	1*	1*	1*	−1*		−1*				1*						1				−1*	
16	1*	1*	1*	1*	−1*		−1*				1*							1				
s_{14}	1*	1*	0*	0*	−1*		−1*	1*			1*				−1*	−1*					−1*	
21	−1*	−1*								1												
23	−1*	−1*											1									
24	−1*	−1*													1							
26	−1*	−1*																	1			
33			1*	1*		−1*			1*	1*		−1*	−1*	−1*								−1*
35	−1*		0*	1*		−1*			1*	1*							−1*					−1*
36			1*	1*		−1*			1*	1*								−1*	−1*	−1*		−1*
43			−1*	−1*									1									
46			−1*	−1*																1		
d^N	3	4	5	4	6	3	2	1	−8	L	L	3	2	4	L	L	18	3	L	2	5	10

4. Enumeration

The SPLP behaves relatively well under enumeration. Enumerative codes often start with all plants open, so that a "forward step" in the search consists of closing a plant. (E.g., see [24].) More generally, one starts at a point with some plants fixed open ($i \in E$, $y(i) = 1$), some fixed closed ($i \in C$, $y(i) = 0$), some free but tentatively open ($i \in F1$, $y(i) = 1$), some free but tentatively closed ($i \in F2$, $y(i) = 0$).

4.1. State enumeration [20]

By the "state" at node ν, we mean a partitioning of the index set into $E, C, F1, F2$. With the state one associates a solution (z^ν, y^ν, x^ν) by:

$$\begin{aligned} &y^\nu(i) = 1 \quad \text{for } i \in E + F1, \\ &y^\nu(i) = 0 \quad \text{for } i \in C + F2, \\ &z^\nu = \sum c(i(j), j), \\ &x(i(j), j) = 1, \\ &x(i, j) = 0, \qquad i \neq i(j), \\ &i(j)\text{: } c(i(j), j) = \min\{c(i, j)\} \text{ over } E + F1. \end{aligned} \tag{4.1}$$

Consider the problem SPLPD dual to (2.1):

$$\begin{aligned} &\max z_D = -\sum w(i, j) + \sum v(j), \\ &c(i, j) + w(i, j) - v(j) \geqslant 0, \\ &w(i, j),\ v(j) \geqslant 0, \\ &i \text{ over } E^\nu \text{ and } F1^\nu, \quad j \text{ over } J = 1, 2, \ldots, n. \end{aligned} \tag{4.2}$$

In terms of z and z^* (a known upper bound on z), one has the Benders inequality:

$$\sum_{F1^\nu} \left(-f(i) + \sum w(i, j) \right) \cdot y(i) + \sum_{F2^\nu} \left(f(i) - \sum w(i, j) \right) \cdot y(i) \leqslant z^* - z. \tag{4.3}$$

In [20, 24], the coefficients of the $y(i)$ (multiplied by -1) were called "global gain functions", $g(i)$, and play a central role in curtailing and guiding the search.

4.2. Strengthening the gain functions

It is important to have the $g(i)$ as small as posible. E.g., $g(i) \leqslant 0$ permits the fixing of $y(i)$.

It is interesting that there is a great latitude in choosing the $w(i, j)$ of (4.2), and with them the $g(i)$. One good and not completely obvious choice of the dual variables is: Let

$$\begin{aligned} &c(\hat{\imath}(j), j) = \min\{c(i, j) \mid i \in E^\nu + F^\nu - \{i(j)\}\}, \\ &v^\nu(j) = \begin{cases} c(\hat{\imath}(j), j) & \text{if } c(\hat{\imath}(j), j) \geqslant c(i(j), j) \\ c(i(j), j) & \text{otherwise,} \end{cases} \\ &w^\nu(i, j) = \max\{0, v(j) - c(i, j)\}. \end{aligned} \tag{4.4}$$

Then one has, correspondingly:

$$\begin{aligned} &g(i) = f(i) - \sum_{j:\, i = i(j)} \max\{0, c(\hat{\imath}(j), j) - c(i(j), j)\}, \quad i \in F1, \\ &g(i) = -f(i) + \sum_j \max\{0, c(\hat{\imath}(j), j) - c(i, j)\}, \quad i \in F2. \end{aligned} \tag{4.5}$$

As in [24], one can also define "local" gain functions to aid in both curtailment of search and strategy.

Now, it is quite clear that the reduced costs of any LP tableau associated with the variables of a state problem at node ν have the properties of gain functions. There is then substantial interest in generating the various bases of Section 3 for node ν, and utilizing the reduced costs in the enumeration.

More fundamentally, it is clear that the gain functions as used in the past only exploit a limited portion of the updated LP tableau. Having the entire updated tableau at one's disposal, at a cost which is relatively modest given the nice properties of "canonical bases", should permit substantial improvements.

In a sense, it provides a grasp of the LP polytope, e.g. by defining the edges leading away from the state point. With some ingenuity, an improvement of enumerative procedures should be attainable.

References

[1] Dj.A., Babayev, Comments on a Note of Frieze, Math. Programming, 7 (1974) 249–252.

[2] E. Balas and M.W. Padberg, On the set-covering problem, J. ORSA, 20, (1972) 1152–1161.

[3] E. Balas and M.W. Padberg, On the set-covering problem II, an algorithm, Mgmt. Sciences Research Rep. 295, Carnegie Mellon Univ., 1972.

[4] M.L. Balinski, Fixed cost transportation problems, Naval Res. Logistics Quarterly, 8 (1961) 41–54.

[5] M.L. Balinski and K. Spielberg, Methods for integer programming: Algebraic, combinatorial and enumerative, in J. Aronofsky, ed., Progress in Operation Res., Vol. 3, (Wiley, New York, 1969) pp. 195–292.

[6] J.F. Benders, Paritioning procedures for solving mixed-variables programming problems, Num. Math., 4 (1962) 238–252.

[7] O. Bilde and J. Krarup, Sharp lower bounds and efficient algorithms for the simple plant location problem, Research Report #75/5, Inst. of Datology, Univ. of Copenbagen, 1975 (based on report in Danish of 1967).

[8] P. Broise, P. Huard and J. Sentenac, Decomposition des Programmes Mathematiques (Dunod, Paris, 1968).

[9] V.P. Cherenin, Reshenie nechotorych combinatornych zadach optimalnogo planirovania metodom posledovatelnych raschetov (Solving some combinatorial problems of optimal planning by the method of successive calculation), in Proc. of the conference of experiences and perspectives of the application of mathematical methods and electronic computers in planning, Mimeograph, Novosibirsk, 1962.

[10] G. Cornuejols, M.L. Fischer and G.L. Memhauser, An analysis of heuristics and relaxations for the uncapacitated location problem, Cornell Univ., Techn. Report #271, Aug. 1975.

[11] G.B. Dantzig and Ph. Wolfe, Principle of decomposition for linear programs, J. ORSA, 8 (1960) 101–111.

[12] P.S. Davis and T.L. Ray, A branch-bound algorithm for the capacitated facilities location problem, Naval Res. Log. Quart., 16 (1969) 331–344.

[13] M.A. Efroymson and T.L. Ray, A branch-bound algorithm for plant location, J. ORSA, 14 (1966) 361–368.

[14] A.M. Frieze, A cost function property for plant location problems, Math. Programming, 7 (1974) 245–248.

[15] A.M. Geoffrion, Lagrangean relaxation for integer programming, Math. Programming Study 2 (1974) 82–114.

[16] A.M. Geoffrion and G.W. Graves, Multicommodity distribution system design by Renders decomposition, Management Sci., 20 (1974) 822–844.

[17] F. Glover, Compact LP bases for a class of IP problems, Report #75-18, Mgmt. Sc. Report Series, Grad. School of Bus. Adm., Univ. of Col., April 1975.

[18] R.S. Garfinkel, A.W. Neebe and M.R. Rao, An algorithm for the M-median plant location problem, Working Paper #7312, Grad. School of Management, Univ. of Rochester, May 1973.

[19] G.W. Graves and R.D. McBride, The factorization approach to large-scale linear programming, Working Paper #208, Western Mgmt. Sc. Inst., UCLA, Aug. 1973.

[20] M. Guignard and K. Spielberg, Search techniques with adaptive features for certain integer and mixed-integer programming problems in: A.J.H. Morrell, ed., Information Processing 68, Volume 1 (North-Holland, Amsterdam, 1968) pp. 238–244.

[21] M. Guignard and K. Spielberg, Triangular structure in uncapacitated plant location, ORSA/TIMS Meeting 1974, San Juan, Puerto Rico.

[22] R.E. Marsten, An algorithm for finding almost all of the medians of a network, Report, Center for Math. Studies in Econ. and Mgmt. Sc., Northwestern Univ., Nov. 1972.

[23] L. Schrage, Implicit representation of generalized variable upper bounds in linear programs, Report #7543, Dept. Econ. & Grad. School of Business, Univ. Chicago, Oct. 1975.

[24] K. Spielberg, Plant location with generalized search origin, Management Sci., 16 (1969) 165–178.

[25] K. Spielberg, Algorithms for the simple plant-location problem with some side conditions, J. ORSA, (17) (1969) 85–111 (based on IBM New York Sci. Centre Report #2909, May 1967).

[26] H.P. Williams, Experiments in the formulation of integer programming problems, Math. Programming Study 2 (1974) 180–197.

OPERATIONS RESEARCH
VERFAHREN
METHODS OF
OPERATIONS RESEARCH

XXV

Herausgeber · Editorial Board RUDOLF HENN, KARLSRUHE
PETER KALL, ZÜRICH
BERNHARD KORTE, BONN
OLAF KRAFFT, AACHEN
WERNER OETTLI, MANNHEIM
KLAUS RITTER, STUTTGART
JOACHIM ROSENMÜLLER, KARLSRUHE
NORBERT SCHMITZ, MÜNSTER
HORST SCHUBERT, DÜSSELDORF
WALTER VOGEL, BONN

Sonderdruck

FIRST SYMPOSIUM ON OPERATIONS RESEARCH
University of Heidelberg, September 1 – 3, 1976

Mitherausgegeben von: ADOLF ANGERMANN, HEIDELBERG
Coedited by: ROLF KAERKES, AACHEN
KLAUS-PETER KISTNER, BIELEFELD
KLAUS NEUMANN, KARLSRUHE
Teil I BURKHARD RAUHUT, AACHEN
FRANZ STEFFENS, MANNHEIM

VERLAG ANTON HAIN · MEISENHEIM AM GLAN

Propagation, Penalty Improvement and Use of Logical Inequalities in Interactive (0,1) Programming

M. Guignard, Philadelphia and K. Spielberg, Stamford

0. Introduction

We deal with the 0-1 integer programming problem

$$\min \quad z = c \cdot y$$
$$\text{s.t.} \quad A \cdot y \leq b,$$

the components of y being 0 or 1 only.

We are concerned with an experimental, partially interactive system which is designed so as to cope with fairly difficult zero-one programs in an adaptive fashion. The system is of experimental nature only, and not accessible to users outside of IBM.

Section 1 of the paper describes a special manner of branch and bound (BB) programming, emphasizing propagation on zero-cost branches.

In Section 2 we concentrate on various ways of generating logical (preferred variable) inequalities. Section 3 will present two methods for penalty improvement based on the findings of sections 1 and 2.

In Section 4 we describe the uses of interactivity in the resolution of difficult 0-1 problems. Section 5 will present experimental results.

1. Propagation and Minimal Inequality BB Programming (ref.<1>)

The basic idea of the special BB algorithm is that of fixing a variable at the value it already has in the current LP solution and looking at the resulting situation. This is particularly interesting when the variable is "preferred", with the propagating value its "suggested" value from a minimal perferred inequality, that is when fixing the variable at the opposite value will force at least one of a set of other variables in the preferred set to take on a particular value. See the appendix for pertinent definitions.

Consider the following example. Solving the relaxed linear program (LP), the problem with "free" y-components between 0 and 1, we find that y_1 is 0, currently. The penalty for setting it to 1 is, say p_1. On the other hand, suppose we have found that the preferred inequality

$$(1-y_1) + y_2 + (1-y_3) \geq 1 .$$

has to be satisfied for any improved integer solution of the constraints. This "suggests" the values 0 for y_1 and y_3, and 1 for y_2; y_1 is a "propagation candidate".

The "alternate" node (alternative to the propagation on variable y_1) will therefore contain the following information:

since y_1 will not be zero, either y_2 should be 1 or y_3 should be 0. We can therefore store the alternate choice as a pair of branches: $y_2=1$ and $y_3=0$. The penalty of this "alternate node" is thus to be computed in terms of p_1 on the one hand, and of the penalties associated with y_2 and y_3 on the other.

Figure 1 illustrates this scheme compared with normal BB bookkeeping (assuming that branching on nonbasic as well as basic variables is allowed).

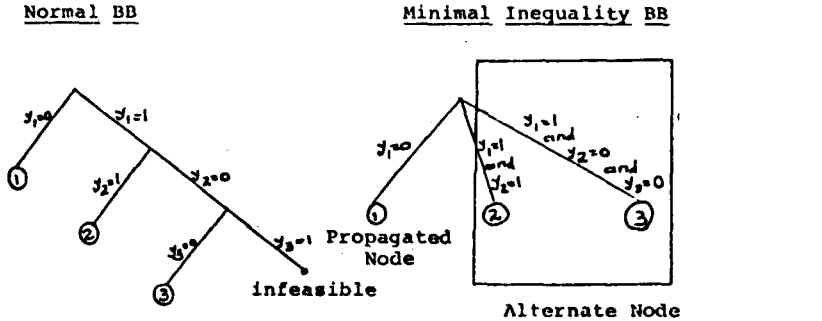


Fig. 1 $\quad (1-y_1) + y_2 + (1-y_3) \geq 1$

If no inequality suitable for propagation should be available, a natural default would be to use the trivial inequality

$$y_1 + (1-y_1) \geq 1$$

as a "degenerate" minimal preferred variable inequality, and to store the alternate node $y_1=1$ with the penalty p_1.

Definition

A variable y(i) is said to be (potentially) a propagating variable if its value in the optimal solution $\bar{y}$ to the current LP is integral.

In that case, indeed, adding the constraint $y(i) = \bar{y}(i)$ to the current LP constraints would not change its optimal value. One often has a wide choice among propagation candidates, usually making the selection so as to have the highest possible penalty for the alternate node.

Notice that a propagating variable can be basic as well as nonbasic. In either case the penalty for the propagation is 0 and one can compute the penalty for the alternate branch in the usual ways or as described in section 3.

For certain problems, systematic propagation on nonbasic variables with large reduced costs (or linked to other variables with large penalties via minimal inequalities) seems to be a very promising approach. Knapsack problems, in particular, seem to lend themselves well to propagation. Consider a relatively difficult 40 variable knapsack problem taken from the literature <2>. Table 1 gives comparative data for a normal BB aproach (branching on basic variables only) and for our minimal inequality BB code.

	Number of LP's	Number of Propagations	Total Number of Nodes
Normal BB	99	0	99
Min. Ineq.BB	5	30	35

TABLE 1

As a second example for BB propagation, we consider a (28,35) problem taken from <3>. Solving the initial LP, one obtains a fractional solution with objective function z = 521.6. Our system could resolve the problem by means of Gomory-Johnson cuts <4>. A normal BB algorithm, such as implemented in standard production codes, would require about 30 to 50 nodes for the BB resolution. By contrast, we can resolve the problem with only one LP in a number of ways. In Fig. 2 we outline the initial phase of a propagation phase:

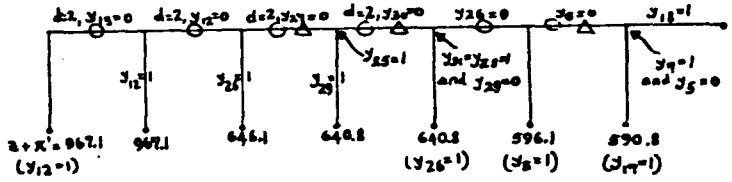


Fig.2

Store 11 alternate nodes with objective function values decreasing from 967.1 to 551.3 .
Then employ State Enumeration for all pending nodes, <5> .

In Fig. 2, a circle on a branch denotes propagation, a triangle denotes contraction. (A contracting branch is one

for which one may predict, in advance, that the degree d of the system, i.e. the size of its minimal inequality system, will be decreased by the branch). In the example, the degree is always two, i.e. the inequalities are always strong (characterizing the problem as "easy"). A contracting branch, therefore, leads to the fixing of at least one variable (by "reduction", i.e. fairly obvious tests on the inequalities). The fixed variables and the branches and penalties are indicated in the diagram.

The propagation phase is carried on, interactively for example, until the lower bounds on the obj. function values stored with the alternate nodes appear to become small, i.e. uninteresting. Then one can complete the solution process by either:

(i) entering a normal BB mode (within which one may still continue to employ propagation procedures whenever new nodes are brought into storage from the BB tree)

(ii) entering a (state) enumeration phase, i.e. an enumeration procedure employing preferred variables and starting from a good starting point ("state").

The (28,35) problem can be finished with the optimal solution z=550, and optimality proved, after processing of 33 enumeration nodes (no LP used) if the enumeration starts from a 0 origin (the fixed variables of Fig. 2 taken into account, of course), or after 9 nodes of enumeration (with a 1-level look-ahead) if the search starts from a point determined by rounding of the initial LP solution (everything above .5 to 1, everything below .5 to 0).

Alternatively, also, one may proceed with normal BB programming. In that case, if one employs reduction procedures and continues to use propagation, one can also resolve the problem without solving another LP. (see <1>).

Examples of Minimal Preferred Variable Inequalities

Minimal preferred inequalities (m.p.i.'s) are basic in much of the paper. This may be a good place to give examples for some nontrivial problems.

As first example we consider a (37,74) problem from <3>, which is difficult for any algorithm. The printout first lists the degrees (named PI in the listing) of the 37 original rows, augmented by 9 Benders inequalities (see section 2). The problem is difficult enough, but the fact that most of the degrees, especially those of the original constraint set, are two, is an encouraging sign for using the methods of this paper. The set of m.p.i.'s is listed next, and is also encouraging.

(3 4) stands for $y_3 + y_4 \geq 1$, (-31 -66) for $-y_{31} - y_{66} \geq -1$

(39 -63) for $y_{39} - y_{63} \geq 0$ etc.. .

There are coefficients of opposite sign, which is usually beneficial in itself, and there are some double contracting indices, e.g. 31 (it appears both +31 and as -31; a branch on variable 31 will reduce the degree by one, both on the forward step and on the eventual backward step of a search algorithm).

```
        PREPVARS
PI 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 2 4 2 2 3 2 3 2 3 2 3 6
2 5 3 3 2 2 13 2 23 7 60 24 13 13 6 6 2

PREPVARS
(  3   4 )(  3   5 )(  3   7 )(  3  15 )(  3  17 )(  3  12 )
(  4   5 )(  4   7 )(  4   8 )(  4   9 )(  4  18 )(  4  19 )
(  4  25 )(  4  26 )(  4  34 )(  4  39 )(  4  40 )(  4  48 )
(  4  52 )(  4  57 )(  4  62 )(  4  65 )(  4  70 )(  4  74 )
(  7   9 )(  8   9 )( 10  35 )( 10  44 )( 12  13 )( 12  14 )
...
...
( -23  39 )( 24  25 )( 24  26 )( 24  27 )( 24  28 )( 24  29 )
...
...
( -31 -66 )( 31 333 )( 31  66 )( 31  39 )( 34  38 )( 34  37 )
...
...
( 39 -63 )( 39 -66 )( 39  43 )( 39  46 )( 39  47 )( 39  44 )
...
...
```

The second example is a (20,28) problem (from <12>, but inadvertently modified). It is inordinately difficult for its size, and its resolution can easily take from 100 to several hundred LP's, even with cutting planes and reduction. The penalties are small, the gap between LP objective function and integer objective function (for optimality) is quite large and the degree is 5, somewhat large.

```
        PREPVARS
PI 13 8 7 12 9 5 10 10 13 13 8 10 10 9 13 7 5 7 9

PREP.VARS.
( 11 -17 -18  24 -28 )( -2   7  10 -20  28 )( 11 -17  23  24 -28 )
...
...
```

To improve the situation we added Benders Inequalities. The second set of m.p.i.'s is better and it will aid considerably in the resolution of the problem. We have not pressed for a final solution with the present code, but we have found the solution and ascertained optimality in the past in about 80 LP's, using a state enumeration algorithm and cutting planes.

PI 13 7 7 12 9 13 6 9 10 12 12 7 9 9 8 10 7 5 6 8 6 9 12 9
13 6 6 6 5 6 6 2 9 4 4 3 2 2 2 2 2 2 3 2 2 2 2 2 2 2 2 2 2
2 2 2

PREF. VARS

(	7	-19	)(	4	7	)(	-7	-17	)(	119	21
(	8	-17	)(	7	-21	)(	-17	19	)(	-5	7
(	-18	-20	)(	7	-20	)(	-7	18	)(	-17	-21
(	-20	28	)(	-7	19	)(	-3	-20	)(	7	-26
(	-20	21	)(	-17	-20	)(	-17	-26	)		

)(	8	-20	)
)(	1	-20	)
)(	7	28	)
)(	1	7	)

2. Generation of Strong Preferred Inequalities (see<6,7>)

Logical inequalities contain potentially useful information about the structure of the original problem. In a sense, they can be used as diagnostic tools for determining the problem characteristics, especially when interactivity permits the user to apply the tool in a systematic, yet flexible, manner.

Aside from giving insight into the structure of the problem, they can help in the branching choices, in penalty computation, in the fixing of variables, in heuristic searches for feasible solutions, etc..

There are ways in which one may facilitate the generating of meaningful logical inequalities if the original system of constraints is relatively intractable for reduction, i.e. if its degree (the degree of the minimal inequality system which can be derived from it) is large.

Such methods of improving the degree of a given system are almost always related to the generation of new inequalities. We have implemented two approaches:

2.1 Benders Inequalities via Guesses (see <10> for B.I.'s)

We provide the terminal user with the opportunity of entering (partial) guesses at solutions. The corresponding LP is solved with those variables fixed and Benders Inequalities of type 1 (LP feasible) or type 2 (LP infeasible) are generated. We have found them to be usually of low degree.

Instead of generating B.I.'s as one processes the tree of solutions, one can make an attempt at finding such inequalities a priori, i.e. at the onset, and derive useful information about the problem before entering into the enumeration or BB phase. Minimal preferred inequalities are then generated in the usual manner.

We have found this very useful for certain difficult problems. It allows us, e.g., to fix 15 variables in the difficult (37,74) problem of <3>, 6 variables in the (5,39) problem of <11>, and 3 variables in the troublesome (20,28) problem modified from <12>.

2.2 Combination of Constraints

Minimal inequalities are generated from individual constraints, and information that is contained in the totality of constraints is more difficult to obtain. Some attempts at combining constraints have been reported, e.g. <14>. We made some attempts, aided by interactivity, at combining constraints and B.I.'s.

For instance, suppose we deal with the knapsack problem

$$\min \quad \Sigma c(j) \cdot y(j) = z$$

$$\text{s. t.} \Sigma k(j) \cdot y(j) \leq L$$

$$y(j) \ 0 \text{ or } 1 \quad ,$$

all k(j), c(j) and L positive.

As soon as a feasible solution y* has been found (with z=z*), one can of course add the constraint $z \leq z^*$, but its coefficients would also be positive. B.I.'s of type 1, on the other hand, usually have mixed signs, + for variables at 1 in the optimal solution, - for variables at 0. The variables with negative coefficients can then often be eliminated by suitable linear combinations of the B.I. and the problem constraint.

Some scheduling problems have constraints with all coefficients negative in $\leq$ form (certain requirements must be met). It is then fairly easy to form useful combinations with the constraint $c \cdot y \leq z^*$.

One can also combine logical inequalities (here preferred inequalities) among themselves, or with the original constraint set, possibly augmented by B.I.'s. This does not necessarily mean using only positive combinations of p.i.'s. Suppose, e.g., that several (different) constraints yield the following logical inequalities:

$$y_1 - y_2 \geq 0$$
$$y_2 - y_3 \geq 0$$
$$-y_1 + y_3 \geq 0$$

Adding these inequalities yields no information whatever. Notice, however, that they imply the following:

$$y_1 \geq y_2, \quad y_2 \geq y_3, \quad y_3 \geq y_1$$

I. e., they imply $y_1 = y_2 = y_3$. While this does not permit the fixing of variables, it represents a reduction in problem size which is probably equally important.

Considerations such as these do not depend on equal degree for the logical inequalities and therefore lead us to questions related to but outside the realm of minimal preferred inequalities.

We can give here an initial analysis of what can probably be accomplished. A preferred inequality (p.i.) of degree d excludes one combination of values for the variables of the preferred set, i.e. it cuts off 2^{n-d} vertices of the hypercube. If k p.i.'s remove 2^{n-1} vertices in one face of the hypercube, one variable can be fixed, i.e. one will now concentrate on the other 2^{n-1} points of the opposite face.

E.g., in R^3 ,

$$y_1 + y_3 \geq 1 \qquad \text{(P1)}$$

eliminates $2^{3-2} = 2$ points, namely (0 0 0) and (0 1 0), i.e. all points with y_1 and y_3 at 0, while

$$-y_1 + y_2 + y_3 \geq 0 \qquad \text{(P2)}$$

eliminates $2^{3-3} = 2^0 = 1$ point, namely (1 0 0), and

$$-y_1 - y_2 + y_3 \geq -1 \qquad \text{(P3)}$$

eliminates $2^{3-3} = 2^0 = 1$ point, (1 1 0). See Fig. 3:

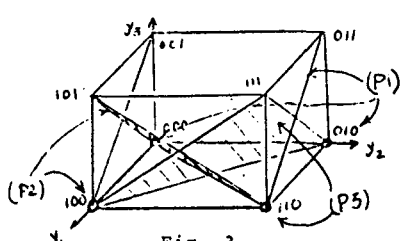


Fig. 3

All three inequalities combined imply that y3 should be one, the lower horizontal face of the cube having been eliminated. Notice that in such a case all coefficients of the variables to be fixed (here y3) have to be of the same sign, the eliminated value being of course the same.

After having checked that there is no redundancy (such as y1 + y2 ≥ 1 and y1 + y2 + y3 ≥ 1, where some information is duplicated, the lst inequality eliminating the two points (0 0 1) and (0 0 0), the second only eliminating one point (0 0 0);if both were generated, the second could be dropped), and having discounted the points eliminated several times (e.g., (0 0 0 0) is eliminated by both y1 + y2 + y3 ≥ 1 and y1 + y2 + y4 ≥ 1) but neither inequality eliminates the other), then if 2^{n-1} different points have been eliminated with the same value for one variable, that variable can be fixed at the other value.

The difficulty of course consists in avoiding duplications in the counting. But consider the previous example as an illustration of how this can be done, at least sometimes: y3 appears with the same sign, namely +, in all three inequalities. Its suggested value thus is 1. Can 0 be feasible? The signs of the coefficients of y1 and y2 tell us that the inequalities eliminate different points. A simple count ($2+1+1=4=2^2$) tells us that y3 cannot be 0, and thus ought to be fixed at 1.

3. Penalty Improvement

Whether one branches on basic or nonbasic variables, it is obviously essential to have penalties as accurate and restrictive as possible. We propose essentially two ways for improving the usual penalty computations.

3.1 Penalty Improvement via Minimal Inequality BB

Suppose that having chosen a m.p.i. of size d

$$\sum_{i=1}^{d} \tilde{y}(i) \geq 1, \text{ where } \tilde{y}(i)=y(i) \text{ or } = 1-y(i) ,$$

we choose the branches

$$y(1)=\varepsilon(1),\ y(2)=\varepsilon(2),\ \ldots,\ y(d)=\varepsilon(d),\ \varepsilon(i)=0 \text{ or } 1,\ i=1,2,\ldots d,$$

the first branch y(1) being propagating.

Assume that the penalties p(1)', p(1)', p(2),p(2)',..., p(d),p(d)' have been calculated in the usual manner, for fixing y(1),...,y(d) at values ε(i) and 1-ε(i). If these penalties are additive (which will be the case, for instance, if they are all reduced costs for nonbasic variables), then one incurs increasingly strong overall penalties as one proceeds in the execution of the branches from left to right (see Fig. 4),the left variables taking on their "non-suggested" values when one proceeds to variables on their right. The ordering of execution will influence the necessity of proceeding with the rightmost branches, and one hopes to avoid a large number of the more constrained cases.

Fig. 4

3.2 Penalty Improvement via M.P.I.'s

3.2.1. Suppose a m.p.i. indicates that if y(i), whose nonbasic values is $\hat{y}(i)$ in the current LP solution $\hat{y}$, is changed to $1-\hat{y}(i)$, at least one of the set P of nonbasic variables would have to change from $\hat{y}(j)$ to $1-\hat{y}(j)$, thus incurring the penalty p(j), then the true penalty for modifying the value of y(i) is actually:

$$p(i) + \min_{j \in P} p(j)$$

3.2.2. Suppose a m.p.i. suggest that at least one of a set P of NONBASIC variables has to be complemented from its current value $\hat{y}(i)$ in the LP optimum to $1-\hat{y}(i)$, incurring a penalty p(i), then the penalty for any branch is at least $\min_{i \in P} p(i)$.

3.2.3 Suppose one wants to compute the penalty for rounding up or down a basic fractional variable y(i). Instead of simulating a pivot step relative to the top row of the current LP tableau, one can simulate a pivot step with respect to the row

$$z \geq z^0 + \Sigma p(k)\tilde{y}(k)$$

where p(k) is the improved penalty incurred for complementing y(k) .

4. Interactivity

Interactivity can play an important role in the resolution of difficult LP problems. Being able, e.g., to prompt the system for m.p.i.'s, to enter guesses in accordance with them, to try the generation of feasible solutions via heuristics after some progress has been made in the fixing of variables, being able to switch from BB programming to enumeration and back, being able to refuse storing alternate nodes with poor penalties, being able to combine inequalities intelligently when it would not be possible to do so in a systematic fashion, being able to orient the search when one has gained some insight into the problem and its structure, has enabled us to solve some difficult problems with surprisingly little work.

Enumeration nodes cost very little compared with solving LP's. Reduction is moderately time-consuming, but is essential if the combinatoric difficulties are to be moderated.

It is quite difficult to render a computer code sensitive enough to do all the things which can be done interactively in automatic fashion, at any rate without overburdening it with a mass of parameters. The interactive approach seems to be a valid one. One of its important aspects for the model builder is that the interactive process turns into a learning process. No two problems behave in quite the same manner. Some respond to cuts, others to reduction. Heuristic searches sometimes do very well, at other times yield little. Clever guesses can yield very strong B.I.'s for otherwise loosely constrained problems. The interactive procedure shows promise of remedying, at least partly, some of the bad characteristics of difficult mixed integer programming problems.

5 . Computational Results

We give a table of results for seven problems of moderate difficulty (at least in a relatively slow interpretive APL environment).

The results above the central horizontal line are essentially taken from <13>, which represents a rather good enumerative algorithm with linear programming auxiliary problems, except for the 99 LP's in the case of the knapsack problem which are ours.. The entries below the line are for our experiments.

Problem size								
m....(rows).....	5	12	28 *		1	10	10	15
n....(columns).....	39	44	35		40	28	20	15
Source	9	3	3		2	9	9	10
Number of LP's soived with DZLP <1>								
min.	66	15	14		__	22	14	___
max.	112	126	32		__	50	16	___
Number of iter.	208	72	63		__	85	67	___
	472	420	137			223	76	
Number of BB nodes	___	___	___		99	___	___	26
Number of heuristic searches	2	1	0	0	0	0	0	0
Number of successful searches **	2	1	___	___	___	___	___	___
Number of guesses	5	0	0	0	0	6	10	10
Number of vars. fixed before BB	6	13	0	0	1	0	0	0
LP optimum before	-10672	55	-521	-521	-4250	-12642	-6155	14.96
and after cuts	-10655	69	____	-550	____	-12436	-6138	16.62
Integer optimum	-10618	73	-550	-550	-4190	-12400	-6120	26
Number of cuts	46	7	0	6	0	13	10	47
Number of pronagating nodes accepted	4	0	0	0	31	0	13	1
Number of enum. nodes(w/o LP's)	305	55	9	0	0	21	3	37
Number of LP's in BB	1	1	1	1	5	1	1	1
Number of LP's for guesses	5	0	0	0	0	6	7	10
Total number of LP's	6	1	1	1	5	7	8	11

* A third run was described in section 1.

** A heuristic search which yields a feasible solution is called "successful".

We have taken the number of linear programs as measuring stick since reliable timings are not easy to obtain in the interactive time sharing environment. It is clear that the additional computational effort must be taken into account as well. But it appears that the resolution of a linear program requires an effort which is an order of magnitude larger than the rest. And this can only be more so in the environment of production linear programs, when most of the data are on auxiliary storage, whereas at least part of the reduction efforts, for example, could be carried on over logical inequalities and Benders Inequalities in high speed storage.

In terms of reduction of effort as measured by linear programs, the results of the table are certainly promising.

6. Appendix: Definition of Some Terms <1>

- A logical inequality in bivalent variables is meant to be an inequality with 0, 1, -1 coefficients only.
- A preferred variable inequality (abbreviated p.i.) is a logical inequality of the form $\Sigma_j \tilde{y}(j) \geq 1$, $\tilde{y}(j)$ being either $y(j)$ or $1 - y(j)$, which expresses the condition that at least one of the $\tilde{y}(j)$, $j \in J$, must be one.
- The values $\tilde{y}(j) = 1$ (i.e., $y(j) = 0$ or $y(j) = 1$, depending on the nature of the $\tilde{y}(j)$) are termed the suggested or indicated values of the preferred variables.
- Given some procedure for generating a set of preferred inequalities, we call minimal (relative to the procedure) those p.i.'s with minimal number of non-zero coefficients.
- The degree d of a m.p.i. system (and by extension the degree of the initial inequality or system of inequalities from which the m.p.i. system was derived) is the number of non-zero coefficients of one of the minimal preferred inequalities.

7. References

<1> Spielberg, K., " Preferred Variable Branch-and-Bound Method", revised version of "A Minimal Inequality Branch-Bound Method", IBM Phila. Scient. Center Report 320-3024, June 1973.

<2> Fayard, D., and G. Plateau, "Techniques de Résolution du Problème du Knapsack en Variables Bivalentes", Lab. de Calcul, Univ. de Lille, 1975.

<3> Lemke, C.E., and K. Spielberg, "Direct Search Zero-One and Mixed Integer Programming", J. of ORSA, Vol. 15, 1967.

<4> Gomory, R. E., and E. L. Johnson, "Some Continuous Functions Related to Corner Polyhedra", Math. Programming, Aug. 1972.

<5> Guignard, M., and K. Spielberg, "The State Enumeration Method for Mixed Zero-One Programming", IBM Phila. Scientific Center Report, 320-3000, Feb. 1971.

<6> Spielberg, K., "Minimal Preferred Variable Methods in Zero-One Programming",revised version of "Minimal Preferred Variable Reduction for Zero-One Programming" IBM Phila. Scientific Center Report, 320-3013, 1972.

<7> Hammer, P.L.," Boolean Procedures for Bivalent Programming", Res. Report,CORR 73-1, Dept. of Comb. and Optimization, Univ. of Waterloo, Oct. 1973.

<8> Balas,E., R. Jeroslow, On the structure of the unit hypercube, Management Science Research Report 198, Carnegie Mellon University, Pittsburgh, Pa.,1969.

<9> Balas, E., and Jeroslow, R., "Canonical Cuts of the Unit Hypercube", SIAM Journal of Applied Mathematics, Vol. 23 (1), July 1972.

<10> Benders, J. F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik, Vol 4, 1962.

<11> Petersen, C., "Computational Experience with Variants of the Balas Algorithm Applied to the Solution of R and D Projects", Management Science, Vol. 13, 1967.

<12> Bouvier, B., and G. Messoumian, "Programmes Linéaires en Variables Bivalentes", Thesis, Faculté des Sciences, Univ. Grenoble, 1965.

<13> Salkin, H.L., and K. Spielberg, "An Adaptive Algorithm for Binary Programming", IBM New York Scient. Center Report, 320-3-13, 1972.

<14> Hammer, P. L., M.W. Padberg, U.N. Peled, "Constraint Pairing in Integer Programming", Univ. of Waterloo, Department of Combinatorics and Optimization, Research Report CORR 73-7, August 1973.

Anschriften:
University of Pennsylvania
USA – Philadelphia, Pa. 19174

IBM Scientific Marketing
W. Barnard Drive
USA – Stamford, CT 06905

Zur Auswahl von Basislösungen bei Verfahren der ganzzahligen Optimierung

W. Habenicht, Göttingen

Summary: Many integer programming methods relax the nonnegativity constraints on the basic variables of the associated linear program. The choice of a suitable basis is a substantial part of the solution strategy for most of such methods. The concepts of dominated or θ-dominated basis, introduced in this paper, provide a 'good' choice of a basis. An example is given to show that starting with an optimal basis is not always the best strategy for such integer programming methods.

1. Einleitung

Bei nahezu allen Verfahren der ganzzahligen linearen Optimierung spielen Relaxationen, deren Lösungsmenge durch die Nichtnegativitätsbedingungen der Nichtbasisvariablen einer Basislösung des zugehörigen linearen Programms gegeben sind, - diese seien im Folgenden als *Kegelprobleme* bezeichnet - eine bedeutende Rolle.

Gruppentheoretische Konzepte (z.B. [GOMORY-1965],[GORRY-SHAPIRO-1971]) lösen in der ersten Phase das "Asymptotische Problem", häufig auch als "Integer Program over the cone" bezeichnet. Hierbei wird i.A. von dem Kegelproblem ausgegangen, das zu einer optimalen Basislösung des zugehörigen linearen Programms gehört.

Branch-and-Bound Verfahren (z.B.[DAKIN-1965]) benutzen Techniken, z.B. *Strafkosten* (vgl.[DRIEBEEK-1966],[TOMLIN-1971]), die auf der Betrachtung eines geeigneten Kegelproblems beruhen.

Suchverfahren, wie jenes von Hillier ([HILLIER-1969]) vorgeschlagene, basieren auf der Suche nach ganzzahligen Lösungen des Kegelproblems, das zu einer optimalen Lösung des zugehörigen linearen Programms gehört.

Alle *Schnittebenenverfahren*, die von einer optimalen Lösung des zugehörigen linearen Programms ausgehen (z.B.[GOMORY-1963], [GOMORY-1960]) bestimmen Schnitte unter Vernachlässigung der Nichtnegativitätsbedingungen der Basisvariablen.*Konvexe-Körper-Schnitte* ([BALAS-1971]) werden bestimmt durch die Schnittpunkte der Kanten des Kegels, der durch die Nichtnegativitätsbedingungen

§ Diese Arbeit wurde unterstützt von der Deutschen Forschungsgemeinschaft.

Preferred Variables and Improved Penalties in 0-1 Programming*

Monique Guignard

The Wharton School
University of Pennsylvania

Technical Report No. 29

August 1973, Revised December 1975
Revised December 1976

* An earlier version of this paper was presented at the 8th International Symposium on Mathematical Programming, Stanford, August 1973.

Abstract

Preferred inequalities[5,6] impose logical conditions on 0-1 structural variables of an integer LP problem. Penalties for altering the continuous optimal solution can under certain conditions be improved in view of these feasibility restrictions.

Notation

Let $Pen(x_k)$ be the penalty incurred for complementing a nonbasic variable x_k. Let $UPen(x_k)$ (resp. $DPen(x_k)$ be the penalty incurred for rounding up (resp. down) a basic variable x_k.

Let A_i^j denote the element in row i, column j, of a matrix A.

I. Introduction

Consider a 0-1 minimization problem, and the expression of its objective function in terms of variables which are nonbasic in the continuous solution.

$$z = z_0 + \sum_{j\in N_0} d^j x_j - \sum_{j\in N_1} d^j x_j, \quad d^j \geq 0 \quad \forall j \in N_0 \cup N_1. \qquad (\omega)$$

$N_0 (N_1)$ is the set of nonbasic variables at 0 (resp. 1). The current value of z is thus

$$\bar{z} = z_0 - \sum_{j\in N_1} d^j$$

Consider also a preferred inequality derived from the constraints $Ax = b$ in 0-1 variables:

$$\sum_{i\in P} \tilde{x}_i \geq 1$$

where $\tilde{x}$ can be either x_i, $i \in P_1$, or $1-x_i$, $i \in P_0 = P - P_1$. (See[5],[6],[7]). Rewritten as

$$\sum_{i\in P_0} x_i - \sum_{i\in P_1} x_i \leq |P_0| - 1 \qquad (\rho)$$

this inequality has the following meaning: a "suggested" value for

x_i, $i \in P_1$, (resp. $i \in P_0$) is 1 (resp. 0) and at least one variable in P must be equal to its suggested value in an integer feasible solution.

Comparing suggested values of nonbasic preferred variables and their current values, one may be able to improve the penalty computation for basic as well as nonbasic variables.

Essentially the idea is the following:

1. If a variable is currently nonbasic at its suggested value, complementing it will force another variable to become equal to its preferred value. If this changes the current nonbasic values, at least another penalty will be incurred.

2. If all preferred variables are nonbasic and different from their suggested values, there is an overall penalty equal to the smallest of the penalties of these preferred variables since at least one of them must change.

3. If a basic fractional variable must become integral, a simulated pivot step on any expression bounding the objective function from below will provide a corresponding penalty.

Let P_{rs} be the set of preferred variables with suggested value r and current value s, $r, s \in \{0,1\}$.

Assumption 1: No preferred variable in (ρ) is basic, i.e. if B is the optimal basis, $P \cap B = \emptyset$.

Then (ρ) becomes

$$0 \geq 1 - |P_0| + \sum_{P_0} x_j - \sum_{P_1} x_j$$

or translated into the same coordinate system as the nonbasis

$$0 \geq 1 - |P_0| + \sum_{P_{00}} x_j - \sum_{P_{01}} \bar{x}_j - \sum_{P_{10}} x_j + \sum_{P_{11}} \bar{x}_j + |P_{01}| - |P_{11}|$$

i.e.

$$0 \geq 1 - |P_{00}| - |P_{11}| + \sum_{P_{00}} x_j + \sum_{P_{11}} \bar{x}_j - \sum_{P_{10}} x_j - \sum_{P_{01}} \bar{x}_j \qquad (\sigma)$$

where $\bar{x}_j = 1 - x_j$

II Improved Penalties

Theorem 1.

If $P_{00} \cup P_{11} = \{k\}$, and if $P \cap B = \emptyset$

$$\text{Pen}(x_k) = d^k + \text{Min}\{d^\ell \mid \ell \in P_{10} \cup P_{01}\}.$$

Proof:

$P_{00} \cup P_{11} = \{k\} \Rightarrow |P_{00}| + |P_{11}| = 1$.

Let $\ell \in P_{10} \cup P_{01}$ be the index of the minimum d^ℓ.

Add d^ℓ times (ρ') to (ω):

$$z \geq \bar{z} + \sum_{N_0 - P_{10}} d^j x_j + \sum_{N_1 - P_{01}} d^j \bar{x}_j + \sum_{P_{10}} (d^j - d^\ell) x_j + \sum_{P_{01}} (d^j - d^\ell) \bar{x}_j + (d^k + d^\ell)\, \hat{x}_k$$

where $\hat{x}_k$ is x_k if $P_{00} = \{k\}$ or $\bar{x}_k$ if $P_{11} = \{k\}$,

and $\forall j \in P_{10} \cup P_{01}$, $d^j - d^\ell \geq 0$.

Theorem 2.

If $P_{00} \cup P_{11} = \emptyset$, and if $P \cap B = \emptyset$,

$\forall k \in N - P$, $\text{Pen}(x_k) = d^k + \text{Min}\{d^\ell \mid \ell \in P_{10} \cup P_{01}\}$.

Proof:

$P_{00} \cup P_{11} = \emptyset \Rightarrow |P_{00}| + |P_{11}| = 0$.

Add d^ℓ times (σ) to (ω):

$$z \geq \bar{z} + d^{\ell} + \sum_{N_0 - P_{10}} d^j x_j + \sum_{N_1 - P_{01}} d^j \bar{x}_j + \sum_{P_{10}} (d^j - d^{\ell}) x_j + \sum_{P_{01}} (d^j - d^{\ell}) \bar{x}_j$$

and $\forall j \in P_{10} \cup P_{01}$, $d^j - d^{\ell} \geq 0$.

Theorem 3.

$\forall i \in B$,

$$UPen(x_i) = \min_{j \in N} \left\{ \frac{\delta^j (1 - t_i)}{-T_i^j} \;\middle|\; -T_i^j > 0 \right\}$$

and $$DPen(x_i) = \min_{j \in N} \left\{ \frac{\delta^j t_i}{+T_i^j} \;\middle|\; +T_i^j > 0 \right\}$$

where $t = B^{-1}b$ and $T = B^{-1}A$ are the updated right-hand side and tableau respectively, and where δ^j is $Pen(x_j)$ obtained from Theorem 1.

Proof: Consider an expression of the type

$$z \geq \bar{z} + \sum_{N} \delta^j x'_j$$

obtained from applying Theorem 1, where x'_j is either x_j or $1 - x_j$, and $\delta^j \geq 0 \forall j \in N$.

Consider also the row of the updated tableau corresponding to x_i, $i \in B$.

$$x_i = t_i - \sum_{j \in N} T_i^j x'_j$$

Consider for instance rounding x_i up to 1:

$$t_i - \sum_{j \in N} T_j^i x'_j = 1.$$

At least one nonbasic variable must become positive.

For $j_0 \in N$, $T_i^{j_0} x'_{j_0} = t_i - 1 - \sum_{N - j_0} T_i^j x'_j$.

if $T_i^{j_0} \neq 0$, $$x'_{j_0} = \frac{t_i - 1}{T_i^{j_0}} - \sum_{N - j_0} \frac{T_i^j x'_j}{T_i^{j_0}}$$

If x'_{j_0} alone changes, and becomes positive, $T_i^{j_0} < 0$ and

$$z \geq \bar{z} + \delta^{j_0} \frac{t_i - 1}{T_i^{j_0}} = \bar{z} + (t_i - 1) \frac{\delta^{j_0}}{T_i^{j_0}}$$

The least increase in z is $\min\left\{ \frac{(-1 + t_i)\delta^{j_0}}{+T_i^j} \;\middle|\; T_i^{j_0} < 0 \right\}$.

which is the up penalty for x_i. One would obtain the down penalty in a similar way.

III Additivity

1. Suppose there is a system of preferred inequalities satisfying the assumptions of Theorem 1 for the same variable k.

If two preferred sets P_1 and P_2 have only k in common, then

$$Pen(x_k) = d^k + \sum_{h=1,2} \min\{ d^{\ell} \mid \ell \in (P_{01} \cup P_{10})_h \}$$

If two preferred sets P_1 and P_2 have an intersection larger than {k}, then

$$Pen(x_k) = \max_{h=1,2} [d^k + \min\{ d^{\ell} \mid \ell \in (P_{01} \cup P_{10})_h \}].$$

2. Suppose there is a system of preferred inequalities satisfying the assumptions of theorem 2.

If two preferred sets $(P)_1$ and $(P)_2$ are disjoint, then

$$\forall k \in N - (P)_1 - (P)_2,\ Pen(x_k) = d^k + \sum_{h=1,2} Min\{d^\ell \mid \ell \in (P_{01} \cup P_{10})_h\}$$

$$\forall k \in (P)_h,\ Pen(x_k) = d^k + Min\{d^\ell \mid \ell \in (P_{01} \cup P_{10})_{h'}\}$$

where $h = 1$ or 2 and $h' = 2$ or 1 respectively.

If two preferred sets $(P)_1$ and $(P)_2$ are not disjoint, then many different situations can arise, and the simplest (if not always best) way to compute penalties is

$$\forall k \in N - (P)_1 \cup (P)_2,\ Pen(x_k) = d^k + \max_{h=1,2} Min\{d^\ell \mid \ell \in (P_{01} \cup P_{10})_h\}$$

$$\forall k \in (P)_h - (P)_{h'},\ Pen(x_k) = d^k + Min\{d^\ell \mid \ell \in (P_{01} \cup P_{10})_{h'}\}$$

$$\forall k \in (P)_1 \cap (P)_2,\ Pen(x_k) = d^k$$

where $h = 1$ or 2 and h' is 2 or 1 respectively.

IV Example

Consider the 6×12 problem of [1]. Variable x_1 can immediately be fixed at zero. The resulting system of constraints yields inequalities of size at least 2 (i.e. containing at least two variables). The relaxed LP has an optimal value of

6.85 and the determinant of the optimal basis is 34,868. Five variables are fractional (out of 12). One can generate five inequalities of size two, and two of them will yield improved penalties:

$$x_5 - x_2 \geq 0$$

$$x_{11} - x_9 \geq 0.$$

The original penalties are

i	1	2	3	4	5	6	7	8	9	10	11	12
down penalty	-	0	2.1	7.1	0	1.1	0	1.1	0	1.2	0	0
up penalty	-	1.8	0.2	1.1	9.7	3.5	6.7	1.4	7.1	1.3	4.6	1.2

In the optimal (fractional) solution, the preferred variables are all zero. Setting x_2 or x_9 at 1 would force x_5 and x_{11} at 1 too. Therefore the penalties can be improved for x_5 and x_{11}:

$$Pen(x_5) = 1.8 + 9.7 = 11.5$$

$$Pen(x_{11}) = 7.1 + 4.6 = 11.7$$

Also x_6 is basic and is equal to 0.54. Using now the improved nonbasic penalties, one can obtain the stronger up penalty $UPen(x_6) = 3.7$ (vs. 3.5)

V Conclusion

The techniques described in this paper have been incorporated in an experimental interactive system for 0-1 programming [4]. Minimal preferred inequalities are one of the basic tools of this system and are generated for a number of reasons. Strengthening of the penalties can be viewed as a further bonus obtained at very little extra computational cost. Preprocessing of the constraints is done at selected nodes. Penalties are used for branching (store nodes with large penalties) or for fixing of variables once a good upper bound on the objective function has been found. Penalty improvement can be substantial, even for larger problems which frequently yield "strong" minimal preferred inequalities. For instance a 28×35 tanker scheduling problem [5] with a continuous optimum of 521.45 sees 6 of the initial 19 nonbasic penalties ranging between 0 and 50 increased on the average by 175. The integer optimum is 550. Similar behaviors have been observed with larger problems, such as the 37×74 problem of [5], especially if one reoptimizes after having added Gomory-Johnson cuts to the optimal tableau[2,3]. In general it seems that these penalty improvements often allow early fixing of variables and therefore reduce the size of the search tree.

VI References

[1] B. Bouvier, G. Messoumian,
Programmes linéaires en Variables bivalentes,
thesis, Faculté des Sciences, Univ. of Grenoble, France, 1965.

[2] R. E. Gomory, E. L. Johnson,
Some Continuous Functions Related to Corner Polyhedra.
Math. Programming 3(1972)23-86, 3(1972)359-389.

[3] M. Guignard, 1971
Inegalités Valides de Gomory-Johnson,
Journées de Combinatoire de l'AFCET, December 1971.

[4] M. Guignard, K. Spielberg,
An experimental interactive system for integer programming,
Bicentennial Conference on Mathematical Programming,
N. B. S. , Gaithesburg, Maryland, 1976.

[5] C. Lemke, K. Spielberg,
Direct search algorithms for 0-1 and mixed integer programming, Opns. Res. 15 (1967) 892-914.

[6] K. Spielberg,
Minimal preferred Variable Reduction for 0-1 Programming,
IBM Philadelphia Scientific Center Report 320-3013, (1972).

[7] K. Spielberg,
A Minimal Inequality Branch-Bound Method,
IBM Philadephia Scientific Center Report 320-3024 ,.(1973)

[8] J. A. Tomlin,
Branch-and-Bound Methods for Integer and Non Convex Programming, in: J. Abadie, (ed.),
in: Integer and Nonlinear Programming,
(North Holland, Amsterdam, 1970)437-450.

Survey of Enumerative Methods for Integer Programming*

M. Guignard
University of Pennsylvania

K. Spielberg
Scientific Computing IBM

U. Suhl
IBM Research**

* Invited talk at SHARE 51 Meeting, Boston, August 1978.

** On leave from: Freie Universitaet Berlin.

Technical Report No. 41

August 1978

Table of Contents.

0. Introduction

For practical purposes, integer and mixed integer programming problems (or shorter: "programs") are usually solved by Branch and Bound ("BB") techniques, Enumerative techniques or Heuristics. There exist many other methods which are highly interesting, such as cutting plane methods, group methods, relaxation methods, etc. These are usually more difficult to implement than Branch and Bound Programming ("BBP") or Enumeration, and also less robust (more likely to break down), and we therefore take the point of view that such methods ought to be used within the context of BBP or Enumeration. Such an approach permits the user to fall back upon the simpler technique whenever the more sophisticated technique shows symptoms of being ineffective or overextended. Similarly, we feel that heuristics are often best imbedded in BB or Enumerative Programming (as auxiliary tools), and dually to this that incomplete BBP or Enumeration runs can be taken as excellent starting points for heuristic approaches.

Mathematically, we consider either the pure integer programming problem in m constraints and n variables $y(j)$, $j=1,2,\ldots,n$:

$$\text{IP:} \quad \min \; c\,y = z \qquad (0.1)$$
$$C\,y \leq b$$
$$y(j) \geq 0 \;\text{ integral},\; j=1,2,\ldots,n$$

or the mixed integer problem with an added set of continuous variables $x(k)$, $k=1,2,\ldots,p$:

$$\text{MIP:} \quad \min \; c\,y + d\,x = z \qquad (0.2)$$
$$C\,y + D\,x \leq b$$
$$0 \leq x(k)\;;\quad y(j) \geq 0\;,\quad \text{integral}$$

1. Branch and Bound Programming and Enumeration

In this paper we shall concentrate on the two main streams of integer programming:
Branch and Bound Programming (BBP), (see <LAD>), andand
Enumerative Programming (EP), (see <BA>,<LS>,<SP.1>),
but use other techniques freely where appropriate in the broader context.

1.1 General Problems

There is also some disagreement, and often some confusion, as to what constitutes the difference between BB Programming and Enumeration. We shall have more to say about this, but want to stress at the outset that for us the main distinction will be that

In BBP the integer program is resolved by the solution of a sequence of relaxed problems (i.e., problems in which a substantial portion of the integer variables are treated as continuous variables), whereas

In Enumeration the integer problem is resolved by the solution of a sequence of highly constrained problems (i.e., problems in which all or a substantial portion of the integer variables are set to a trial value (we shall say that this choice determines a "STATE", and that the corresponding problem is the "state problem").

What confuses the issue is that the "tree structure" for the two approaches tends to look similar. A BB tree as defined originally by Land and Doig <LAD> has a structure as in Fig. 1.

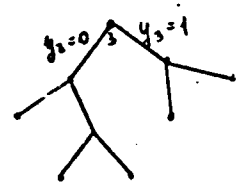


Fig. 1

The top node corresponds to the fully relaxed problem, with all integer variables (taken to be 0-1 variables in the sequel) relaxed to be continuous (i.e. to be in the interval (0,1)). Its two successors denote the problems obtained from the first problem (or node) by the fixing of some fractional basic variable, say variable 3 in the example, at 0 or 1 and the solution of the corresponding problems in which the remaining variables are still relaxed; etc., etc..

In a production system one must often deal with large masses of data, so that the execution of each tree problem (be it a relaxed problem or a state problem) requires the reading of data from auxiliary storage files. To minimize the search for "pending nodes", one therefore tends to pursue the successors of the currently resident pending node (FIFO strategy) until this is no longer necessary. Therefore the tree tends to take the shape of Fig. 2, which is also the shape of Enumeration trees.

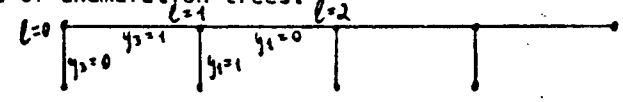


Fig. 2

It must be stressed again, however, that in BBP all nodes correspond to relaxed problems , whereas for Enumeration the nodes are associated with state problems. The actual situation may be more complex than this: in Enumeration the pending nodes (indicated by the vertical branches) usually do not indicate resolved problems but problems which are to be solved after return of the search to the level in question. (This is similar to the more rarely used pseudo nodes of BBP). Also, the state problem needs to be resolved only when

preliminary logical calculations suggest that the state is feasible and may lead to a new solution.

In its simplest form the Enumeration starts at level 0 with the determination of an initial state (e.g., the initial state is often taken as the zero vector $y = (0,0,\ldots,0)$, or as the vector which results from the rounding of the solution to a strictly auxiliary relaxed problem. A branch variable is then chosen and "fixed in a forward step", from level 0 to level 1, i.e. usually fixed at the value opposite to the one it took in the state. At level 1 the procedure is repeated. In general, however, the auxiliary relaxed problem is solved only rarely and often this is also the case for the state problem, whereas the resolution of at least one relaxed problem is mandatory for BBP (unless one purposely wishes to introduce enumerative methods into BBP, as we would certainly advocate).

Of course, there are various logical tests which must be performed at each node, and which may lead to the recognition that no more improvement can be expected on a forward step, so that a "backward step" can be taken. E.g., if the execution of an auxiliary relaxed problem yields an objective function which is lower than z^*, the "incumbent" upper bound on z, then a backup is warranted. The attainment of an integer solution may (but need not always) permit a backup. The inspection of pure integer constraints may also permit the fixing of some of the free variables, or the shrinking of bounds on the continuos and on the slack variables. Instead of describing such tests, which are well known from the literature, we shall suppose the existence of a set of general logical "reduction" procedures which serve the same function, but which also permit the construction of logical inequalities on the zero-one variables. Our experiments indicate that these can play a central role in the guidance and the curtailment of enumerative procedures for a substantial class of problems.

1.2 Structured Problems.

A fundamental advantage of EP over BBP is the possibility of handling certain simple constraints implicitly, usually on the (0,1) variables. By that we mean that one refrains from adding them to the matrix. I.e., one either does not use them in the solution of the auxiliar linear programming problem at all, or one solves that problem by special methods. We give some examples to illustrate this point.

Transportation Problems (possibly with some fixed charges)

These problems are somewhat outside the main subject of our paper. But the reader may readily recall that one usually treats the linear programs by special purpose methods: stepping stone method, Hungarian method, out-of-kilter methods. Similar words could be said about the more general network problems.

ultiple Choice Problems

Many problems have constraints of the form

$$\sum y(j) = 1 \quad , \quad j \in S(i), \quad i=1,2,\ldots,m \quad . \qquad (1.1)$$

.e., various subsets of the (0,1) variables $y(j)$ must sum to exactly ne.

It is true that BB production codes usually have special features "special ordered sets") for dealing with such structures. But these re specially added features. In EP, on the other hand, multiple hoice sets are handled easily in the normal processing. For example, state would be defined only such as to satisfy the multiple choice equirements.

ogical constraints

Many problems, often of the scheduling type, have to conform to mple implications such as $y(j)=0 \dashrightarrow y(k)=1$ or $y(j)=1 \dashrightarrow y(k)=1$, or some j and k, (priority, precedence, conflict constraints). More eneral logical implications may be of the form:

$$\sum \tilde{y}(j) \geq 1 \quad , \qquad (1.2)$$

here $\tilde{y}(j)$ is either $y(j)$ or its complement $\bar{y}(j)=1-y(j)$.

Such constraints can be stored in binary matrices and can be used o substantial advantage, especially if the number of non-zero elemts re the same for each such constraint (one then speaks of this number s the "degree" of the system of logical constraints. In section 3 we hall describe ways of generating logical inequalities for any (0,1) r mixed (0,1) system.

. Simple State Enumeration under MPSX/370 (with cutting planes)

We give some results for pure zero-one problems (private ommunication <JS>) of experiments conducted by means of the extended ontrol language of MPSX/370 (<MP.1,2,3>).

The implemented experimental system (not available outside IBM) s a HYBRID code, in the sense that it consists of two quite ndependent phases. Phase 1 is the execution of a very fast tand-alone enumerative code which starts the search at the origin =(0,0,...,0), and is in essence that of <GA>. This code relies eavily on highly efficient storage and processing of the coefficient atrix C (assumed to be sparse), especially for the purpose of what e shall call logical tests of degree 0 (establishment of nfeasisibility) and degree 1 ("cancellation" or "fixing" of ariables at values implied by the system of inequalities (see <SU>. t the end of phase 1, limited by user-controlled parameters, one sually has found a good feasible solution.

In phase 2 one commences with the execution of the auxiliary inear programming problem (via MPSX/370) augmented by the addition f multiple Gomory-Johnson cutting planes and reoptimization, <GJ>. The udicious use of cutting planes has the effect of improving the lower ound on the objective function, and also appears to render the ractional solution vector y_f closer to the integer solution vector

. The initial vector y_f is rounded to give a starting state for the enumerative phase.

One then branches away from the starting state in accordance with a feasibility improvement criterion. At each node one executes degree 0 and degree 1 tests, checks for integrality, and objective function value, and at selected nodes one resolves an auxiliary relaxed problem. Again the emphasis is on efficient storage and node processing.

Some experimental results are summarized in tables 1 and 2. The comparisons are with standard runs of MPSX-MIP/370. As such they are somewhat unfair to the production code, in as much as no attempts were made to improve its performance via finer tuning of MIP. Moreover, the HYBRID system itself is coded in the Extended Control Language of MPSX/370 and calls the optimization routines of MPSX/370 for resolution of its relaxed auxiliary problems, as well as for addition of cuts and reoptimizations. Also, the times associated with generating and adding the Gomory-Johnson cuts were discarded. They are much higher in the Extended Control Language than would be required if the cuts were added within the system.

Nevertheless, the experiments indicate that enumeration can be highly effective for certain problems. For problems 1, 2 and 5, the pure enumeration code was almost sufficient on its own, phase 2 aiding principally in the clean-up phase (the proof of optimality: a notoriously difficult task, sometimes considered important, sometimes not necessary in practice).

Table #1: Problem Data

Problem number	Problem origin	no. of rows	no. of cols	density	zc	zI
1	BM 23	20	27	.885	20.57	34
2	BM 24	20	28	.884	24.57	38
3	BM 25	20	30	.885	21.03	43
4	LS E	28	89	.124	834.68	1120
5	IB 1	157	78	.064	-1990197.89	-1786000
6	IB 2	225	1994	.04	3478.31	3534

All problems are pure 0-1 problems. The BM problems are small, but quite difficult problems from <BM>. LS E is the most difficult problem from reference <LS>. The two IB problems are practical problems made avallabe to us by IBM Paris. They are not available outside IBM.

zc and zI stand for optimal continuous objective function value (with cuts added) and optimal integer objective function value, respectively.

Table #2: Comparison Results within System MPSX/370

First entry in column: standard run with MPSX-MIP
Second entry: HYBRID, an enumerative code using MPSX/370 for resolution of the relaxed problems

Optimal Integer Sol. Found						Optimality Ascertained					
#nodes		#LP Iters		CP-time (secs)		#nodes		#LP Iters		CP-time (secs)	
427	10	2137	0	27	0	463	1087	2356	155	28	2
93	9	478	0	7	0	295	1425	1573	274	22	4
468	43	2454	83	29	0	4209	3510	4209	458	54	13
9174	823	35710	282	450	4	>15000	23029	46625	3203	>720	564
241	14	1293	0	26	0	348	735	348	0	42	2
594	2654	13979	2110	1348	201	774	15381	17037	10471	1780	1271

3. State Enumeration with Logical Inequalities

We now consider the important task of *guiding an enumeration* algorithm logically, in a fashion which is determined by the data themselves. This can, in some measure, be accomplished by the generation and systematic exploitation of logical inequalities of the form (1.2) during the enumeration (possibly also within BB algorithms, but perhaps not as naturally in that context).

3.1 The Associated (Auxiliary) Inequalities

The derivation of logical inequalities always proceeds from an initially available set of integer inequalities. This may be the original system of constraints (0.1). But more interesting is the case when the set of integer inequalities is derived from linear programming tableaux during the enumerative process. We shall call such inequalities the "associated inequalities", and write them in the form:

$$B \cdot y \leq r + \delta \cdot z^* \quad . \qquad (3.1)$$

The notation indicates that the associated (also "auxiliary") inequalities may or may not depend on the current upper bound on the

objective function, i.e. on z^*. It is well known (see <BE>), that one can obtain these inequalities for both (0.1) and (0.2), i.e. also for the mixed integer case, from any dually feasible linear programming tableau resulting after full or partial substitution of a vector y. One usually distinguishes between two types of "Benders" inequalities, type 1 if the linear program is feasible, type 2 if it is not.

Type 1 $\quad (c+u.C).y - u.b \leq z^* \qquad (3.2)$

Type 2 $\quad - v'.(b - C.y) \leq 0 \quad . \qquad (3.3)$

The u and v are variables of the problems dual to (0.1) or (0.2). But one knows equally well how to read such inequalities directly from the top row (in case of feasibility) or from an infeasible row of the appropriate LP tableau. For what follows it is basic, and this is essentially a matter of experimentation and experience, that the associated inequalities often lend themselves readily to the derivation of tight logical conditions on the integer variables.

3.2 (Minimal) Logical Inequalities

Given a set of associated inequalities (and the original constraints in case of IP), one can usually derive logical constraints of the form (1.2), see <LS>, <BJ>, <GH>, <SP. 1>. We illustrate the procedure by an example. Take as a given inequality :

$$-2y1 + 2y2 - 5y3 + y4 + y5 \leq -1 \quad .$$

One first complements the positive coefficients to get a maximally "strong", i.e. negative, right hand side.

$$-\bar{y}4 - \bar{y}5 - 2y1 - 2\bar{y}2 - 5y3 \leq -5 \quad .$$

The coefficients have been ordered in decreasing sequence. Now one proceeds to add multiples of appropriate hyper cube inequalities ($y(j) \leq 1$ or $y(j) \geq 0$) to eliminate coefficient from the left. E.g., the first term is removed by addition of $\bar{y}4 \leq 1$, the third term by addition of $2\bar{y}2 \leq 2$. This process may be continued as long as the right hand side remains negative. In such a fashion one obtains $-2\bar{y}2-5y3 \leq -1$, so that the (0,1) nature of the variables finally implies $\bar{y}2 + y3 \geq 1$, i.e.: (either y2 is 0) or (y3 is 1) .
If the logical implication involves two variables, one says the degree of the inequality is two .

Logical implications of degree two are probably the most important ones. Of course there are stronger (lower degree) ones, for example:
$y1 - 3y2 \leq -4$ is infeasible, d = 0,
$y1 - 3y2 \leq -3$ leads to degree 1 (both $\bar{y}1 \geq 1$ and $y2 \geq 1$). However, logical tests for degree 0 and degree 1 implications are known older

features of enumeration algorithms, and only degree two implications promise further improvements.

One easily extends the above to systems of inequalities. For these one is interested in the smallest degree (over the rows of the system) and designates it as the degree of the system. Finally, one may generate all (or a specified number) of the minimal logical implications (termed "minimal preferred inequalities" in <SP.1>) and carry them along for use in the form

$$Q \,.\, y \leq q \qquad . \qquad (3.4)$$

Q has entries of 0, +1, and -1 only, with d non-zero elements per row, and q is implied by Q and (1.2). As an example we give a 6 by 12 system.

-1	3	-12	0	-1	7	-1	0	0	3	-5	-1
3	-7	0	1	6	0	0	0	0	0	0	0
11	0	1	-7	0	-1	2	1	-5	0	9	0
0	5	6	0	12	.	0	3	1	-8	0	5
-7	-1	-5	3	1	4	0	-2	7	1	0	-7
-2	0	0	-4	0	0	-3	-5	-1	0	1	1

Its degree is 1, with the implication y1 = 0. Once y1 has been fixed at one, the degree of the system becomes two, and the matrix Q of all degree two implications becomes (over the variables y2 through y12):

0	-1	0	0	1	0	0	0	0	0	0
-1	0	0	1	0	0	0	0	0	0	0
0	0	-1	0	0	0	0	0	0	1	0
0	0	0	1	0	0	0	0	-1	0	0
0	-1	0	0	0	0	0	0	0	-1	0
0	0	0	0	0	0	0	-1	0	1	0
0	0	0	1	1	0	0	0	0	0	0
1	-1	0	0	0	0	0	0	0	0	0
0	0	0	0	0	1	0	0	0	1	0
0	1	0	1	0	0	0	0	0	0	0
0	-1	0	0	0	0	0	0	1	0	0
0	1	0	0	0	0	0	0	0	1	0
1	0	0	1	0	0	0	0	0	0	0
0	0	0	0	-1	0	0	0	0	1	0
0	0	0	1	0	0	0	0	0	0	1
0	0	0	0	0	0	1	0	0	1	0
0	0	-1	0	0	0	0	-1	0	0	0

3.3 Probing (Looking Ahead) with logical inequalities

There are a number of ways in which a system (3.4) can be utilized to direct the enumerative search. From the columns of the matrix Q one can predict what branches will lead to a contraction of the search in the sense of leading to new nodes at which the degree is smaller. The objective is to steer the search into areas of the integer lattice in which a maximal number of variables can be fixed through logical tests. One may also combine the information within Q with other information, for example with penalties. We have proposed and tested several approaches with good results, see e.g. <GS.1,2,GU, SP.2>.

Here we would like to discuss briefly a new approach which works very well for problems with degree d=2. This approach is likely to have some elements in common with the "cascading" procedure in <SB>. Our experience has shown, that there is a wide class of problems for which the degree is small. Also, we have found that we can often induce degree 2, over major portions of the enumerative tree, by generating Benders inequalities, either during the search or even before the search by trying several random or slightly more sophisticated guesses at the solution, and by imposing reasonable upper bounds $z*$ on the objective function.

The basic idea is quite simple. Looking at a column of the sample matrix Q in section 3.2 entries of 1 or -1 one perceives that fixing the correponding variable at some value will generally cause other variables to be fixed as well. For example, row 3 of Q signifies that either variable y11 (corresponding to column 10) must be 0 or variable y4 must be 1. Hence, setting y11 to 1 must imply y4=1. Similarly, setting y11 to 0 requires y3 to be 1 via row 5 and column 7 of Q. Thus, the consequence of setting column y11 at some value can be traced by means of Q: e.g.y3=0, induced by y11=0, leads to additional fixings via other constraints represented by Q.

In one experimental code we follow roughly the following steps at each node of the enumeration:

1. Logical reduction of degree 0 and 1.
   . backup if degree 0
   . fixing (cancellation) of variables if degree 1.

2. Local search for a possible integer solution
   • [illegible] of the current state
   • [illegible] of the vectors arising from the current state by successive complementation of the free components.

3. [illegible] d≥2, determination of the next branch variable [illegible] from [illegible] consideration [illegible] contr[illegible] (ass[illegible] that d is [illegible] at the next node).

4. For d=2 (we maintain a battery of auxiliary constraints to aid in getting d to be small):

   Invoke the procedure PROBE so that:
   either variables get fixed (in that case return to 1)
   or the best branch variable is determined, in the sense that a maximal number of other variables will be fixed after the branch.

3.4 The procedure (APL function) f PROBE p

The function has the two arguments f and p. It also presupposes, of course, the presence of the constraint matrix (and also the auxiliary matrix), the logical matrix Q, a mask MAS which identifies the currently free variables, and the bounds on all structural and slack variables (in SBW). p is the number of variables to be considered for the probing; f is either 0 or 1, and specifies the extent to which the probing is to be conducted.

PROBE considers, in turn, the p variables for which the number of non-zero entries in Q is maximal. Let index j identify such a variable. Then PROBE explores the consequences of setting y(j)=0 (and y(j)=1), in terms of all the inequalities of $Q.y \leq q$. For f=0, PROBE gives the index j* (one of the p indices examined), and the setting of y(j*), which maximizes the number of implied cancellation.

When f=1, PROBE continues the examination further. It substitutes the implied cancellations, recomputes the logical inequalities and traces arising implications of degree 0 and 1. This may have two consequences:

(I) It may enlarge the list of cancelled variables so as to make the above output more meaningful, or
(II) It may lead to an infeasibility, which in turn permits the cancellation of y(j); i.e. if y(j)=0 leads to infeasibility one may surely fix y(j) at 1 and proceed to step 1 of the scheme in section 3.3.

In the next section we give a number of examples which should be quite adequate in illustrating these ideas.

3.5 Examples of Probing

We illustrate the concepts of section 3.4 by demonstrating their use in the solution of three test problems (the last two are taken from <LS>).

3.5.1 Solution of the example of section 3.2

We annotate the APL printout selectively: (the logical matrix Q is that of section 3.2 in the first call to PROBE; see *)

```
STATE IS 1 1 1 1 1 1 1 1 1 1 1 1        start search with all vars at 0
SEARCH LEVEL IS 1                       local search one level deep
PCO,PCF 0.9 0.01
ZTB,SNT,NF 0 ¯21 12                     Σinfeasibil. -21, 12 free vars
ZTB,SNT,NF 0 ¯21 11                     11 free vars by degree 1 reduction
MIN.  PI 2                              minimal degree is  2
SETTING VAR. 11 TO 0 RESULTS IN         PROBE over four variables (*)
5 11                                    y(5) and y(11) at 0
3                                                y(3) at 1
SETTING VAR. 11 TO 1 RESULTS IN
2 3 6 7 8 10
4 6 9 11
VAR. 11 FIXED TO 0                      y(6) can not be both 0 and 1
ZTB,SNT,NF 0 ¯21 10
ZTB,SNT,NF 3 ¯11 8                      8 free vars after degree 1
MIN.  PI 2                              reduction
SETTING VAR. 6 TO 0 RESULTS IN
6

SETTING VAR. 6 TO 1 RESULTS IN
2 10 12
6 10
VAR. 6 FIXED TO 0                       y(10) can not be both 0 and 1
ZTB,SNT,NF 3 ¯11 7
ZTB,SNT,NF 3 ¯11 7
MIN.  PI 2
SETTING VAR. 4 TO 0 RESULTS IN
4 7 8 9
9 12
VAR. 4 FIXED TO 1                       y(9)          "
ZTB,SNT,NF 5 ¯5 6
ZTB,SNT,NF 11 ¯1 3                      3variables after reduction
MIN.  PI 2
SETTING VAR. 8 TO 0 RESULTS IN
8
7
SETTING VAR. 8 TO 1 RESULTS IN
2 7
8
SETTING VAR. 7 TO 0 RESULTS IN
2 7
8
```



```
SETTING VAR. 7 TO 1 RESULTS IN
8
7
SETTING VAR. 2 TO 1 RESULTS IN
8
2 7
MAX COUNT JPX IS   2.5                  max. no. of cancellations is 2
V2 IS ¯8                                on branch  y(8)=1
ZB,ZTM,SMB,SM 45.112¯1¯1                single level search not feasible
8                                       for  y(2)=1
ZB,ZTM,SMB,SM 1218¯10                   feasible for  y(7)=1, with obj.
                                        function 18
FEASIBLE SOLUTION 18
OLUTION FOUND AT NODE  1
************************************
EVEL 0 PHASE 1
BJECTIVE FUNCTION 18
EW ZSR =18
OLUTION 0 0 1 1 0 0 1 0 0 1 0 1
************************************
NT. SOL. FEASIBLE  , Z= 18
B,ZTM,SMB,SM 121300                     feasible for  y(8)=1, with obj.
                                        function 13
FEASIBLE SOLUTION 13
OLUTION FOUND AT NODE  1
************************************
EVEL 0 PHASE 1
BJECTIVE FUNCTION 13
EW ZSR =13
OLUTION 0 0 1 1 0 0 0 1 0 1 0 1
************************************
NT. SOL. FEASIBLE  , Z= 13
PC IN SEARCH 1
S IS 8                                  branch on  y(8)=1
B,SNT,NF 13 0 2
B,SNT,NF 13 0 0                         2 more variables fixed as predicted
8                                       by PROBE
B,SNT,NF 11 ¯1 2                        backup, not feasible
D OF ENUMERATION
****************                        resulting problem not feasible
PTIMAL SOLUTION 0 0 1 1 0 0 0 1 0 1 0 1
PT. OBJ. FUNCTION 13
U TIME 8.1 SEC
MBER OF NODES 3
MBER OF LP'S   0
 AUX. INEQUS ARE SATISFIED BY ZSR
D
```

3.5.2 Solution of a 28 row, 35 variable test problem

For a relatively easy 28 row, 35 variable test problem, the enumeration can be summarized as follows:

Level 0, Node 1

- Solve the initial relaxed linear program
  z = 521.053
  two variables are fractional
- The degree is 2
  PROBE gives : maximal count of 3, in case y(30) is fixed at 0
- The state is determined by rounding the values of the fractional solution up if $y \geq .5$, down if $y < .5$
- The local search produces an integer solution by a change of y(1) to 1 in the state
  z = 550
  components of y at 1 are :
  1,2,7,8,10,11,14,15,17,18,19,20,23,24,25,28,31,32,35
  (there exist several other integer solutions at z=550)
- Branch on y(1)=1 from feasibility considerations

Level 1, Node 2

- 25 variables are fixed from d=1 tests
  9 free variables are left
- Maximum count is 8, with y(11)=0
  (i.e., all variables can be fixed after the branch y(11)=0)
- Branch on y(11)=0

Level 2, Node 3

- All variables are fixed
  Problem is not feasible (at an improvement)
  Return to level 1

Level 1, Node 4

- 8 variables free after backup
  7 variables free (1 variable is fixed by d=1)
- variable 2 is fixed by PROBE
  6 variables free

- 5 variables can be fixed upon branch y(4)=0
- Branch on y(4)=0

Level 2, Node 5

- all variables are fixed, no improved solution, backup

Level 1, Node 6

- 5 variables free
- all free variables are fixed by d=1
  problem not feasible, backup

Level 0, Node 7

- 34 variables free (y(2) fixed to 0 on backup)
  14 variables free after 20 cancellation (d=1)
- PROBE fixes
  y(11)=y(4)=y(15)=1, y(16)=0
  all other variables are fixed from d=1
- the resulting vector is feasible, but with z=575

Therefore the enumeration is terminated

after 1 Linear Program
and 7 Nodes

with optimal solution $z^* = 550$.

3.5.3 Solution of a larger test problem

The difficult 37 row, 74 column problem of <LS> also lends itself to logical probing, but requires more effort. One APL test run required resolution of 125 linear programs and development of 175 nodes.

An upper bound of -521 was imposed on the objective function to increase the effectiveness of the logical tests. The initial LP objective function value is -573.11, the optimal integer o.f. value is -540. An integer solution of z = -538 was found at node 2, but the optimal solution was attained only at node 153. What is most important, is that the level of the search stayed as low as 2 to 3 through most of the search. The function PROBE resulted in typical forecasts of from 5 to 10 cancellations on some projected branch.

The overall results are comparable with the best results obtained in <SAS> and <PI>. We believe that a current version of our program could do somewhat better, but more importantly we believe that incorporation of "propagation" (see <SP.3>, <GS>) would permit the saving of a sizeable fraction (perhaps 1/3 to 1/2) of the required LP programs.

4. Comparison with Branch and Bound Methods

In table #3 we reproduce a summary of distinctions between BBP and EP. Some of these points have already been discussed in this paper. For a fuller point-by-point discussion we refer to <SP.2>.

TABLE #3: Branch-Bound versus Enumeration

1. integer problem replaced by sequence of relaxed problems	1. integer problem replaced by sequence of constrained problems
2. relaxed problem resolved at each node	2. full state problem often solved only at subset of nodes
3. examined lattice points "close" to each other	3. examined lattice points can be far apart
4. surrogate and Benders inequalities tend to be weak	4. generated inequalities tend to be strong (esp. in a "logical sense")
5. structural logical constraints are difficult to accommodate	5. structural constraints can be handled to advantage
6. mixed-integer problems handled easily and automatically	6. mixed-integer problems require generation and use of Benders inequalities
7. special-purpose codes difficult to code (when robust LP-code needed)	7. special-purpose codes easily constructed
8. relaxed problem often degenerate	8. state problem often not degenerate

5. References

<BA> E. Balas, "An Additive Algorithm for Solving Linear Programs with Zero-One Variables", J. ORSA 13, 517-546, 1965

<BB> R. Breu, C. A. Burdet, "Branch and Bound Experiments in 0-1 Programming", Math. Progr. Study 2, 1-50, 1974

<BE> J. F. Benders, "Partitioning Procedures for Solving Mixed-Variables Programming Problems", Numerische Mathematik 4, 238-252, 1962

<BJ> E. Balas, R. G. Jeroslow, "Canonical Cuts on the Unit Hypercube", SIAM J. of Appl. Math. 23, 61-69, 1972

<BM> B. Bouvier, G. Messoumian, "Programmes Lineaires en Variables Bivalentes", These, Faculte des Sciences, Univ. Grenoble, 1965

<GH> F. Granot, P. L. Hammer, "On the Use of Boolean Functions in 0-1 Programming", Meth. of OR 12, 154-184, 1972

<GJ> R. E. Gomory, E. L. Johnson, "Some Continuous Functions Related to Corner Polyhedra", Math. Progr. 3, 23-85, 1972

<GS> M. Guignard, K. Spielberg, "Propagation, Penalty Improvement and Use of Logical Inequalities in Interactive (0,1) Programming", Methods of OR 25, 157-171, Verlag Anton Hain, Meisenheim, 1977

<JS> E. L. Johnson, U. Suhl, "Enumeration with Cutting Planes", Working Paper IBM Research, Yorktown Heights N.Y., 1978

<LAD> A. H. Land, A. G. Doig, "An Automatic Method for Solving Discrete Programming Problems", Econometrica 28, 497-520, 1960

<LS> C. E. Lemke, K. Spielberg, "Direct Search Algorithms for Zero-One and Mixed Integer Programming", J. ORSA 15, 892-914, 1967

<MP.1> IBM Math. Progr. System Extended, (MPSX/370), General Information Manual, GH19-1090, 1976

<MP.2> IBM MPSX/370, Program Reference Manual, SH19-1095, 1976

<MP.3> IBM MPSX/370, Control Languages, SH19-1094, 1975

<MS> P. Mevert, U. Suhl, "Implicit Enumeration with Generalized Upper Bounds", Annals of Discrete Mathematics 1, 393-402, North-Holland, 1977

<PI> C. J. Piper, "Application of Combinatorial Programming to a Class of All-Zero-One Integer Programming Problems", Working Paper 115, Sch. of Bus. Adm., Univ. West. Ontario, 1974

<SAS> H. M. Salkin, K. Spielberg, "Adaptive Binary Programming", Rep. 320-2951, IBM N.Y. Sc. C., 1968

<SLS> L. Slate, K. Spielberg, "The Extended Control Language of MPSX/370 and possible Applications", IBM Systems Journal 17, 64-81, 1978

<SP.1> K. Spielberg, "Minimal Preferred Variable Methods in Zero-One Programming", to appear in J. ORSA, (revision of "Min. Pref. Var. Reduction for Zero-One Progr.", Rep. 320-3013, IBM Phil. Sc. C., 1972)

<SP.2> K. Spielberg, "Enumerative Methods in Integer Programming", to appear in Discrete Mathematics

<SP.3> K. Spielberg, "Minimal Inequality Branch-Bound Method", Rep. 320-3024 , IBM Phil. Sc. C., 1973

<SU> U. Suhl, "Entwicklung von Algorithmen fuer ganzzahlige Optimierungsmodelle ", Beitraege zur Untern. F., Heft 6, Freie Univ. Berlin, 1975

A Direct Dual Approach to a Transshipment Formulation for Multi-layer Network Problems with Fixed Charges

Monique Guignard
University of Pennsylvania

Kurt Spielberg
IBM

TECHNICAL REPORT NO. 43

August 1979

1. Introduction

We have given a "direct dual method" for the mixed (i.e., capacitated as well as uncapacitated) plant location problem in reference [3]. In the present paper we attempt to deal with a class of network problems with fixed charges via a transshipment formulation which can be viewed as an extension of the mixed plant location formulation. This will permit us to take advantage of some of the algorithmic tools which we developed in [3].

The overall approach is as follows. We are dealing with a structured mixed-(0,1) program. In such a case one may conjecture , e.g. from experience with the capacitated plant location problem, that the linear programming relaxation of the program can, with a certain effort, be formulated so as to be quite strong. If so, one can then expect to get good results for the mixed-(0,1) problem from an overall branch-bound or enumeration approach, especially if one has methods for obtaining good feasible integer solutions.

Therefore, in this paper, we put emphasis on (i) a "strong" linear programming (LP) formulation, (ii) a direct method for providing dual feasible solutions to the strong LP relaxation (applying LP methods in a standard manner is prohibitive on account of the huge dimension of any reasonably sized problem), (iii) methods for obtaining feasible integer solutions, and (iv) the generation and use of simple auxiliary integer constraints.

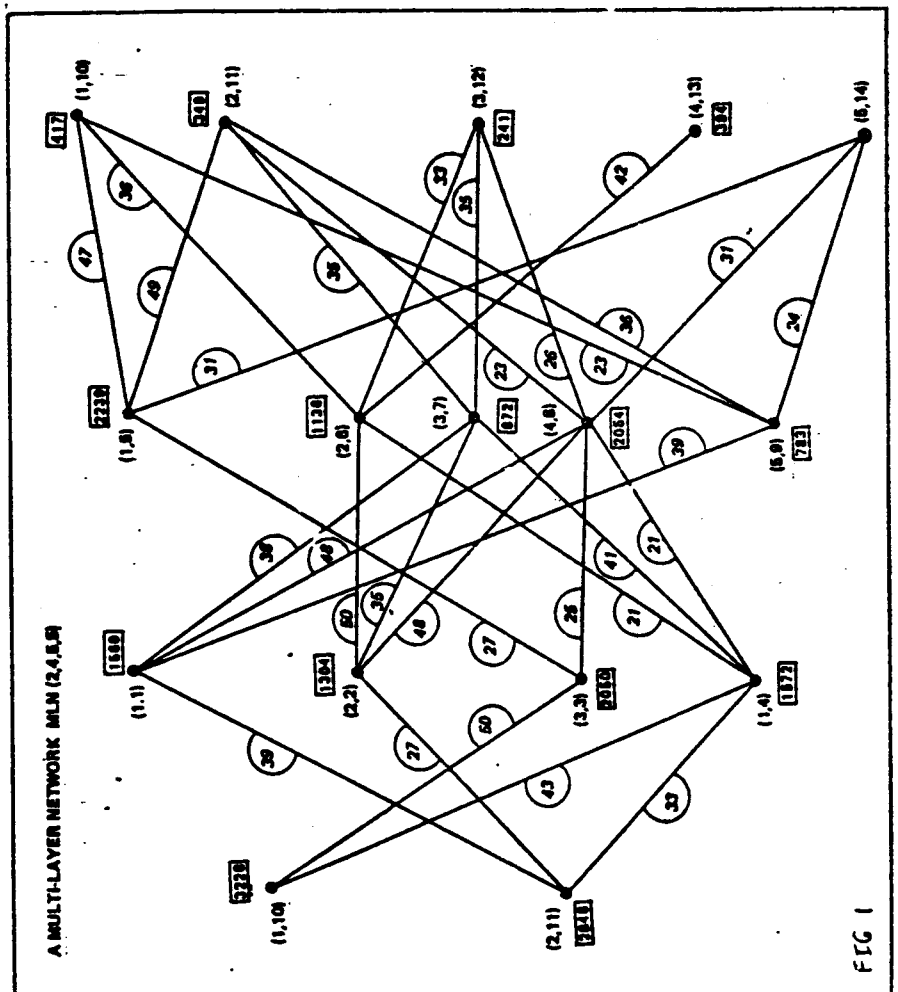
While the transformation "Network Problem" to "Transshipment

Problem" (with fixed charges) follows the well-known pattern from linear programming, the mixed integer programming nature of our problem appears to make certain special transshipment node constraints essential for any efficient solution method. In particular, we develop special constraints over the slacks between intermediate network node capacities and the shipments through these nodes. The associated dual variables have to be taken into account in the direct dual method. Similarly, we can derive logical constraints over the (0,1) variables at the source- and transshipment points. We can derive all of these constraints most easily for a special class of network problems which we shall term "multi-layer network problems", for which shipments are permitted only from one layer to the next.

In this paper, then, we concentrate on multi-layer network problems with fixed charges introduced by means of (0,1) variables on all nodes except in the last layer (the layer of demand points). The multi-layer set of problems can probably be enlarged easily to allow also shipments from the initial layer to any layer, and from any layer to the last layer. Eventually, we expect that completely general networks can also be handled (probably more directly) at only a moderate loss of algorithmic efficiency.

2. Multi-Layer Networks with Fixed Charges.

We shall illustrate the type of network which we identify as "multi-layer" via an example (see Fig. 1). The layers of nodes are



numbered from left to right. In our example there are four layers, a source layer, two transshipment layers and a demand layer. The notation MLN(NS;NN;ND) connotes the numbers of nodes per layer (the example is of the form MLN(2;4,5;5)):

NS ... no. of sources
(in our example there are 2 sources in the first layer)

ND ... number of demand points
(here: five demand points in layer four)

NN ... vector containing the numbers of points in successive transshipment layers (we allow an arbitrary number of transshipment layers)
(in our example NN = (4,5); there are 4 nodes in layer 2 and 5 nodes in layer 3)

2.1 Indexing of the Nodes.

We number (conceptually) the nodes in two ways:

1) the nodes in each layer are numbered consecutively from top to bottom, or
2) first, associate the index set T = (1,2,...,MT) with the MT transshipment nodes, counted from top to bottom within a layer, the layers taken in order from left to right.

Then, construct two index sets I and J:

. I, the index set for transshipment points plus source nodes:
I = (T, MT+1, MT+2, ... , MT + number of source nodes) .

. J, the index set for transshipment points plus demand nodes:
J = (T, MT+1, MT+2, ... , MT + number of demand nodes).

The resulting two index sets provide the indices for the transportation problem in section 3. In Fig. 1, the two numbers in parentheses, next to a node, give the indices under the above two systems.

2.2 Network Problem Data.

The source nodes have attached "availabilities", pertaining to a single commodity which is to be shipped through the network. In our example, the two availabilities are a(1)=3228, a(2)=2848. Similarly, there are "requirements" d(1)=417, d(2)=349,..., d(5)=270, at the last (demand) layer.

Flows of the commodity are permitted only from one layer to the next, and only between nodes which are connected by an arc in the network. In Fig. 1, we show the unit cost of shipment attached to the arc (e.g., c(11,2)=27; i.e., the cost of shipping one unit of commodity from source 2 to the second transshipment point is 27).

Each transshipment point has a capacity limitation on the flow through it, such as 783 for node 9 (the fifth transshipment point in layer 3). If there is no such restriction, then it is possible and desirable to compute such a number from the data (e.g., the flow through any transshipment point will be no larger than the sum of all requirements; in our case no larger than 1671).

There are even tighter restrictions; e.g., the capacity at node 5 should be replaced by 1036. Such a "tightening" of the problem data, either from a knowledge of the actual data, or from a careful inspection of the supplied data, is known to be important for all mixed integer programming problems (e.g., see [7,8,10,11]). It appears to be of an especially pronounced importance for the problems (and algorithms) which we dicuss here. We shall therefore return to this matter in section 4.

The obvious problem associated with Fig. 1 is that of shipping the requirements through the network to the demand points at minimal overall cost. We can easily add the further parameter $\mu(i,j)$ (not shown in Fig. 1) to any arc (i,j), to present an imposed upper bound of the flow. Again, if such bounds are not supplied (or if they are too loose) they can be computed (or recomputed) to some extent of tightness from the data themselves.

2.3 Fixed Charges.

Up to this point, the problem is a simple network problem and there exist extremely efficient algorithms for its resolution. What really interests us is the very much harder problem in which the "existence" (or usage) of all nodes other than the demand nodes is put in question. By this we shall mean that a (large) fixed cost $f(i)$ is associated with each node i of all layers except the last, so that it is meaningful to ask what subset of such nodes should be chosen so

that the requirements can be met at minimal overall cost (including the fixed charges).

The problem then is a mixed (0,1) problem of the general form:

$$\begin{aligned} &\min \quad f.y + c.x = z \\ &\text{s.t.} \quad F.y + C.x \leq b \qquad (2.1) \\ &\qquad y \in \{0,1\}^{n_1},\ x \geq 0 \end{aligned}$$

We shall write down a complete formulation of the problem in section 5, not in terms of Fig.1, but in terms of an equivalent bipartite transshipment problem. We have not specified any fixed charges $f(i)$ with nodes i in Fig. 1. For a uniform set of such fixed charges, of say \$ 1000.- on all nodes except the demand nodes, we give a solution to the problem (within 1.53 % of the optimum) in Fig. 2. Its cost is the sum of the fixed charges (\$ 6000.- for the 6 nodes of the solution) and the transportation costs (which add up to \$ 145546.-) for a total of \$ 151946.- .

3. The Bipartite Transshipment Problem with Fixed Charges.

The transformation of the network problem to a bipartite problem (i.e., a problem represented by a bipartite network (or graph)) is well known in the field of linear programming. We simply list all nodes other than those of the last layer as the source nodes of a transportation problem, with availabilities $a(i)$ equal to the given capacities (for transshipment points) or to the given availabilites (for the source nodes). Then we list all nodes other than the source

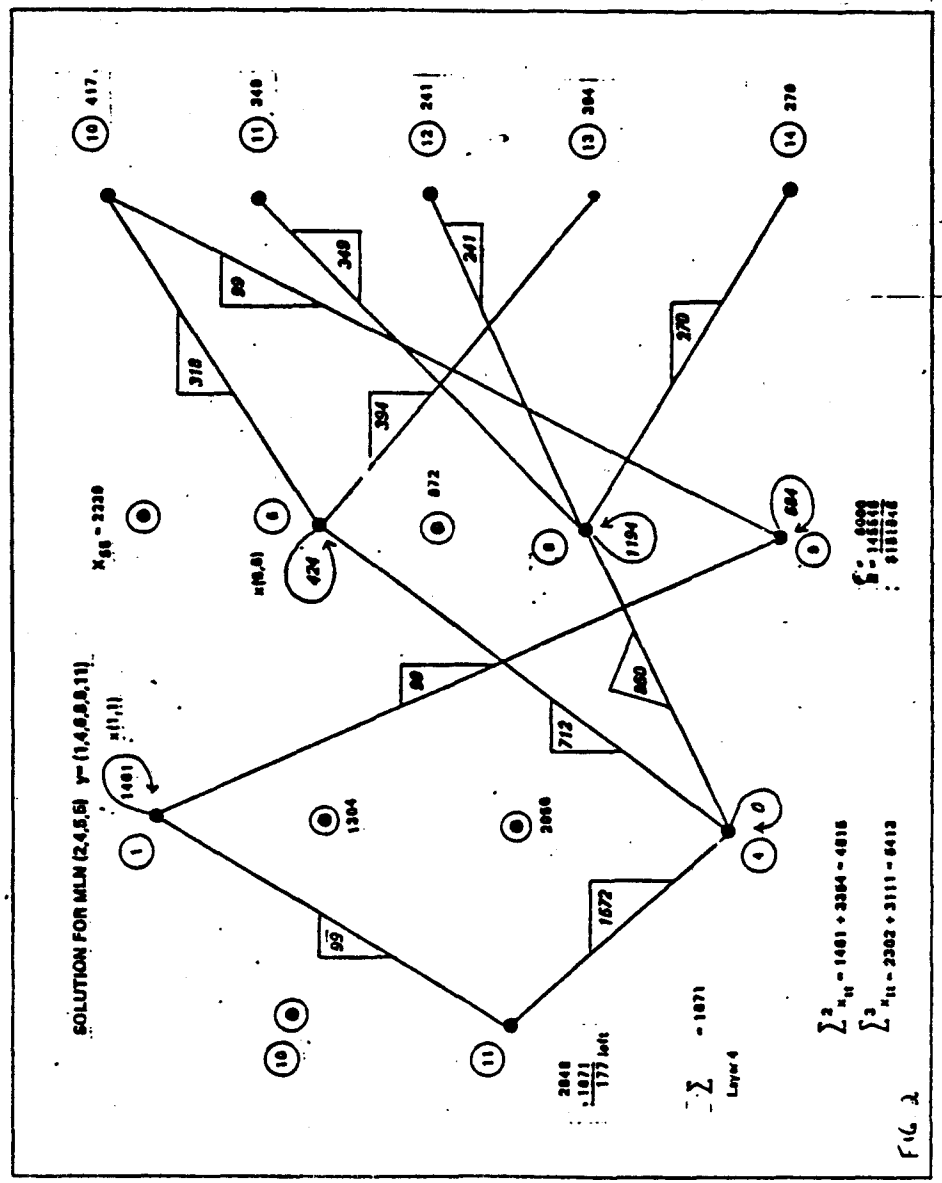


FIG. 2

nodes as the destination nodes of the transportation problem, with requirements b(j) equal to the capacities and the given requirements, respectively.

The order is not really material, but we shall list the transshipment points first in each instance. The matrix of the transportation costs c(i,j) is then filled in with the given costs where there are arcs, with prohibitively large costs where there are none, and with 0 where i and j both correspond to the same transshipment point.

3.1 The Special Treatment of Transshipment Points. Slacks x(t,t).

The last specification (setting c(i,j) = 0, when i = j = t, $t \in T = (1,2,\ldots,MT)$, the index set of the transshipment points), is most important for the solution of our problem. A "shipment" x(t,t), $t \in T$, will be interpreted, in a natural fashion, as the slack at transshipment point t, i.e., as the amount of the capacity a(t)=b(t) which is not utilized by the flow of the solution. If a solution has x(t,t) = a(t), then the node t is, de facto, not used, and all real requirements are actually satisfied by shipments via other transshipment nodes. In that instance, the fixed charge for node t should not be incurred. This, then, is the crucial fashion in which our transshipment formulation differs from that of the capacitated network problem. In the capacitated plant location problem all x(i,j)>0 lead to the imposition of a fixed charge for source i, in

our current problem this is so for all routes (i,j), except for the routes (t,t). The constraints and the solution methods must be modified to deal with this anomaly.

We list the data for our sample problem as a tableau in Fig. 3. There are MT=9 diagonal entries of 0 cost. The index sets I and J (for the transportation problem source and destination points) are partitioned as follows:

$$I = \{T , S\} \quad , \quad J = \{T , D\} \quad . \qquad (3.1)$$

The index set T, in turn, is partitioned to reflect the various transhipment point layers, into:

$$T = \{T1, T2, \ldots , TL\} = \{ T^{\ell} , \ell=1,\ldots,L \} \quad ,(3.2)$$

where L is the number of transhipment layers, i.e., the cardinality of the set NN.

3.2 Special Constraints for the Transshipment Model.

3.2.1 Structurally Essential Constraints.

Suppose one tried to apply the algorithm for Capacitated Plant Location to the problem of Fig. 3. One should then find that it would tend to open all "plants" i, $i \in T$. For, the algorithm works with constraints

$$\Sigma_j \; x(i,j) \leq a(i).y(i) \quad , \qquad (3.3)$$

and for i=j=t ($\in$T) this would imply, incorrectly, that the "pseudo" requirements b(j), $j \in T$, could usually be met only for y(t)=1.

Transshipment Formulation for MLN (2;4,5;5)

		T^1				T^2					D				
	j	1	2	3	4	5	6	7	8	9	10	11	12	13	14
	b_j	1560	1304	2050	1572	2239	1136	872	2054	783	417	349	241	394	270
i	a_i														
T^1	1	1560	0	-1	-1	-1	-1	-1	38	48	39	-1	-1	-1	-1	-1
	2	1304	-1	0	-1	-1	-1	50	35	48	-1	-1	-1	-1	-1	-1
	3	2050	-1	-1	0	-1	27	-1	-1	25	-1	-1	-1	-1	-1	-1
	4	1572	-1	-1	-1	0	-1	21	41	21	-1	-1	-1	-1	-1	-1
T^2	5	2239	-1	-1	-1	-1	0	-1	-1	-1	-1	47	49	-1	-1	31
	6	1136	-1	-1	-1	-1	-1	0	-1	-1	-1	36	-1	33	42	-1
	7	872	-1	-1	-1	-1	-1	-1	0	-1	-1	-1	35	35	-1	-1
	8	2054	-1	-1	-1	-1	-1	-1	-1	0	-1	-1	23	26	-1	31
	9	783	-1	-1	-1	-1	-1	-1	-1	-1	0	23	36	-1	-1	24
S	10	3228	-1	-1	50	43	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1
	11	2843	39	27	-1	33	-1	-1	-1	-1	-1	-1	-1	-1	-1	-1

Fig. 3 - 1 stands for prohibitively large unit costs c(i,j)

Hence, one needs to replace the summation over all j in J by a summation over $j \in J(t)$, the set of all j other than $j=t$. For $i=j=t$ this gives:

$$\sum_{J(t)} x(t,j) \leq a(t).y(t) \quad , \qquad (3.3)'$$

$$J(t) = \{j \mid j \in J,\ j \neq t\} \quad .$$

Similarly, if one considers the disaggregated form of (3.3), which is known to be of crucial importance for the strength of the linear programming relaxation for (2.1) (see [1, 3, 5]), one must limit j to $J(i)$ in

$$x(i,j) \leq m(i,j).y(i) \quad , \ i \in T,\ j \in J(i) \quad . (3.4)$$

One may wish to augment (3.4) by $x(t,t) \leq m(t,t)$. In the above, $m(i,j)=\min[a(i),b(j),\mu(i,j)]$. For points $t \in T$, one will always set $m(t,t)=a(t)=b(t)$.

Finally, it is necessary to apply a fixed charge $f(t)$ whenever there is a shipment out of transshipment point t. Therefore, it is necessary to enforce that $y(t)=0$ correspond to $x(t,t)=b(t)$, and that $y(t)=1$ correspond to $0 \leq x(t,t) < b(t)$. I.e., one requires the constraint:

$$b(t) - x(t,t) \leq b(t).y(t) \quad . \qquad (3.5)$$

3.2.2 Computationally Essential Constraints.

Constraints (3.3)' through (3.5) are necessary for the structural validity of the model, which would yield wrong results without them. We now turn to other constraints which do not add to the specificity

of the model, but which appear to be essential if the algorithm (certainly our algorithm, but most likely others as well) is to be efficient. One can demonstrate a serious degradation in algorithm behavior if such constraints are omitted or weakened by poor choice of the constraining parameters.

Most important appear to be upper bound constraints on the transshipment node slacks $x(t,t)$. It became clear to us soon, that a dual variable τ, associated with the single constraint

$$\sum_{T} x(t,t) \leq \lambda \quad , \qquad (3.6)$$

would be essential for the direct dual algorithm. We could see that choosing λ tightly was important, and could verify computationally that the loosening of the constraint (via choosing λ too large) would consistently and strongly degrade our algorithm.

We now replace (3.6), more strongly, by a set of equality constraints, one per layer ℓ:

$$\sum_{T^\ell} x(t,t) = \lambda^\ell , \qquad (3.7)$$

where T^ℓ is defined in the partitioning (3.2). The λ^ℓ can be determined from the problem parameters:

Proposition 1 (Multi-layer Network Slacks):

Let x be a feasible flow for the multi-layer network. Then, the sum of the slacks $x(t,t)$ in layer ℓ is equal to the total capacity of layer ℓ minus the total demand in layer D:

$$\lambda^\ell = \sum_{t \in T^\ell} b(t) - \sum_{j \in D} b(j)$$

Proof:

Let $t \in T^\ell$, $\ell \in \{1,2,..,L\}$ and let $P(t)$ be the set of predecessor nodes to t. Then, $\sum_{T^\ell} x(t,t) = \sum_{T^\ell} b(t) - \sum_{t \in T^\ell} \sum_{i \in P(t)} x(i,t)$,

in which the last term is equal to the total flow through level ℓ, and therefore equal to the total demand .

Note: While this is not necessary above, $P(t)$ can in fact be assumed to be non-empty, as we can remove all t without predecessors from the problem.

The dual variables π^ℓ, $\ell = 1,2,..,L$, associated with (3.7) , are unconstrained in sign. They are of great importance in several phases of our direct dual algorithm.

3.2.3 Flow Constraints.

One understands readily that (3.3') through (3.7) play the role of representing the transshipment points in the model, and are therefore essential for the integer programming aspects of the model (should a node be used by the flow, at the expense of a fixed charge $f(i)$?). But there are also a great number of constraints (essentially the classical equations of Kirchhoff from Physics) which capture the network flow nature (the linear programming aspects) of the model. We may list them in three forms:

1) Flow conservation:

$t \in T$: $\sum_{i \in P(t)} x(i,t) - \sum_{j \in S(t)} x(t,j) = 0$, (π_t)

$s \in S$: $-\sum_{t \in S(s)} x(s,t) \leq 0$. $\pi_s \geq 0$ (3.8)

$d \in D$: $-\sum_{t \in P(d)} x(t,d) \leq 0$. $-\pi_d \geq 0$

$P(i)$ and $S(i)$ are the sets of predecessor and successor nodes to node i. The dual variables are listed on the right, a parenthesis, as in (π), denoting that π is unconstrained in sign.

2) Node Inflow, Outflow Inequalities:

$t \in T$: $\sum_{i \in P(t)} x(i,t) \leq b(t)$, $\rho(t) \geq 0$ (3.9)

$t \in T$: $\sum_{j \in S(t)} x(t,j) \leq b(t)$, $\sigma(t) \geq 0$

3) Node Inflow, Outflow Equalities:

$t \in T$: $\sum_{i \in P(t)} x(i,t) + x(t,t) = b(t)$, $(-\hat{\pi}(t))$ (3.10)

$t \in T$: $\sum_{j \in S(t)} x(t,j) + x(t,t) = b(t)$. $(-\hat{\sigma}(t))$

We have made a preliminary analysis of the effects of introducing (3.8) through (3.10) on the dual problem. There are possibilities of exploiting the dual variables $(\pi, \rho, \sigma, \hat{\pi}, \hat{\sigma})$ so as t increase the dual objective function.

The task of choosing the "right" subset of these constraints and variables, and of implementing and testing the appropriate algorithmic tools, is quite substantial, and we have therefore deferred it for future work. It is clear, however, that these constraints will take on considerable importance in a general network framework (not of the multi- layer type). It should be

noted that the notation in (3.8) to (3.10) is oriented towards general networks. Using $(-\mathfrak{V})$ and $(-\mathfrak{H})$ in (3.10), by the way, is arbitrary and was chosen so as to have the $\mathfrak{V}$ and $\mathfrak{H}$ appear with the same sign in the constraints as do other similar variables.

4. Auxiliary Integer Variable Constraints, Data Preprocessing.

We have discussed the great importance of simple logical constraints over the (0,1) variables y(i) in a number of publications usually oriented towards enumeration (e.g., see [8,4,6]). Recent computational work establishes fairly conclusively that for a wide class of problems, which we would expect to include the problems we deal with here, enumeration may be able to outperform standard branch-and-bound techniques substantially ([10]). An alternative approach might be the introduction of logical techniques into branch-and-bound codes, so as to possibly achieve similar improvements.

In [3] we showed that one may include the logical (or also general integer) constraints also in the direct dual algorithm via an "auxiliary linear program". We therefore first describe the logical constraints for the multi-layer model. Then we show how the data of the problem can be tightened as an immediate consequence.

4.1 Auxiliary Integer Variable Constraints.

The integer constraints which we use are logical conditions which are transparently clear when stated in words, yet need nevertheless be supplied to the model to facilitate the computational phase. We shall

call them: flow-incidence constraints.

1) At least one of the suppliers of each demand point must be open.

$$1 \leq \sum_{P(d)} y(j)$$

2) At least one of the suppliers of each open transshipment point must be open.

$$y(t) \leq \sum_{P(t)} y(j), \quad \text{for all } t \in T$$

3) There is no need to open a plant unless one of its successors is also open:

$$y(t) \geq \sum_{S(t)} y(j), \quad t \in T^{\ell}, \ \ell = 1,2,..,L-1$$

4) In each layer enough plants should be open to let the total demand flow through. A logical constraint is derived from:

$$\sum b(t).y(t) \geq \sum b(d) \quad .$$

We shall illustrate these constraints for the problem of Fig. 1 further below. Here we note that the above conditions can be strengthened somewhat if the parameters a(i), b(j), $\mu(i,j)$ are taken into account. Logical inequalities can be derived easily form the given inequalities by "logical reduction" as treated, e.g., in [8].

4.2 Data Preprocessing.

In the following, let the capacity of a node i be denoted by c(i), even though in the transshipment description we use a(i) for $i \in I$ and b(j) for $j \in J$ (and even though we set the capacity to b(t) in the constraints, whenever a(t)=b(t)). The constraints of 4.1, which are flow-incidence constraints defining the predecessor and successor sets P(t) and S(t), permit the following implications:

1) The capacity of a transshipment point t is the minimum of its initial capacity and of the sum of the capacities over its predecessors:

$$c(t) \leftarrow\!- \min \{ c(t) , \sum_{p(t)} c(i) \} , \quad t \in T .$$

2) A transshipment point at the last level L can not have a throughput larger than the sum of the demands of its successors:

$$c(t) \leftarrow\!- \min \{ c(t) , \sum_{S(t)} d(j) \} , \quad t \in T(L) .$$

3) A transshipment point at an intermediate level can not have a throughput larger than the sum over the capacities of its successors:

$$c(t) \leftarrow\!- \min \{ c(t) , \sum_{S(t)} c(j) \} , \quad t \in T^1, T^2, \ldots, T^{L-1} .$$

4) Finally, a supply point need not have an availability larger than the sum of the capacities of its successors:

$$a(s) \leftarrow\!- \min \{ a(s) , \sum_{p(s)} c(j) \} , \quad s \in T$$

and a transshipment point at the first layer will not have a throughput larger than the sum of the availabilities of the connecting supply points:

$$c(t) \leftarrow\!- \min \{ c(t) , \sum_{p(t)} a(i) \} , \quad t \in T .$$

4.3 Illustrative example MLN(2;4,5;5)

. The auxiliary integer constraints are:

4.1.1

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

4.1.2

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

4.1.3

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

4.1.4

$$\begin{bmatrix} 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 1 \\ -1 & -1 & -1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & -1 & -1 & -1 & -1 & -1 & 0 & 0 \\ -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 & -1 \end{bmatrix} \leq \begin{bmatrix} 1 \\ 1 \\ 2 \\ 4 \end{bmatrix}$$

. the initial data vectors

a(i) = 1560 1304 2050 1572 2239 1136 872 2054 783 3228 2848

b(j) = 1560 1304 2050 1572 2239 1136 872 2054 783 417 349 241 394 270

are reduced as per 4.2 to

a(i)' = 1560 1304 1896 1572 1036 1052 590 860 783 3228 2848

b(j)' = 1560 1304 1896 1572 1036 1052 590 860 783 417 349 241 394 270

. the parameters λ , defined by (3.7), i.e. the sums over the multi-slack are given by:

$$\lambda^1 = 4661 , \quad \lambda^2 = 2650 .$$

5. Primal and Dual Constraint Sets.

We now can give a fairly comprehensive set of constraints for both the primal and dual LP relaxation to the transshipment form of the Multi-Layer Network problem with Fixed Charges. We believe that most of the constraints will be needed for an efficient algorithm. The emphasis in this paper will be on a direct (ascent) method for the dual problem.

5.1 The Primal Set of Constraints.

In the following, we give the primal constraints P1,P2,..., and list the associated dual variables on the right. The index sets are (briefly): I = (S,T); set of source nodes plus transshipment nodes; J = (T,D), set of transshipment nodes plus demand nodes; $J(i) = J - \{i\}$; K...set of indices correponding to arbitrary auxiliary integer constraints; T^{ℓ}... partitioning of T into L layers (see (3.2)); L...set of indices for the transshipment layers.

Objective Function (minimize):

$$z = \sum_I \sum_J c(i,j).x(i,j) + f(i).y(i)$$

Primal constraints:

			dual variables
P1:	$\sum_{J(i)} x(i,j) - a(i).y(i) \leq 0$	, $i \in I$	; $u(i) \geq 0$
P2:	$\sum_J x(i,j) \leq a(i)$	, $i \in I$	; $\mu(i) \geq 0$
P3:	$\sum_T \ell x(t,t) \leq \lambda(\ell)$	, $\ell \in L$	; $\tau(\ell) \geq 0$
P4:	$-\sum_I x(i,j) \leq -b(j)$,	$j \in J$	; $v(j) \geq 0$
P5:	$x(i,j) - m(i,j).y(i) \leq 0$,	$i \in I$, $j \in J(i)$	; $w(i,j) \geq 0$
P6:	$x(t,t) \leq b(t)$,	$t \in T$	; $w(t,t) \geq 0$
P7:	$-x(t,t) - b(t).y(t) \leq -b(t)$,	$t \in T$	; $\Omega(t) \geq 0$
P8:	$y(i) \leq \beta(i)$,	$i \in I$	; $t(i) \geq 0$
P9:	$\sum_I R(k,i).y(i) \leq r(k)$,	$k \in K$	; $\rho(k) \geq 0$
P10:	$x(i,j) \geq 0$,	$i \in I$, $j \in J(i)$	; $e(i,j) \geq 0$
P11:	$x(t,t) \geq 0$,	$t \in T$	; $e(t,t) \geq 0$
P12:	$y(i) \geq \alpha(i)$,	$i \in S$	; $g(i) \geq 0$
P13:	$y(t) \geq \alpha(t)$,	$t \in T$	; $g(t) \geq 0$

The parameters $\alpha(i), \beta(i)$ are of significance for the enumerative phase only. Essentially, one needs to consider merely the cases (0,1) and (1,1). For our purposes here, it is sufficient to take $\alpha(i)=\alpha(t)=0$ and $\beta(i)=\beta(t)=1$.

We do not at all claim that P1-P13 are, in any sense, a complete set of constraints. We have, e.g., left out the flow constraints of 3.2.3, and there may be other primal restrictions of importance. But we have computational results which indicate that the given set of constraints is a computationally fairly tight one.

5.2 The Dual Set of Constraints.

The great advantage of the dual system is the relative simplicity of its constraints. It will permit relatively easy procedures for the

construction of feasible solutions and the improvement of such solutions, when the corresponding tasks for the primal system would be much more formidable.

Objective Function (maximize):

$$ZD = +\sum b(j).v(j) + \sum b(t).\Omega(t) + \sum \alpha(i).g(i) - \sum a(i).\mu(i) - \sum \lambda(\ell).\tau(\ell) - \sum b(t).w(t,t) - \sum \beta(i).t(i) - \sum r(k).\rho(k)$$

The summations extend over all indices for which the dual variables are defined.

Dual constraints:

D1 - D9 : nonnegativity for $u(i)$, $\mu(i)$, $\tau(\ell)$, $v(j)$, $w(i,j)$, $\Omega(t)$, $t(i)$, $\rho(k)$.

D10: $e(i,j) = c(i,j) + \mu(i) + u(i) - v(j) + w(i,j) \geq 0$, $i \in \bar{I}, j \in J(i)$

D11: $e(t,t) = \mu(t) + \tau(\ell) - v(t) - \Omega(t) + w(t,t) \geq 0$, $t \in T$

D12: $g(i) = f(i) - a(i).u(i) - \sum_j m(i,j).w(i,j) + t(i) + \sum R(k,i).\rho(k) \geq 0$, $i \in S$

D13: $g(t) = f(t) - a(t).u(t) - \sum_{j \in J(t)} m(t,j).w(t,j) - b(t).\Omega(t) + t(t) + \sum R(k,t).\rho(k) \geq 0$, $t \in T$.

Note: $\tau(\ell)$ plays an essential role in place of $c(t,t)$ which is always 0. It prevents the blocking of the dual ascent procedure by the 0 cost entry.

6. Outline of a Direct Dual Algorithm.

We now proceed in the spirit of reference [3]. In a direct dual algorithm one executes a number of algorithmic steps, iteratively, until no more "improvement" is attained. In each of the algorithmic steps one starts with a dual feasible solution and computes increments to a subset of the dual variables, so as to:

1. retain dual feasibility, and
2. increase the dual objective function, or
3. increase the "resources" which are likely to permit a successful accomplishment of 2) in a future step of the direct dual method.

The basic idea of such an algorithm stems from O. Bilde and J. Krarup, in connection with the "Simple Plant Location Problem", see ref. [1]. These authors noticed that for the simple constraints:

$c(i,j) + w(i,j) - v(j) \geq 0$, and $g(i) = f(i) - \sum_j w(i,j)$

one could obtain a good (i.e., large) value of the dual objective function $ZD = \sum_j b(j).v(j)$, by (very roughly) the following "V-W Dual Ascent" procedure:

1. Initialize $v(j) = \min_I c(i,j)$, and then:
2. try to increase $v(j)$ in column j to the next larger value $c(i,j)$, if possible, by increasing $w(i,j)$ in row i, whenever the "resource" $g(i)$ is large enough to permit it.

   Repeat the attempt over all j until there is no longer any increase for any j.

One may say, intuitively, that the initial $g(i)$, namely $f(i)$,

is distributed among the w(i,j) with the purpose of permitting the objective function to be increased. Of course, there are arbitrary choices to be made, e.g. the order in which the columns j are processed. But the algorithm appears to be quite good , even independent of the order, for most data. And it is always possible to refine such an approach further, as was shown by D. Erlenkotter who added an algorithmic steps in which certain v(j) pairs are chosen so that the decrease in one is offset by an increase in the other with a resulting improvement of the objective function, ref. [2].

Our work on the more difficult capacitated plant location problem consisted of devising several more algorithmic steps. In some of them the dual objective function was increased, in others one obtained an increase in the g(i), which subsequently could be used as new initial values (new "resources") for another application of the V-W dual ascent procedure. We also added one algorithmic step, ALP: the "auxiliary linear program", in which a linear program was resolved over a subset of the dual variables , in particular including those which are associated with the explicitly present integer constraints.

During this work it became clear to us, that fruitful algorithmic steps could be found either:

(i) by inspection, or by

(ii) formulation and resolution of a linear program (if its dimension made this appear wise), or by examination of the various pivot steps which are locally possible and their translation into heuristic rules.

In this paper we shall not go into the details of the algorithmic steps which we introduced in[3], even if some algorithmic modifications were required for the multi-layer network. We shall, instead describe one important new algorithmic step (called, for want of a simpler name, the TNVM step, that is "transshipment node variable modification" step) in considerable detail.

We also introduce a bigger auxiliary linear program, XALP, even though it turns out to be somewhat more expensive than the ALP of [3]. It seems to us one may have to extend the size of the auxiliary LP even further (of course, nowhere near the size of the overall LP) if the structure of the network is to be captured in the algorithm. One should not be deterred by the fact that in the interpretive language APL (the medium of our experimental work) the execution of a linear program is relatively difficult.

6.1 Skeleton of the algorithm.

We first give the basic structure of our current algorithm, with only a few remarks. It should perhaps be emphasized that these computational steps are meant to be executed at every node of a branch-bound or (preferably) enumerative code. In general, therefore, there are free variables y(i), and variables y(i) which are fixed at 0 (node removed) and fixed at one (node "open"). The first case presents no difficulties, the second is best treated as that of a free variable

with a fixed cost of 0 (added into the objective function separately as a constant).

1. Preprocessing. (at a new node)
   (see refs [4,8,9,10] for details).
   . Execute logical tests over the integer inequalities
   . Discard node if problem is of degree 0 (logically infeasible).
   . Fix variables from degree 1 tests and "probing" (systematic exploitation of logical inequalities of degree 2).
   . Generate new flow-incidence matrix.
   . Reduce data a(i) and b(i) to reflect current network.
   . Recompute the new parameters λ^l.
2. Initialization.
   As will be seen, one can get remarkably far with a simple initial choice of the dual variables.
3. THVM computation.
   As will be described in some detail, this step focuses on the (t,t) routes, especially on the e(t,t) constraint. It considers various cases of fixing a subset of the variables in the constraint and of adjusting the others so as to satisfy the constraints and increase the objective function or increase the "resources" g(t).
4. V-W computation.
   This is the computation corresponding to the dual ascent of Bilde and Krarup. All dual variables are considered fixed, except v(j) and w(i,j). One works with modified "costs" c(i,j)' and tries to

increment the v(j) at the expense of increasing selected variables w(i,j). The dual constraints D11 through D14 (and, of course, nonnegativity) must be respected. The step differs from the one we gave in [3] only along the routes (t,t), in the sense that w(t,t) may be increased without decreasing the corresponding g(t).

5. THVM computation.
   The same calculation as under 3, within a loop which starts at point 4.
6. W-U interchange.
   As in [3] one attempts to replace the w(i,j) in row i by an increase in u(i), if this results in an increase of g(i). There are two minor modifications and additions. When the change comes from a w(t,t), one has an increase in the dual objective function instead of the increase in the g(i) ; but this appears to occur only rarely. Also, one may attempt to replace an increase in g(i) by a decrease in t(i), which again causes a direct increment of the objective function.
7. U-W interchange.
   Again as in [3], one attempts a partial replacement of some u(i) by suitable increases in the w(i,j), if this produces positive results. The modifications for the multi-layer network case are similar to those described under 6. above. Terminate if no change in 4,5,6 or 7. (We test only for changes in 6 or 7, at present).
8. XALP, Extended Auxiliary Linear Program.
   In this important algorithmic step one focuses on a manageable

subset of the variables (keeping those which would blow up the problem intolerably, in particular the variables w(i,j) and e(i,j), fixed) and resolves the resulting linear program completely. This step is the only one (so far) which takes into account the dual variables ρ^k associated with the side constraints on the integer variables. Also, this step (starting out with many variables in the interior of their bound intervals) will place at least a subset of variables at the boundary of the feasible region. This appears to be quite important if the orthogonality conditions are to be used for finding primal solutions.

Go to 4.

Note: We can terminate the computation within XALP (for which we have set up a special primal algorithm), should the computational effort be excessive.

The above are certainly not all conceivable interchange steps. We have experimented with some others. But this is a matter in which one must proceed with some caution if the algorithm is to be manageable and fast. We allude to some further improvements later on.

6.2 Some Detailed Algorithmic Steps.

In this section we give details for the steps 1 (initialization, 3 and 5 (TNVM), and XALP.

6.2.1 Initialization.

One can easily construct a dually feasible solution which generally produces a good approximation to the final dual objective function. Note that preprocessing is a prerequisite for example to avoid that a(i) is 0 for $i \in I$.

$$
\begin{aligned}
&u(i) = f(i)/a(i) && , i \in I\\
&\mu(i) = 0 && , i \in I\\
&v(j) = \min_{i \in I}\left[c(i,j) + f(i)/a(i)\right] && , j \in J\\
&v(t) = \min_{i \to t}\left[c(i,t) + f(i)/a(i)\right] && , t \in T\\
&w(i,j) = 0 && , i \in I,\ j \in J(i)\\
&\gamma(\ell) = \min_{T(\ell)} v(t) && , \ell \in L\\
&w(t,t) = v(t) - \gamma(\ell) && , t \in T(\ell)\\
&e(t,t) = 0 && , t \in T
\end{aligned}
$$

6.2.2 The TNVM (transshipment node variable modification) Step.

A. We consider the dual variables which appear in 911, as well as the slacks g(t). We explicitly focus on the possible increments $\Delta g(t)$, $\Delta Q(t)$, $\Delta w(t,t)$, $\Delta\mu(t)$ and $\Delta\gamma(\ell)$, keeping all other dual variables and the slacks e(t,t), not necessarily the e(i,j), $i \neq j$, constant.

We intend to increase the objective function, i.e., to maximize a linear function in the increments. This function is separable in ℓ, so that one can consider the case for a particular layer ℓ.

Let $\bar{\mu}(t) = \min\{\mu(t), \min_{j(t)} e(t,j)\}$. The problem can be formulated as the following linear program:

Maximize: $\Delta ZD(\ell) = \sum_{T\ell} b(t).((\Delta\Omega(t) - \Delta w(t,t) - \Delta\mu(t)) - \lambda(\ell).\Delta\gamma(\ell)$

Subject to : (for all $t \in T(\ell)$)

$$-\Delta\Omega(t) + \Delta w(t,t) + \Delta\mu(t) + \Delta\gamma(\ell) = 0$$

$$\Delta g(t) + b(t).\Delta\Omega(t) = 0 \quad , \text{ with the bounds:}$$

$$\Delta g(t) \geq -g(t)$$

$$\Delta\Omega(t) \geq -\Omega(t)$$

$$\Delta w(t,t) \geq -w(t,t)$$

$$\Delta\mu(t) \geq -\bar{\mu}(t)$$

$$\Delta\gamma(\ell) \geq -\gamma(\ell) \quad .$$

Note that the problem has the "current" values, i.e. the set of zero increments, as a feasible solution , which is basic but not at an extreme point (in general, most lower bounds are not 0). We do not solve the LP, even though this could be done, but look to simple pivot steps for a possible improvement.

We consider three cases in turn. Other possibilities exist but look distinctly less promising.

We give details for the second case.

Case 1.

Take $\Delta\mu(t)$ and $\Delta g(t)$ as the basic variables. Denote the total overall demand by B, $B = \sum_{J} b(j)$.

Then $\Delta ZD(\ell) = B \cdot \Delta\gamma(\ell)$.

Set $\Delta\gamma(\ell) = \min_{T\ell} \bar{\mu}(t)$, and

$\Delta\mu(t) = -\min_{T\ell} \bar{\mu}(t)$.

The objective function increase is: $\Delta ZD(\ell) = B \cdot \min_{T\ell} \bar{\mu}(t)$.

Note that even a small increment in the variables can lead to a substantial increase in the objective function on account of the multiplying factor B.

Case 2.

Take $\Delta\Omega(t)$ and $\Delta g(t)$ as the basic variables.

Set: $\Delta\gamma(\ell) = \min_{T\ell} g(t)/b(t)$

$\Delta g(t) = -b(t).\min_{T\ell} g(t)/b(t)$

$\Delta\Omega(t) = \min_{T\ell} g(t)/b(t)$

Then: $\Delta ZD(\ell) = B \cdot \min_{T\ell} g(t)/b(t)$.

Proof:

We eliminate $\Delta\Omega(t)$ from the second constraint and have the "current" constraint set:

$$\Delta\Omega(t) - \Delta w(t,t) - \Delta\mu(t) - \Delta\gamma(\ell) = 0$$

$$\Delta g(t) + b(t).(\Delta w(t,t) + \Delta\mu(t) + \Delta\gamma(\ell)) = 0 \quad .$$

The first constraint allows us to express the objective function in terms of the $\Delta\gamma(\ell)$ only.

$\Delta ZD(\ell) = \sum_{T\ell} b(t).\Delta\gamma(\ell) - \lambda(\ell).\Delta\gamma(\ell)$. From proposition 1, $\lambda(\ell) = \sum_{T\ell} b(t) - \sum_{D} b(j)$, hence $\Delta ZD(\ell) = B \cdot \Delta\gamma(\ell)$.

In the pivot step one therefore wishes to increase the nonbasic variable $\Delta\gamma(\ell)$. The other nonbasic variables $\Delta w(t,t)$, and $\Delta\mu(t)$ remain fixed. As $\Delta\gamma(\ell)$ increases, $\Delta\Omega(t)$ must increase (no problem, as there is no upper bound on it) and $g(t)$ must decrease. The decrease of $\Delta g(t)$ is limited below by $-g(t)$. Therefore, from the second constraint, $\Delta\gamma(\ell)$ is limited above by $g(t)/b(t)$, for all t, and $ZD(\ell) = B.\Delta\gamma(\ell) = B.\min_{T\ell} g(t)/b(t)$. ∎

Case 3. (the most successful case, in general)

Take as basic variables: $\Delta w(t,t)$ and $\Delta g(t)$.

$\Delta ZD(\ell)$ becomes $\theta.\Delta\tau(\ell)$, again:

Set: $\Delta\tau(\ell) = \min_{T(\ell)} w(t,t)$, and

$\Delta w(t,t) = -\min_{T(\ell)} w(t,t)$.

Then: $\Delta ZD(\ell) = \theta \cdot \min_{T(\ell)} w(t,t)$.

Our analysis for the other possibilities of pivot steps seems to show that the corresponding changes would be dominated by the ones described above.

B. As a final part of the THVM step we place the remaining $w(t,t)$ into $\mu(t)$:

Set: $\mu(t)$ to $\mu(t) + w(t,t)$.

Set: all $w(t,t)$ to 0.

The objective function remains unchanged, but $\mu(t)$ proves a useful resource in the U-W calculation.

6.2.3 XALP, An Extended Auxiliary Linear Program.

We impose the following restrictions:

R1: all the $u(i)$ and $v(j)$ are incremented by δ

R2: all $\mu(s)$, $s \in S$, are unchanged

R3: all $\Omega(t)$ are variables in XALP, $t \in T$

R4: all $\mu(t)$ are variables, $t \in T$

R5: all $w(t,t)$ are variables, $t \in T$, all other $w(i,j)$ are unchanged

R6: all $\tau(\ell)$ are variables, $\ell = 1,2,\ldots,L$

R7: all $\rho(k)$ are variables, $k \in K$

R8: all $g(i)$ and $t(i)$ are variables, $i \in SUT$.

Let initial values be indicated by a superscript 1. Let Δx stand for the increment of x (from its initial value x^1).

Proposition 2 (Extended Auxiliary Linear Program):

Under restrictions R1-8, the maximal increase, Δ, in the dual objective function is given by the auxiliary LP:

$$\min \quad \Delta = \Sigma_T a(t).[\Delta\mu(t) + (\beta(t)-1).\Delta\Omega(t) + \Delta w(t,t)] + \Sigma_L \Delta\lambda(\ell).\Delta\tau(\ell) + [\Sigma_I \beta(i).a(i) - \Sigma_J b(j)].\delta + \Sigma_K (r(k) - \Sigma_I \beta(i).R(k,i)).\Delta\rho(k) + \Sigma_I (\beta(i)-\gamma(i)).\Delta g(i).$$

subject to:

$$\Delta\mu(t) + \Delta\tau(\ell) + \Delta w(t,t) - \delta - \Delta\Omega(t) - \Delta e(t,t) = 0 \ , \ \forall t \in T$$

$$-\Delta g(s) + \Delta t(s) - a(s).\delta + \Sigma_K R(k,s).\Delta\rho(k) = 0 \ , \ \forall s \in S$$

$$-\Delta g(t) + \Delta t(t) - a(t).\delta - b(t).\Delta\Omega(t) + \Sigma_K R(k,t).\Delta\rho(k) = 0 \ , \ \forall t \in T$$

$$\Delta g(i) \geq -g(i)^1 \ , \ \Delta t(i) \geq -t(i)^1 \ , \ \Delta\mu(t) \geq \max\{-\mu(t)^1, \max_{s \neq t} -e(t,j)^1\} \ ,$$

$$\Delta w(t,t) \geq -w(t,t)^1 \ , \ \delta \geq \max\{\max_T -u(t)^1, \max_S -u(s)^1, \max_J -v(j)^1\} \ ,$$

$\Delta\gamma(\ell) \geq -\gamma(\ell)^1$, $\Delta\Omega(t) \geq -\Omega(t)^1$, $\Delta e(t,t) \geq -e(t,t)^1$, $\Delta\rho(k) \geq -\rho(k)^1$.

If $\overline{\Delta}$ is the optimal value for Δ, then the new dual objective function value is $ZD^1 - \overline{\Delta}$, and is no less than ZD .

Proof:

D10 must be satisfied:

$c(s,j) + u(s) + \mu(s) - v(j) + w(s,j) = c(s,j) + u(s)^1 + \delta + \mu(s)^1 - v(j)^1 + w(s,j)^1 \geq e(s,j)^1 \geq 0, \forall s \in S,$

$c(t,j) + u(t) + \mu(t) - v(j) + w(t,j) = c(t,j) + u(t)^1 + \delta + \mu(t) - v(j)^1 - \delta + w(t,j)^1 = e(t,j) + \Delta\mu(t) \geq 0$, iff $\Delta\mu(t) \geq -e(t,j)^1, \forall t \neq j.$

D11, D12 and D13 translate into the constraints listed. $u(i) \geq 0$, $v(j) \geq 0$ translate into $u(s)^1 + \delta \geq 0$, $u(t)^1 + \delta \geq 0$, $v(j)^1 + \delta \geq 0$, and yield the given bound on δ.

All other variables must remain nonnegative and the changes cannot be smaller than the negative of the initial values of these variables .

Note:

For larger dimensions, the size of this ALP may become prohibitive with limited workspace size in APL. We have, therefore, also experimented with a restricted ALP, in which only changes in u(i), v(j), ρ, g(i) and t(i) are allowed , i.e.:

$\Delta\mu(t) = 0$, $\Delta\gamma(\ell) = 0$, $\Delta\Omega(t) = 0$, $\Delta w(t,t) = 0$.

7. Computational Results. Improvements.

We summarize computational results for five experiments with four test problems in a table. The first problem is our illustrative problem. The other problems were generated in a pseudo-random fashion which we describe below.

7.1 The Data.

The data generation process which we used is as follows. One enters a sparseness parameter between 0 and 1. We found that .6 or .7 was required to render the generated problems feasible. One enters the number of layers and the number of nodes per layer; such as [2,4,5,5] for the sample problem MLN(2;4,5;5).

The program then generates the network layer by layer, starting with the supplies. It associates with each node in a layer a certain number of outgoing arcs, according to the density parameter. It checks for any node in the next layer which is not reached by an arc, and generates additional arcs if necessary (quite likely for densities below .5).

One enters a range R(a) for the availabilities. The program generates randomly the availabilities within the range, and computes an average availability per demand point.

Now one enters a range for the demand small enough (guided by the above) so that the problem:

(i) is likely to be feasible, and

(ii) is likely to have enough slack as to be combinatorially interesting.

E.g., if the average availability per demand point is 700, we entered a demand range R(d) = |300,500|. Demands are then generated randomly within R(d). Should the total demand exceed the total availability, in spite of the precautions taken, one is prompted to provide a different R(d).

The program generates arcs in accordance with the density parameter, and assigns random costs within a specified range. Finally, capacities are generated for the transshipment points, large enough to render the problem feasible, but not so large as to render it uncapacitated.

7.2 The Experiments.

We present a set of computational results in tabular form. As can be seen from the values of the GAP (defined as: ((best ZP value) - (best lower bound))/(best ZP value), we have no difficulties at all with the first three problems. For the most difficult fourth problem, with 60 zero-one variables, we give two columns. For the calculations in the fourth column we used a dual linear program (for XALP) which was too slow to permit inclusion of the side constraints (RO-variables). For the last column we used a special primal algorithm dealing with a nonbasic starting solution, so that side constraints could be accommodated. The final lower bound becomes somewhat better,

Computational Results

Problem	2,4,5,5	4,5,5,10	4,10,10,10,30	10,20,30,20 dual, no RO	10,20,30,20 primal, RO	
F(I)	1000	10,000	100,000	100,000	100,000	INIT.
Initial TNVH	131,333	209,060	2,374,257	2,108,645	-----	
	"	"	"	"	-----	
V-W Ascent	138,080	235,631	2,613,840	2,244,234	-----	1ST ITER.
TNVH	"	"	2,645,222	2,268,616	-----	
U-U	-, "	-, "	-, "	-, "	-----	
U-W	+, "	+, "	+, "	+, "	-----	
XALP	148,138	243,562	2,983,425	2,357,674	2,448,759	
V-W Ascent	"	"	3,032,655	2,415,816	2,464,367	2ND ITER.
TNVH	"	"	"	"	"	
U-U	-, "	-, "	+, "	-, "	-, "	
U-W	+, "	+, "	+, "	-, "	+, "	
XALP	"	244,599	3,114,189	2,425320	2,534,046	
			.....			
Final ZD	148,138	244,622	3,140,767	2,521,335	2,579,992	FINAL DUAL RELAXED O.F.
ZRL	149,227	254,850	3,499,763	2,521,850	"	
ZP(ZRL)	151,546	298,818	5,742,451	3,978,864	4,295,856	PRIMAL
Best ZP	"	280,157	3,613,359	3,652,669	3,666,745	INTEGER SOL
GAP	1.5%	9.03%	3.14%	30.96%	29.6%	

but does not translate into a better primal solution. By replacing the greedy step for finding a primal solution (which requires resolution of some 4 to 6 transportation problems), by the initial phases of a local enumerative scheme over the integer variables for which both g(i) and t(i) are 0 in the final dual solution, the gap can be lowered to 15% at the expense of the solution of some 15 transportation problems. From this point on, an overall enumerative scheme should have little difficulty in resolving the problem completely.

In the table, the f(i) are the fixed charges which we used. The unit costs c(i,j) are in the range from 0 to 500. The other entries in the table are usually the values of the dual objective function.

We always executed a preprocessing step which reduced the data. As a consequence, the initial solution was usually quite good. If one had no good initial solution, the rest of the algorithm would catch up fairly quickly, albeit at some additional computational expense.

Applying the TNVM step after the initialization gave no improvement, and could probably be omitted. Within the iterative scheme, however, all computational steps come into their own at some part of the algorithm. The number of iterations varied between 3 and 7. The WU and UW steps, while they did not produce improvements in the dual objective function, were nevertheless quite important in that they produced increases in the g(i) (marked in the table by a +). These increases translated, in the other algorithmic steps, into increases of the dual objective function which could not have been realized otherwise.

After a final value for the dual objective function has been attained (final ZD), we solve one Lagrangean relaxed problem in which the final dual variables are used to put all constraints other than the transportation problem constraints into the objective function. The corresponding lower bound on the objective function is denoted by ZRL.

The relaxed problem also yields an integer solution (with objective function value ZP(ZRL)). But we usually did better by computing a primal solution from a greedy enumerative phase. As already pointed out, we extended this local enumerative phase somewhat for the last problem, to produce a gap of less than 15%.

7.3 Possible Improvements and Generalizations.

There are many other possibilities for dual variable changes. We are at present experimenting with:

(I) A XSI-V Ascent.

One deal with the primal flow conservation constraints (3.8) and the dual variables ξ. We first increase the v(d), with an immediate effect on the objective function. Then we increase the v(s) and v(t), obtaining no increase in ZD but rather a "more balanced situation" among the dual variables. In essence, the network structure appears to be better represented by this algorithmic step than was possible before. It is also clear, that any general network algorithm would require such an algorithmic step to represent the network structure.

(II) A RO dual ascent.

We concentrate on the side constraints and the $\rho(k)$, which up

to now were altered only in the XALP step. One can obtain noticeable improvements in the dual objective function without employing the full LP formalism of XALP.

In summary one may say, after some preliminary computations, that (I) and (II) together have roughly the same effect as XALP, but are much faster. Overall, there is a slight improvement in the dual objective function, but not as much as one would like to see.

One further improvement would be to use L dual increments $\delta(\ell)$ instead of one increment δ in the auxiliary LP. The resulting relaxation should be beneficial.

We also consider solving one Lagrangean problem per iteration. It is known that the coresponding ZP(ZRL) are not monotonic (as is ZD), so that one might get good primal solutions along the way. We find that we can initialize the Lagrangean problems efficiently so that the resolution of the relaxed problem (the transportation problem with modified costs) is fairly inexpensive.

Finally, of course , one needs to generalize the model itself. A first step might be to permit shipments from the first (source) layer to any layer, and from any layer to the demand layer. Then, one might wish to treat completely general networks. Most likely, the major difficulty will be to get good bounds on the sum of the shipments x(t,t) over certain subsets of T as suggested by the network structure.

8. References.

1. O. Bilde, J. Krarup, "Sharp lower bounds and efficient algorithms for the simple plant location problem", (based on a Technical Report in Danish, of 1967), Annals of Discrete Mathematics 1 (1977).
2. D. Erlenkotter, "A dual-based procedure for the uncapacitated plant location problem", Operations Research 26 (1978).
3. M. Guignard, K. Spielberg, "A direct dual method for the mixed plant location problem with some side constraints", Mathematical Programming 17 (1979).
4. M. Guignard, K. Spielberg, U. Suhl, "Survey of enumerative methods for integer programming", Proceedings, SHARE 51, Aug. 1978.
5. M. Guignard, K. Spielberg, "Search techniques with adaptive features for certain integer and mixed-integer programming problems", Proceedings, IFIPS Congress, Edinburgh, North Holland, 1968.
6. M. Guignard, K. Spielberg, "The state enumeration method for mixed zero-one programming", IBM Phila. Sci. Rep. 320-3025, 1977.
7. E.L. Johnson, U. Suhl, "Experiments in integer programming", IBM Research Report, submitted to Applied Discrete Mathematics , 1979.
8. K. Spielberg, "Logical reduction methods in zero-one programming (Minimal preferred variables)", based on IBM Phila. Sci. Rep. 320-3013, 1972), accepted for publ. by Operations Research.
9. K. Spielberg, "Enumerative methods in integer programming", Annals of Discrete Mathematics 3 (1979).
10. K. Spielberg, U. Suhl, "An efficient experimental enumeration program for 0-1 problems within MPSX/370", to appear as IBM Research Report, RC... , 1979.
11. H.P. Williams, "Experiments in the formulation of integer programming problems", Mathematical Programming Study 3 (1974).

Logical Reduction Methods in Zero-One Programming

(Minimal Preferred Variables)

Monique Guignard

University of Pennsylvania

Kurt Spielberg

IBM Corporation

White Plains

Abstract

In the first part of the paper, the concept of logical reduction is presented. Minimal preferred variable inequalities are introduced, and algorithms are given for their calculation. A simple illustrative example is carried along from the start, further examples are provided later. The second part of the paper describes certain properties of the generated logical inequalities. It then explains some of the decreases of computational effort which may be achieved by the use of minimal preferred inequalities and outlines a number of concrete applications with some numerical results. Finally, a number of more recent concepts and results are discussed, among them the notion of "probing" and a related zero-one enumeration code for large scale problems under the extended control language of MPSX/370.

All enumeration and branch-bound algorithms for zero-one programming require insight into the structure of the given problem if the search is to be directed in a rational manner. An important tool for gaining such insight is the generation of systems of logical (Boolean, canonical, preferred) inequalities by means of simple logical and arithmetic operations. We denote such an overall procedure, which is strictly delimited in scope and does not include the combination of rows (other than with faces of the unit hypercube), by the term logical reduction.

The idea of using "minimal preferred inequalities" to identify "preferred variables" for branching in an enumerative algorithm (a "weak" form of reduction, in the sense that a "zero" origin is presupposed, i.e., branches are restricted to setting variables to 1) was introduced by Lemke, Spielberg [20]. The subject of the present paper (a somewhat stronger form of reduction and its manifold practical applications) has been developed by the authors in a number of working papers. For a description of a small part of this work see [13]; see also [11], primarily for extensions to integer programming.

Independently, strong reduction has also been developed by Hammer

[15], Granot, Hammer [8] and Balas, Jeroslow [1], with the emphasis more (but not solely) on the characterization of the solution sets and polytopes of (0,1) problems.

There are also papers which are concerned with replacing one inequality by a stronger one or by an equivalent and simpler inequality, a good example being Bradley, Hammer, Wolsey [3] (as pointed out by a referee). However, while such methods can be important, they do not compete with reduction as presented here. They can possibly be employed as a first step before reduction.

It must be emphasized that logical reduction is meant to be invoked within computer programs for the resolution of zero-one problems. That this is important is still not widely realized, even though the described work already dates back a number of years. We can cite Williams [27] and his "mining example", which we shall carry along for illustrative purposes. Reference [27] recommends, quite rightly, that the mining example constraint (MEC)

$$-y2 - y3 - y4 - y5 + 4\,y1 \leq 0$$

be replaced by the four constraints

$$y2 - y1 \geq 0,\ y3 - y1 \geq 0,\ y4 - y1 \geq 0,\ y5 - y1 \geq 0.$$

The second formulation is then shown to be much superior to the first,

and it is advocated that problem formulation take such matters into account.

In the context of the current paper, the new set of constraints are "logical" (i.e., all coefficients are 0, 1 or -1) inequalities derived from the initial constraint. Logical reduction will automatically generate these simple inequalities (as the minimal preferred inequalities associated with the given constraints) and any enumerative or branch and bound procedure oriented towards the use of logical inequalities would exploit them successfully. That is, the legitimate intent of [27] can be carried out without user intervention by the incorporation of automatic logical reduction procedures into the computer software designed to handle such problems. More complex and yet potentially useful structures of a problem, which may often not be as easy to detect as in the above example and which also may arise only within the enumeration procedure, can often be detected and exploited by the logical reduction process.

The mining example, by the way, is really quite special. As the author of [27] points out, the new set of inequalities (in the "disaggregated" form, as we may say) is totally unimodular so that solving a linear program over it would result in the correct integer solution. Extrapolating from this behavior, one is tempted to suggest as an obvious use of logical reduction the addition of the generated logical constraints to the given constraints. Our initial experiments in such a direction, on relatively small problems, did not yield particularly good results and we therefore advocated the direct logical exploitation of the minimal preferred inequalities instead. This assessment, however, has been shown to have been too pessimistic, especially by recent work of E.L. Johnson, together with H. Crowder, M. Padberg, U. Suhl, (private communication). They have found the addition of suitable logical inequalities to a given general constraint set to be crucial in the resolution of some very large 0-1 problems by a production mixed integer code. Also, for very special problems, in particular the plant location family, disaggregation (replacement of one inequality by many) has been known to be an important improvement for many years (see the surveys [12,19], for example). Just as with the MEC, we can show, in this instance, that the disaggregated form follows from the aggregated form automatically, by what we shall call "strong reduction" in this paper.

At present, then, it appears that the methods of this paper have

broad and important applicability (see section 3 for further details):

a) low degree inequalities, derived automatically from the constraint set (including generated Benders constraints) allow the fixing of many 0-1 variables outright (especially by "probing" as suggested in Guignard, Spielberg, Suhl [14]).

b) in addition to a) the inequalities can be used to guide the search; moreover the logical procedures can be employed in large scale enumeration codes, using efficient data structures within the environment of a production LP system, as in Spielberg , Suhl [26] (also Johnson, Suhl [18], who exploit more general Gomory-Johnson cutting planes [7]).

c) massively large travelling salesman problems (up to more than 300 cities) have been resolved completely by a branch-and-bound code after suitably chosen logical inequalities (more complex than the ones described here) have been generated and added to an original formulation of the problem; see Crowder, Padberg [5].

d) large mixed integer problems, with thousands of continuous and hundreds of 0-1 variables, are clearly prime candidates for logical reduction (over Benders inequalites (see [2]), for example), since the logical reduction work will be negligible compared to the other data-processing chores which are necessary; in some sense, the excellent results of Mairs, Wakefield, Johnson, Spielberg [21] can be viewed in this manner (another way of viewing them is as an example of successful disaggregation).

e) the addition of logical inequalities to the given constraint set (after full logical exploitation as under a)) can render large general problems easy (or easier) for production branch-bound codes.

f) normal penalty (especially reduced cost) information can often be combined with logical information to lead to substantially strengthened penalties and bounds (see the notions of "propagation", "additivity" and "persistence").

1. (Minimal) Preferred Inequalities.

Definition.

Consider a linear system of m inequalities in n 0-1 variables $y(1),\ldots,y(n)$:

$$A \cdot y \leq b \qquad . \qquad (1)$$

Combining the i th row of (1), i = 1,...,m , with the hypercube constraints $0 \le y(j) \le 1$, one can usually derive a set of constraints of the form

$$\sum_{j \in E(i)} \tilde{y}(j) \geq 1 \qquad , \qquad (2)$$

where $\tilde{y}(j)$ is either $y(j)$ or $\bar{y}(j)=1-y(j)$, and is called a preferred variable, $E(i) \subset J = \{1,2,\ldots,n\}$.
Inequality (2) is called a preferred variable inequality. (abbreviated p.i.). Setting a $\tilde{y}(j)=1$ is called a preferred branch, and the corresponding value of y(j) is the preferred value.
E(i) is called the preferred (index) set.
d(i), the minimal cardinality of all derived E(i), is called the preferred weight (or degree) of row i, and the sets E(i) with cardinality d(i) are minimal preferred sets of row i
(d(i) is, in a sense, a measure of the tightness of the i-th constraint in 0-1 variables).
Occasionally a row i yields no meaningful preferred set (e.g. $y1+y2 \leq 2$). The degree is then set to the default value n+1.
Let $d = \min d(i)$, i = 1,...,m . The totality of derived preferred inequalities of size d will be called minimal preferred inequalities (or: m.p.i.'s) of the system (1).
When d = 0, the system is inconsistent. When d = 1, the system determines the value of at least one variable. When d = 2, the m.p.i.'s may still provide direct information which allows the elimination of variables (see "contraction property 2" in section 2 of this paper).

Row i with d(i) = d is called a preferred row (see [20]).
When d=n+1 (the default assignment), then the single rows of system (1) are not constraining in the integer sense and one should concentrate on either simple cost considerations or on methods which combine rows.

Another type of m.p.i. has been featured in the independent work of P.L. Hammer and associates (e.g. [8,15]) as well as by Balas and Jeroslow [1] and Guignard [9,10]. In that context, a "minimal" set E(i) is one from which no element may be dropped without invalidation of constraint (2). Which type of m.p.i.'s is meant will usually be clear from the context. If one wishes to be explicit, one may talk of minimal size p.i.'s, or minimal set p.i.'s, respectively.

In terms of the original variables y(j), all preferred inequalities (minimal or not) form a system

$$Q \cdot y \leq q \qquad (3)$$

with coefficients q(k,j) in $\{1,0,-1\}$, and

$$q(k) = -1 + \sum_K q(k,j), \quad K = \{j \mid q(k,j)=1\}$$

(directly implied by the definition of the $\tilde{y}(j)$).

The intent of this paper is to summarize several types of (m.p.i.)-systems (3) , with the emphasis on minimal size p.i.'s, and their computational use. The larger class of minimal set p.i.'s is probably less important in computation on account of its prohibitive size. For single-constraint problems, Hammer, Johnson, Peled in [16] establish connections between minimal set p.i.'s and the facetial structure of the integer polytope.

[Zero Origin Enumeration]

An early attempt at using (3) was concerned with implicit enumeration [20]. At a node in the enumeration tree let J be the set of "free" variables, temporarily considered at 0 and to be possibly set to 1 in "forward steps" of the algorithm. Let (3) be the updated constraint set. Rows indexed by

$$I = \{i \mid b(i) < 0\}$$

are "currently" infeasible and forward branches are to remove the infeasibilities.

Consider row i of (1), $i \in I$. Let $J(i)$ be the set of j such that $a(i,j) \neq 0$.

$$\sum_{P(i)} a(i,j).y(j) + \sum_{N(i)} a(i,j).y(j) \leq b(i) \qquad (4)$$

where $P(i) = \{j \in J \mid a(i,j) > 0\}$ and $N(i) = \{j \in J \mid a(i,j) < 0\}$.

In order to satisfy (4) over any of the successor nodes to the current node, the y(j) must satisfy

$$\sum_{N(i)} a(i,j).y(j) \leq b(i) \quad .$$

We describe an algorithm which constructs one minimal (size) p.i. and the corresponding minimal preferred set $E(i)$ for a given inequality of such a type, namely for:

$$\sum_{J(i)} \alpha(i,j) \cdot \tilde{y}(j) \leq \beta(i) \qquad (5)$$

$$\beta(i) < 0, \ \alpha(i,j) < 0 \text{ for all } j \in J(i) \ ,$$

in which we allow the possibility of $J(i)$ being empty.

This algorithm yields a minimal "local" (or "weak") condition (i.e. a condition over successor nodes only) , such as required for zero-origin enumeration, when one sets $J(i)$, $\alpha(i,j)$, $\beta(i)$ and $\tilde{y}(i)$ to $N(i)$, $a(i,j)$, $b(i)$ and $y(j)$, respectively. As will be shown later, it also deals with the more general case of minimal "global" (or "strong") conditions for the inequality (4), if $J(i)$, $\alpha(i,j)$, $\beta(i)$ and $\tilde{y}(i)$ are defined differently.

ALGORITHM 1:

(1) Initialize $e(i,j)$ to $\alpha(i,j)$, $h(i)$ to $\beta(i)$, $E(i)$ to $J(i)$, $d(i)$ to $|J(i)|$.

(2) If $E(i) = \emptyset$, terminate. Otherwise, consider:

$$\sum_{E(i)} e(i,j).\tilde{y}(j) \leq h(i) \qquad (6)$$

Find j^*: $e(i,j^*) = \max\{e(i,j) \mid j \in E(i)\}$.

(3) If $h(i) \geq e(i,j^*)$, terminate.

Otherwise, set $h(i)$ to $h(i) - e(i,j^*)$ and $e(i,j^*)$ to 0.

(In effect one adds to the current (6) the multiple hypercube constraint $-e(i,j^*).\tilde{y}(j^*) \leq -e(i,j^*)$)

Set $E(i)$ to $E(i) - \{j^*\}$ and $d(i)$ to $|E(i)|$.

Go to (2).

Remarks:

1) The conditions with inequality (5) ensure the existence of a preferred set. The degree $d(i)$ is at least equal to $|J(i)|$. The special case of empty $J(i)$, $d(i) = 0$, is interpreted as infeasibility.

2) The minimality of $E(i)$ is clear from construction. In fact, step (2) of the algorithm amounts to an ordering and elimination of the (negative) $\alpha(i,j)$ in decreasing order, so that step (3) eliminates a maximal number of $\alpha(i,j)$ for a given $\beta(i)$.

In the zero-origin enumeration procedure (for $1 < \text{dmin } d(i)$) one restricts forward branches to a minimal preferred set of the system (1), updated for the status of the enumeration search, with substantial economies over normal implicit enumeration.

Example: (MEC)

MEC does not yield a m.p.i. at the origin, as $\beta(i)=0$.

The constraint becomes effective after a branch on y1: $y1 \geq 1$.

There are then four degree 1 inequalities, namely $y2=y3=y4=y5 \geq 1$.

Generalized Search Origin (Salkin, Spielberg [11-15]):

Suppose an enumeration procedure starts at a point $y = \hat{y}$, instead of at the origin $y = (0,0,\ldots,0)$. Then it is clear that the redefinition

$$\tilde{y}(j) \leftarrow y(j) \quad \text{for } \hat{y}(j) = 0 \ ,$$
$$\tilde{y}(j) \leftarrow 1-y(j) \quad \text{for } \hat{y}(j) = 1 \ ,$$

followed by the procedures of ALGORITHM 1 will lead to sets P(I) and N(I), and corresponding m.p.i.'s for y, which depend on the choice of $\hat{y}$.

As shown in [12-15], a suitable choice of $\hat{y}$ can drastically improve the efficiency of an enumeration algorithm. It is not at all clear what is a most suitable choice. A rational approach which suggests itself is that of constructing preferred inequalities which are "strong" in the sense of having small degree, as is done in the following sections of this paper.

Example (MEC)

As the simplest such origin let $\hat{y} = (1,0,0,0,0)$.

$$-4\,\tilde{y}1 - \tilde{y}2 - \tilde{y}3 - \tilde{y}4 - \tilde{y}5 \leq -4$$

or $-4\,\bar{y}1 - y2 - y3 - y4 - y5 \leq -4$.

The degree is two, and the m.p.i.'s are (expressed logically):

Either y1 at 0, or y2 at 1
" , or y3 at 1
" , or y4 at 1
" , or y5 at 1 , i.e. the constraints of [7].

Note that all preferred branches lead away from the origin: y1 is at 1 and would be fixed at 0 in the branch, the others are at 0 and would be fixed to 1.

General ("Strong") Reduction: (see also [1,8,15]).

We shall now consider the full row index set $I = \{1,2,\ldots,m\}$ (permitting the case $b(i) \geq 0$), a given row $i \in I$ of form (4) (with the natural assumption that not both P(i) and N(i) are empty), and the task of generating from it one strongest (lowest degree) preferred inequality.

One seeks to combine the given inequality (4) with a positive linear combination of the hypercube constraints (i) $y(j) \leq 1$ and (ii) $-y(j) \leq 0$, so as to eliminate as many of the left hand terms as possible.

One needs only, if at all, use one of the two hypercube constraints for a given $y(j)$ in the linear combination, and the hypercube constraints actually used will be independent. As a consequence, the construction of the required positive linear combination is equivalent to considering the terms of (4) separately, in a sequence to be determined in an advantageous manner.

We shall call "reduced inequality" any inequality of the form (5), i.e. $\sum \alpha(i,j) \cdot \tilde{y}(j) \leq \beta(i)$, over all $j \in J(i)$, with the additional condition that $0 > \beta(i) \geq \alpha(i,j)$ for all j.

A reduced inequality in (0,1) variables $\tilde{y}(j)$ implies directly the preferred inequality (2) over the same set of $\tilde{y}(j)$. This implication is a special form of Gomory's derivation of all-integer cuts, [6].

The MEC can be put into the form (5) via the transformation
$\tilde{y}1 = 1 - y1 = \bar{y}1$: $-4\bar{y}1 - y2 - y3 - y4 - y5 \leq -4$.
One may then add to this inequality the hypercube constraints $y3 \leq 1$, $y4 \leq 1$ and $y5 \leq 1$ (for example) to obtain the reduced inequality : $-4\,\bar{y}1 - y2 \leq -1$, which in turn implies the minimal preferred inequality $\bar{y}1 + y2 \geq 1$.

In general, a preferred variable inequality is implied by an infinite family of equivalent reduced inequalities, and can be considered the simplest member or representative of that family. Simple examples: the logical disjunction "y(1) is one, or y(3) is zero" is represented by the m.p.i. $-y(1)-\bar{y}(3)\leq -1$, which in turn follows from any one of an infinite family of reduced inequalities such as $-3y(1)-4\bar{y}(2)\leq -2$, $-.5y(1)-.3\bar{y}(2)\leq -.3$,..., etc.. The deduction of a m.p.i. from a given inequality (4) proceeds via the generation of a reduced inequality.

The degree d(i) of an inequality (4) or (5) is established through the construction of one m.p.i. by means of algorithm 1, as described in the following theorems.

Theorem 1:

Let $\tilde{y}(j)=y(j)$ for $j\in N(i)$ and $\tilde{y}(j)=1-y(j)$ for $j\in P(i)$. Let Algorithm 1 be applied to the inequality (5), with $\alpha(i,j) = -|a(i,j)|$ and $\beta(i) = b(i) - \sum_{P(i)} a(i,j)$. Then, a m.p.i. will be obtained if $\beta(i)<0$, and it will have the smallest degree attainable from row i by a combination with a positive linear combination of the hypercube constraints.

The degree d(i) is related to an ordering of the coefficients. Let n(i) be the number of non-zero a(i,j). Let the non-zero a(i,j) be ordered such that $|a(i,j_1)|\leq|a(i,j_2)|\leq\ldots\leq|a(i,j_{n(i)})|$. We exclude the two cases: (i) inequality (5) is reduced, and (ii) the sum over all $\alpha(i,j)$ is larger than $\beta(i)$ (infeasibility).

Theorem 2:

The degree d(i) of (4) will be given by

$$d(i) = n(i) - k \quad ,$$

k being an index in the half-open interval $[1,n(i))$ determined by:

$$s(i,k) = \sum_{\ell} |a(i,j_\ell)| \ , \ \ell \text{ from 1 to } k,$$

such that $s(i,k)<|\beta(i)|$, $s(i,k+1)\geq\beta(i)$.

Proof:

As discussed before, we can consider the y(j) one at a time. If $a(i,j)<0$, the term $a(i,j).y(j)$ can be removed from (4) by adding $|a(i,j)|.y(j)\leq|a(i,j)|$ either to (4) directly or to a linear combination of hypercube constraints to be combined with (4) later. Positive terms $a(i,j)>0$ are at first dropped by addition of $-a(i,j).y(j)\leq 0$, i.e. without alteration of the right hand side.

If a sequence of L negative terms are now removed as above, then the resulting inequality implies a logical condition on the y(j) if and only if $b(i) + \sum_{\ell=1}^{L}|a(i,j_\ell)|$ remains less than 0, the j_ℓ referring to the removed terms with negative a(i,j). If the number of terms to be removed is to be maximal, it follows that the indices should be chosen so as to minimize the partial sums $\sum_{\ell=1}^{L}|a(i,j_\ell)|$ over the negative a(i,j)as L is increased, so that the negative a(i,j) must be ordered by magnitude.

(Example: $-3y1 - 2y2 - 4y3 \leq -6$ could be combined with $4.(y3\leq 1)$ to yield the reduced inequality $-3y1 - 2y2 \leq -2$, of degree 2. However, it is better to order the coefficients and to add $2.(y2\leq 1)$ and then $3.(y1\leq 1)$ instead, so as to obtain $-4y3 \leq -1$ of degree 1 .

Now, reconsider a positive a(i,j), say a(i,j*). It can be removed without affecting the right hand side (and with it the removal of the negative coefficients). However, one can also replace the term a(i,j*).y(j*) by -a(i,j*).$\bar{y}$(j*) and reduce the right hand side correspondingly by a(i,j*). If one then pursues the procedure outlined above, the new negative term -a(i,j*) participates in the ordering that determines the removal of the negative terms. If it does get removed by addition of a(i,j*).$\bar{y}$(j*) $\leq$ a(i,j*)., this is entirely equivalent to the removal of the original uncomplemented term a(i,j*).y(j*) by addition of -a(i,j*).y(j*) $\leq$ 0. But if it does not get removed , then the decrease of the right hand side via complementation has actually permitted the removal of some other term in its place, and may result in an increase in the number of terms which can be removed (but clearly not in a decrease).

Finally, if there are several positive terms a(i,j), then the process may be applied to all of them, resulting in a maximally reduced right hand side β(i). A removal of terms becomes possible only if β(i) is closest to coefficient, and is governed by the ordering of Theorem 2, i.e. by the partial sums s(i,k) over all coefficients.

Example: -y1 - y2 + 2y3 $\leq$ -1 is of degree 2 under weak reduction. I.e., one has the m.p.i. y1 + y2 $\geq$ 1 after deletion of the positive term. But after complementation, one has -y1 - y2 - 2$\bar{y}$3 $\leq$ -3, and strong reduction leads to $\bar{y}$3 $\geq$ 1 or y3 $\leq$ 0 (add y1 $\leq$ 1 and y2 $\leq$ 1 to obtain the reduced inequality -2$\bar{y}$3 $\leq$ -1 .

Algorithms for All Minimal Preferred Inequalities:

One may wish to generate all , or at least a subset of all , minimal preferred inequalities. We shall rely on simple enumeration of all possible index sets (see the examples). The algorithms are then characterized by the conditions which need be verified for a given set of indices.

A presorting of the coefficients as in Theorem 2 will permit the search to be curtailed when the partial sums attain a certain value. The computational effort involved will be discussed somewhat further in the concluding remarks.

In the following, the β(i) and κ(i,j) are defined as in Theorem 1.

Corollary: (for all minimal size m.p.i.'s , ALGORITHM 2)

Let the degree of system (1) be d. Let E $\subset$ J consist of d indices, and let $\bar{E}$ = J - E, Let i be such that d(i) = d, and let β(i) < 0.

Then, $\sum_{E} \bar{y}(j) \geq 1$ is a minimal size p.i. iff

$$\sum_{\bar{E}} \kappa(i,j) > \beta(i) \qquad (7)$$

Inequality (7) merely checks whether the removal of the y(j), j$\in \bar{E}$, leaves β(i) negative. By virtue of Theorem 1 and the fact that d(i)=d the result will then be a reduced inequality.

Proposition: (for all minimal set p.i.'s , [8,15], ALGORITHM 3)

For a given row i consider all index partitions E + $\bar{E}$ = J, in an order which arises from enumeration of index sets based on the presortings.

Then, E will define a minimal set p.i. iff (7) holds and if

$$\sum_{E \ni j} \alpha(i,j) \leq \beta(i) \quad \text{for all} \quad j \in E \ .$$

In the above, one essentially derives all reduced inequalities, regardless of degree. i.e. all logical implications not dominated by others. A major drawback of this approach is the excessively large number of such inequalities.

Remarks:

It is clear that a minimal set p.i. of degree d is a minimal size p.i., and conversely.

The enumeration of minimal size p.i.'s is usually manageable because

(i) the preordering permits curtailment of the search,

(ii) the a(i,j) matrix is often sparse, so that only few coefficients need be considered in a row,

(iii) the degrees d of interest are usually small, if not for the original system then for an augmented one (including generated auxiliary inequalities); one may also simply forego the use of m.p.i.'s if d exceeds a certain number, say 3 or 4,

(iv) there are easy termination tests for rows based on sums over the a(i,j) over the free variables in P(i) or P'(i) .

Further Examples:

a) Consider the one inequality (i=1, $\gamma(1) = \gamma$, $\beta(1) = \beta$).

$$-2y1 + 2y2 + 5y3 + y4 + y5 \leq -1 \ .$$

For zero-origin enumeration one obtains only the (minimal) preferred set $\{1\}$, i.e. one has $y1 \geq 1$.

After ordering of coefficients, inequality (5) is:

$$-\bar{y}4 - \bar{y}5 - 2y1 - 2\bar{y}2 - 5\bar{y}3 \leq -10 \ .$$

d is clearly 1. The minimal (size) p.i.'s are $\bar{y}3 \geq 1$, $\bar{y}2 \geq 1$, $y1 \geq 1$.

The sets $\bar{E}$ are $\{\bar{4},\bar{5},1,\bar{2}\}$, $\{\bar{4},\bar{5},1,\bar{3}\}$, $\{\bar{4},\bar{5},\bar{2},\bar{3}\}$, and for (set) m.p.i's also $\{1,\bar{2},\bar{3}\}$ (bars are used whenever $\tilde{y}(j) = \bar{y}(\cdot) = 1-y(j)$).

Correspondingly the minimal (set) p.i.'s are given by $E = J - \bar{E}$: $\{\bar{3}\}$, $\{\bar{2}\}$, $\{1\}$ and $\{\bar{4},\bar{5}\}$.

The matrix O of (3) is

$$\begin{pmatrix} -1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 1 \end{pmatrix} ,$$

and $q = (-1, 0, 0, 1)^T$.

b) Consider the same constraint, but weakened by an increase in the right hand side of 5 units. To illustrate how one obtains all m.p.i.'s, we list the coefficients in ascending order:

$$-5\bar{y}3 - 2\bar{y}2 - 2y1 - \bar{y}5 - \bar{y}4 \leq -1 \ .$$

The degree is two. One therefore considers all "bit patterns (or masks)" with two entries of one and computes coefficient sums over the complementary pattern, and checks inequality 7 (yes if satisfied, no otherwise):

1	1	0	0	0	yes: $-4 > -5$
1	0	1	0	0	yes: $-5 > -5$
1	0	0	1	0	no: $-5 = -5$
1	0	0	0	1	need not be tested
0	1	1	0	0	no: $-7 < -5$

all other patterns need not be tested.

The minimal preferred inequalities are:

$$\bar{y}3 + \bar{y}2 \geq 1, \quad \bar{y}3 + y1 \geq 1 \quad .$$

c) Consider the system:

$$\begin{pmatrix} -1 & 3 & -12 & 0 & -1 & 7 & -1 & 0 & 0 & 3 & -5 & -1 \\ 3 & -7 & 0 & 1 & 6 & 0 & 0 & 0 & -6 & 0 & 0 & 0 \\ 11 & 0 & 1 & -7 & 0 & -1 & 2 & 1 & -5 & 0 & 9 & 0 \\ 0 & 5 & 6 & 0 & 12 & -7 & 0 & 3 & 1 & -8 & 0 & 5 \\ -7 & -1 & -5 & 3 & 1 & -6 & 0 & -2 & 7 & 1 & 0 & -7 \\ -2 & 0 & 0 & -4 & 0 & 0 & -3 & -5 & -1 & 0 & 1 & 1 \end{pmatrix} \cdot \begin{pmatrix} v \\ w \\ x \\ y \\ z \end{pmatrix} \leq \begin{pmatrix} -6 \\ -1 \\ -4 \\ -8 \\ -7 \\ -4 \end{pmatrix}$$

When y1 is already determined to be zero, the degree is 2 and the matrix A becomes:

$$\begin{pmatrix} 0 & 0 & -1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & -1 & 0 & -0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -0 & 1 & 0 \\ 0 & 0 & -0 & 0 & 1 & 0 & 0 & 0 & 0 & 1 & -0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & -0 & 0 & -1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & -1 & 0 & 1 & 0 \\ 0 & 0 & -0 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & -1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & -1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 1 & -0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & -0 & 0 & 0 & 0 & 1 & -0 & 0 & 1 & 0 \\ 0 & 0 & 0 & -1 & 0 & 0 & 0 & 0 & -1 & 0 & 0 & 0 \end{pmatrix}$$

d) Consider the constraints for simple plant location in aggregated form:

$$\sum_j x(i,j) \leq n(i).y(i), \quad \sum_i x(i,j) \geq 1 \quad , \quad (\text{e.g., [12]}).$$

We are concerned only with the first constraint, in which n(i) is the number of routes (i,j) which are permitted for row i. It serves the purpose of prohibiting all shipments in row i when y(i) = 0, and allowing the n(i) permitted shipments when y(i) = 1. We complement all x(i,j) and obtain: $-\sum \bar{x}(i,j) - n(i).y(i) \leq -n(i)$. Without loss of generality one now focuses on an arbitrary column, say j1 (such that (i,j1) is permitted), and adds all n(i)-1 hypercube constraints $\bar{x}(i,j) \leq 1$ for the other columns.

This leads to $-1 + x(i,j_1) - n(i).y(i) \leq -1$, or the reduced inequality $x(i,j1) \leq n(i).y(i)$, or finally the m.p.i $x(i,j1) \leq^{\bullet} y(i)$. Hence, column j1 having been arbitrary, the aggregated system implies the disaggregated system of constraints $x(i,j) \leq y(i)$ for all permitted (i,j).

e) As a slight generalization (the $x(i,j)$ are no longer 0-1 variables), consider the aggregated form of the capacitated plant location problem:

$\sum_j x(i,j) \leq a(i).y(i)$, $\sum_i x(i,j) \geq b(j)$, $0 \leq x(i,j) \leq m(i,j)$, in which $m(i,j)$ is the minimum of $a(i)$, $b(j)$ and any possible prespecified bound on the shipment, say $\mu(i,j)$.

Complement $x(i,j)$ to $\bar{x}(i,j) = m(i,j) - x(i,j)$ and substitute in the first constraint. This produces

$-\sum \bar{x}(i,j) - a(i).y(i) \leq -\sum m(i,j)$, the sums being over those j in row which are permissible (have finite cost). Focusing again on an arbitrary column j1 and adding the constraints

$\bar{x}(i,j) \leq m(i,j).y(i)$ to the other (permitted) columns, one easily obtains the disaggregated form:

$x(i,j) \leq m(i,j).y(i)$, for all permitted routes (i,j).

In a manner which differs somewhat from the general approach of this paper, one can simplify d) and e) above by the device of adding selected inequalities $-x(i,j) \leq 0$ to the original inequalities (thus avoiding the complementations).

2. COMPUTATIONAL DEVICES AND SCHEMES.

In this section we outline various uses of minimal size p.i.'s. Let Q be the corresponding matrix of μ rows (indexed by k) and n columns (indexed by j).

All rows of Q have exactly d non-zero entries. If d=1, at least one variable can be fixed.

For the problems of this paper we have rarely found μ to exceed 20. For problems with up to 100 variables we limit μ to about 40 and again find that the limit is rarely attained (on an admittedly small sample of experiments).

The usefulness of the techniques to be discussed will depend on the data. It is certainly desirable that the degree of the system be relatively small (say, no more than 5 for moderate size problems, or (n/10) for larger systems). Also, the original constraint set should not be so special as to leave no room for meaningful reduction (e.g., it should not be of the set covering or multiple choice type); but notice the later remarks on the possible use of Benders inequalities in such situations.

The reduction procedures of section A can be programmed very easily and can, first of all, be used as simple "diagnostic devices". Examining a variety of test problems, one will find that they have a tendency to partition into two sets. One of them contains problems with large d and usually very simple structure. The other set is quite well behaved (has small d). It usually consists of problems of fairly high density with medium sized coefficients of alternating signs.

We have found as a matter of experimentation that Benders inequalities [2] frequently yield low values of d. Hence, adding Benders inequalities to the original constraint set can transform an unsuitable problem into a suitable one. Given a reasonable bound on the objective function, the degree is often found to be 2 over most of a branch and bound tree (for example for the 37 row, 74 variable problem of [20]).

In this connection it should be noted that the reduction procedures work essentially row-by-row. They do not, in themselves, capture the interaction of the constraints. But they automatically focus on the most constraining inequality (as is desirable) and they also, in a sense, capture the interrelation between original inequalities, the hypercube constraints, and the zero-one conditions.

For suitable problems, we discuss a) useful Q-matrix properties, b) simple BB schemes, c) other, more complicated, applications.

a). Q-matrix Properties.

(i) Contraction Properties.

When d>1, consider the effect of "branching" on a variable y(j) in an enumerative scheme. We say that variable y(j), or index j, or the branch itself is 1-contracting

(0-contracting), if it is clear that fixing y(j) at 1 (0) will lead to an updated matrix Q' with d' = d - 1.

CONTRACTION PROPERTY 1

y(j) is 0-contracting if $\exists i$: q(i,j) = -1

y(j) is 1-contracting if $\exists i$: q(i,j) = +1.

We say that y(j) is double-contracting if it is both 0- and 1-contracting. For the last example, y(2), y(3), y(6), y(10) and y(11) are double-contracting. Double-contraction implies, it should be noted, that the m.p.i.'s derive from at least two original inequalities and are therefore quite significant

(ii) The special case d=2.

Quadrilateral Patterns:

Quadrilateral non-zero patterns of Q permit the direct elimination of variables:

CONTRACTION PROPERTY 2

	y(i) y(j)	Inference
1)	$\begin{pmatrix} +1 & +1 \\ +1 & -1 \end{pmatrix}$	y(i) = 0
2)	$\begin{pmatrix} -1 & +1 \\ -1 & -1 \end{pmatrix}$	y(i) = 1
3)	$\begin{pmatrix} +1 & +1 \\ -1 & -1 \end{pmatrix}$	y(i) = 1 - y(j)
4)	$\begin{pmatrix} +1 & -1 \\ -1 & +1 \end{pmatrix}$	y(i) = y(j)

In this fashion, one may fix nine of the twelve variables for the second numerical example, simply by reduction and the use of Q for d = 1 and d = 2.

k - cycles (for d=2)

More complicated patterns of Q are also of interest. By a cycle we mean a pattern of k rows and columns of the form:

$$\begin{pmatrix} x & & & x \\ x & x & & \\ & x & x & \\ & & \ddots & \\ & & & x \end{pmatrix}$$, where x stands for +1 or -1 .

We consider three special cases:

First case:

all $x = +1$ implies $y(1) + y(2) + \ldots + y(k) \leq \lfloor k/2 \rfloor$

all $x = -1$ implies $y(1) + y(2) + \ldots + y(k) \geq \lceil k/2 \rceil$.

These are [illegible] useful constraints. While not of the m.p.i. type, they derive [illegible] m.p.i.'s by addition and division by 2. The derivation may be [illegible] if it comes from from only one constraint.

Example: $5y1 + 4y2 + 3y3 \leq 6$ implies $y1 + y2 + y3 \leq 1$.

Reduction gives $y1+y2 \leq 1$, $y1+y3 \leq 1$, $y2+y3 \leq 1$, and addition of these constraints leads to $2.(y1+y2+y3) \leq 3$ and to the conclusion.

The derivation is more valuable if the implication follows from several original inequalities only.

Example:

① $5y1 + 7y2 + 3y3 \leq 8$

② $6y1 + 3y2 + 4y3 \leq 7$ together imply $y1 + y2 + y3 \leq 1$.

Neither of these inequalities is sufficient alone, nor does their sum help.

But ① implies $y1+y2 \leq 1$ and $y2+y3 \leq 1$, and ② implies $y1+y3 \leq 1$.

Hence, as above , one has the conclusion via generation of the m.p.i.'s and detection of the 3-cycle.

Second case: k-cycles with alternating signs, k>2

One frequently encounters patterns such as $\begin{pmatrix} +1 & -1 & \\ -1 & +1 & -1 \\ & -1 & +1 \end{pmatrix}$.

For k=2, this was case 4) of contraction property 2. The same implication is valid for all k>2.

From the nature of Q it follows that the right hand side q is the zero vector. Hence one has:

$y(k) \geq y(1) \geq y(2) \geq \ldots \geq y(k-1) \geq y(k)$

so that all $y(j)$ are equal, $j = 1,2,\ldots,k$. Note again that these m.p.i.'s can not have arisen from a single original constraints (otherwise all entries in a column would have to be of equal sign), so that these implications are by no means trivial.

Third case: k-cycle with alternating sign except for one column.

We give just one case, such as in the pattern $\begin{pmatrix} +1 & & +1 \\ -1 & +1 & \\ & -1 & \ddots \\ & & \ddots & +1 \end{pmatrix}$ with the last column not alternating. The right hand side vector is (1,0,0,...,0).

Adding all constraints other than the first one obtains: $y(1) + y(k) \leq 1$

and $-y(1) + y(k) \leq 0$, i.e. $y(k)=0$ by case 1 of the contraction property 2. There are clearly other patterns which will reduce to other cases discussed under contraction property 2.

While such special cases are of interest in principle, it is not certain that one will wish to exploit them computationally. The reason for this doubt is that the technique of probing, discussed later, will automatically lead to the effective exploitation of such structures of degree 2.

b) Enumerative Schemes.

We use "enumerative" in the broad sense, to cover both (strictly) enumerative methods and branch-and-bound schemes.
All such methods suffer from a lack of efficient devices for choosing branch variables. Current procedures rely usually on some form of "penalties" p(j), which are underestimates of the "cost" (increase in objective function for a minimization problem) for the lowering of the upper bound or raising of the lower bound of y(j). We assume that a table of such penalties is computable.
Minimal preferred variable inequalities, we claim, furnish a second important device to be combined with the first from three points of view:

(I) Decrease of Combinatorial Complexity

Consider two simple tree searches, one without reduction (scheme A), one with reduction (scheme B). In either case, let the search proceed from an initial level 0 by successive explicit branches to levels 1,2,...,L.
Search A will have to consider

f(0)xf(1)x...xf(L) points, where

f(0)>f(1)>f(2)>...>f(L) are the numbers of free variables at levels 0,1,..,L.

In contrast, search B will consider at most

d(0)xd(1)x...xd(L) points,

the product of the degrees d(k), k=1,2,...,L .
It is clear that d(k)<f(k) for all levels, and that usually

d(k)<<f(k), especially in the absence of other effective curtailing devices for search A , when the first product will be (f)!/(f-L)! (a very large number, indeed).
The degrees, on the other hand, are usually small, d(k) being often ≤ 2 at all levels, so that the second product becomes quite manageable. It is true that any reasonable scheme will contain some search curtailing devices, so that scheme A above can not be assumed to be as bad as outlined. Yet, in general one can expect (and our computational experience tends to confirm this) a substantial improvement from scheme B, not only on account of the simple analysis above, but also because there is evidence to the effect that the required maximum level of the search L can often be decreased by reduction (e.g., as in (II)) .

(II) Contraction

Contraction properties can be viewed as tools for rational direction of the search. E.g., choice of a double-contracting variable for branching ensures a reduction of d at the next level of the search. If there are no double-contracting variables, one may concentrate on branches which yield high penalties in one direction and contraction in the other.

(III) Propagation.

The usual branch-and-bound codes for (0,1) programming use dichotomies of the type: either y(j) is one or y(j) is zero, for some suitable basic y(j). In general both "suggested"

subproblems require the solution of a linear program.

It is clear that row k of Q represents another type of dichotomy which may be more interesting. If J(k) is the set of those indices for which $q(k,j) \neq 0$, then one must be concerned with the d preferred possibilities: $y(j) = 1$ for $j \in J(k)$ and $q(k,j) = -1$, and $y(j) = 0$ for $j \in J(k)$ and $q(k,j) = 1$. Of special interest are preferred branches which require no work. I.e., during the enumeration one usually has a "current tableau" in which the variables take on current values. If the current value of a variable is the same as the preferred value (for some row k), then one can clearly take care of that possibility by merely fixing the bounds and storing the other possibilities in a node table (as is usual in branch-and-bound programming). We call this procedure propagation and the respective variable y(j) a propagation variable.

Propagation is of particular interest when the variables can be chosen such that the penalty of the "alternate" (stored) node is large, so that it may never need to be explored by the BB method. Minimal preferred variable inequalities can be used to make such choices (often so as to generate, at almost no cost, a sequence of alternate nodes with large penalties. Below we shall give special cases (frequently occurring) which are particular favorable for such a procedure: see also [13].

(iv) Additivity and Persistence of large Penalties.

In some of our work we have concentrated on exploiting large reduced costs and m.p.i.'s via propagation. Instead of the general case, we shall here discuss important special cases which are quite common. Let d=2, and let there be a m.p.i. with a propagating variable, say y(j): $\tilde{y}(j)+\tilde{y}(h) \geq 1$ (in row k of Q, say), i.e. let $\tilde{y}(j)=1$ correspond to the value which y(j) has in the current optimal tableau of the relaxed problem.

Then the alternate node has $\tilde{y}(j)=0$ and therefore also $\tilde{y}(h)=1$. Let π^k be the overall penalty of the alternate node. It can frequently be shown to be the sum of the penalties for $\tilde{y}(j)=0$ and $\tilde{y}(h)=1$.

Additivity: $\pi^k = \pi(\tilde{y}(j)=0) + \pi(\tilde{y}(h)=1)$.

We call a row k of Q with this property, and of degree 2 (additivity has meaning for d>2, too), a perfect row. Perfect rows arise commonly from two nonbasic variables y(j) and y(h), linked by the minimal p.i.: $y(j)-y(h) \leq 0$; i.e., $\tilde{y}(j)=1-y(j)$ and $\tilde{y}(h)=y(h)$.

Of course, the situation is advantageous when π^k is large, but especially so when the reduced cost of y(h) is appreciably larger than that of y(j), so that the large penalty "persists".

Persistence of (large) penalties:

When $\pi(y(h)=1) >> \pi(y(j)=0)$, then propagation not only results in an alternate node with large penalty, but variable y(h) remains a free variable (y(j) gets fixed), and the large penalty persist (remains availabe for the possible generation of other favorable alternate nodes via the m.p.i.'s, altered or not).

Example:

For our (6,12) example, the penalties are:

The two m.p.i.'s k=2 and k=6 are perfect:

j	$\pi(y(j)=0)$	$\pi(y(j)=1)$
1	0	0
2	0	1.8166
3	2.1181	0.1644
4	7.1318	1.1317
5	0	9.7343
6	1.061	3.4868
7	0	6.7364
8	1.1317	1.3707
9	0	7.1268
10	1.2076	1.3242
11	0	4.6178
12	0	1.2181

k=2: $y(5) - y(2) \leq 0$

k=6: $y(11) - y(9) \leq 0$.

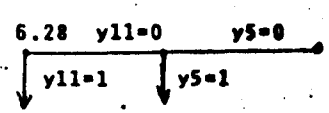
All variables are nonbasic, and y(5) and y(11) can be chosen as propagating variables (at 0).

k=2: $\pi^2 = \pi(y5=1) + \pi(y2=1) = 1.8166 + 9.7343 = 11.55$

k=6 $\pi^6 = \pi(y11=1) + \pi(y9=1) = 4.6178 + 7.1268 = 11.7446.$

The second case is better, for the large penalty is not associated with the propagating variable but with the variable linked to it, so that the larger penalty (of 7.1268) can be used further on in the tree.

These penalties can be added to the objective function value of the relaxed problem ($z_0 = 6.28$ for the example), so that one may start as follows:



, et cetera.

$z \geq 17.74$ $z \geq 17.85$.

As one can readily find integer solutions with z = 15 or 13 (the optimum), the alternate nodes will never have to be processed, so that variables y11 and y5 have effectively been fixed at 0 by the propagation procedure.

c) Some Numerical Results with Reduction and Contraction.

(i) Branch and Bound Experiments

Table 1 summarizes experiments with a standard branch-and-bound (BB) scheme coded in APL. The table entries give the number of linear programs required for resolution.

Column 2: Normal BB method

Column 3: BB, d = 1 used for fixing variables

Column 4: Same as above. Branches selected from Q by descending order of column counts for non-zero entries

Column 5: Branches selected by a set of priorities:

(I) First choose double contracting variables

(II) After exhaustion of (I), choose the branch variable in terms of maximal column count of Q

(III) Use standard penalties to break ties.

For the first three problems the results are fairly good. The (28,35) problem, e.g., was found to require about 47 linear programs with a commercial code. The (12,44) problem has somewhat poor structure after the initial fixing of some variables. As seen from Table 2, this problem is solved readily when Gomory-Johnson cuts (see [6,9]) are used.

TABLE 1: BRANCH AND BOUND EXPERIMENT

(NUMBER OF LP's)

PROBLEMS (m,n)	STANDARD BB	BB FIXING VARIABLES FROM d=1	BB CHOOSE BRANCH VAR. FROM Q	BB MINIMAL INEQU. STRATEGIES
(6,12)	15	11	8	8
(10,20)	41	41	29	23
(28,35)	57	49	30	20
(12,44)	45	45	200	---

(II) State Enumeration with Gomory-Johnson Cuts [10,17]

Good results are attainable from combining a general enumeration approach, in which linear programs are resolved over problems for which the "free" zero-one variables are fixed at carefully chosen tentative values, with the use of cutting planes due to Gomory and Johnson. Table 2 presents results of experiments with and without reduction in the sense of this paper.

TABLE 2, STATE ENUMERATION WITH GOMORY-JOHNSON CUTS

(NUMBER OF LP's)

PROBLEMS	STANDARD BB		STATE ENUMERATION WITHOUT / WITH REDUCTION	
(6,12)	15	8	5	2
(10,20)	41	17	5	5
(28,35)	57	12	10	10
(12,44)	45	72	5	7
(10,28)	--	385	33	21
(15,30)	15	--	22	4
(31,31)	567	13	7	6

d) Integer Programming.

The logical reduction concept can be generalized to integer programming. A minimal preferred inequality then becomes a logical condition of the form: either (raise a lower bound of variable j) or (lower an upper bound of variable h) or ... . When the ranges of the integer variables are large, such conditions may be too weak to merit exploitation. For small-interval integer programming, of course, the conditions may play roles similar to the ones described in this paper.

e) Probing, an effective tool for zero-one programming.

Most promising appears to be the recent concept of "probing", (see [14]). Given an m.p.i system $Q.y \leq q$, one selects a number of columns of Q (say 3 or 4) with maximal number of non-zero entries (possible giving additional weight to double contraction). For each of the selected columns (or free variables) one pursues both values $y(j)=0$ (1) through the system in turn. That is, one can usually fix other variables as a consequence (conditional upon the initial selection and value), and pursue such "cancellations" further (both through examination of Q and also by a final substitution in the initial problem itself).

One may eventually arrive at an infeasible situtation and be able to fix the selected variable. If not, one chooses the selected column and value which gave rise to a maximal number of conditional cancellations for the next branch in the enumeration (or BB scheme). In this fashion, PROBING serves to select a branch which permits a maximal number of variables to be fixed as a consequence.

3. Summary; Recent Work.

Summary.

In the introduction we have summarized various recent and possible uses for logical reduction and logical inequalities (in items a) to f)), in this section we shall elaborate on these points somewhat, drawing in part on the recent papers [14,18,21,25].

There exists enough evidence to suggest that all of the above items are important in some context and for some problems. A large scale system for (0,1) and mixed (0,1) programming should therefore provide the corresponding capabilities, to be invoked by the user as the situation may suggest. Thus the system should contain programs for:

1) Fast logical tests of degree 0 and 1 (infeasibility and fixing of variables); such tests have been incorporated in practically all known enumeration codes, but they have not been exploited in Branch-and-Bound codes.
2) Fast construction of low degree (strong) inequalities, especially those of degree 2 for probing. Such procedures have apparently only been implemented in the cited references, except possibly in a rudimentary fashion as "look-ahead" tests in some enumerative codes.
3) Construction of pure (0,1) inequalities during execution of the algorithmic procedure, usually Benders inequalities. However, sometimes other simple inequalities suggest themselves readily and from the start, such as "side conditions" (as e.g. in [24], also [12]); while these may appear trivial on inspection, they are often quite important in union with the other constraints.
   This item is absolutely necessary if logical reduction is to be employed at all for mixed (0,1) problems. It is essential for pure problems from a practical point of view when the given constraints are not themselves of low degree (Benders inequalities tend to be of low degree if good upper bounds on z are known or can be conjectured).
4) Procedures for "improving" a given set of pure integer constraints. There exist a number of possibilities. Reference [3] pursues one approach. Another approach is that of trying to generate "strong" Benders inequalities from the start. One may exploit the (usually) great multiplicity of dual solutions to the relaxed linear program to choose dual variables which yield strong inequalities (or coefficients: see [24] where an attempt is made to obtain maximally effective "gain-functions" which are, de facto, improved coefficients of Benders inequalities).
5) Addition of all (or of selected) logical inequalities to the constraint matrix for the purpose of strengthening the auxiliary linear programs (of either the Branch-and-Bound or Enumeration method employed). One may wish to add only a subset of the generated logical inequalities, for example those which violate some conditions (such as a "currently" available LP constraint set; one sees here a connection to the well known cutting plane methods of Gomory and others).

6) Computation of penalties (or at least of reduced costs, i.e. of penalties for the nonbasic variables).
Computation of strengthened "conditional" penalties, guided by the logical constraints. In addition to utilizing this information for the possible fixing (or "almost" fixing: see propagation) of variables, one may take the improved penalties as a criterion for branching.

It is clear that the list is not exhaustive, and that there is much room for variation. But it suggests some important directions in which production codes should be enhanced.

Recent large scale work.

We first describe some features of the (purely experimental) code "ENUM" of reference [25]. It is designed so as to implement points 1) to 3) in a highly efficient manner, and addresses point 6) to some extent. The main features of ENUM are:

- use of a fast and robust linear programming system (in this instance, IBM's MPSX/370, with interfaces provided by its "extended control language").
- data-structures and related procedures which assure the efficient implementation of logical tests (especially of probing) featured in this paper.

The adopted data structure consists of a row-wise storage of the nonzero coefficients in decreasing order of their absolute values. In addition to the associated row and column indices , one requires a column-link and n column pointers.

Logical tests can then be limited to those rows which have recently been updated, and within a row they are not only limited to the nonzeros but can also be stopped as soon as a certain condition arises. Thus, there are no redundant operations. Moreover, the structure permits the ready addition of derived integer inequalities, such as Benders inequalities or logical constraints.

In a first phase, strong heuristics are employed , based on the current linear programming solution, its reduced costs and an ordering of the variables which has some similarity to that of propagation. For

most large problems, solutions are found readily.

In a second phase, logical tests are used exhaustively, whenever there is any change in the status of the enumeration. Experiments show that, for large problems, improvements of an order of magnitude are possible over efficient branch and bound production programs.

We now turn briefly to the work of E.L. JOhnson and colleagues (Crowder, Padberg and Suhl), mentioned in the introduction. For a class of difficult large problems, they concentrate on conditioning the given problem so as to make its final version tractable for the branch and bound scheme.

They generate constraints from

. disaggregation
. logical reduction (before and after resolutions of linear programs)
. suitabletruction of solewhat stronger inequalities from given logical inequalities and from the original constraints,

and add such constraints (selectively) to the relaxed problems. In this fashion they appear to be able to shrink gaps (between the objective function of the integer program and of the relaxed problem) from large values (such as 30-40%) to insignificant amounts (say, less that 5%).

Evidently, such work can have a major impact, in future, on the much needed task of making mathematical programming systems capable of handling very large mixed integer problems, at least for a significant subset of such problems.

4. References.

1 Balas, E., and R. Jeroslow, Canonical Cuts on the Unit Hypercube, SIAM J. Appl. Math. 23 (1972) 61-69.

2 Benders, J.F., Partitioning Procedures for Solving Mixed-Variables Programming Problems, Numerische Mathematik 4 (1962) 238-252.

3 Bradley, G.H., Hammer, P.L., and L.A. Wolsey, Coefficient Reduction for Inequalities in 0-1 Variables, Math. Progr. 7 (1975) 263-282.

4 Breu, R., and C.A. Burdet, Branch and Bound Experiments in 0-1 Programming, Math. Progr. Study 2 (1974) 1-50.

5 Crowder, H., and M.W. Padberg, Solving Large Scale Symmetric Travelling Salesman Problems to Optimality, IBM Research Report RC 7743, June 1979.

6 Gomory, R.E., All-Integer Integer Programming, in Graves, P.L., and Ph. Wolfe, "Recent Advances in Mathematical Programming", Mc Graw Hill, 1963.

7 Gomory, R.E., and E.L. Johnson, Some Continuous Functions Related to Corner Polyhedra, IBM Res. Rep. RC3311, Feb. 1971, (see also: Mathematical Programming 3 (1972) 23-85 and 359-389).

8 Granot, F., and P.L. Hammer, On the Use of Boolean Functions in 0-1 Programming, Methods of Oper. Res. 12 (1972) 154-184.

9 Guignard, M., Methodes Heuristiques de Resolution d'un Systeme d'Inegalites Lineaires en Variables Entieres ou Bivalentes, Publications du Lab. de Calcul, Rapp. 32, Univ. de France, Lille, 1972.

10 Guignard, M., and K. Spielberg, Zero-One Zero-One Programming and

Enumeration, Proceedings, Symp. on Nonlinear Progr., 1974, Univ. of Wisconsin Press (1975) 333-360.

11 Guignard, M., and K. Spielberg, Reduction Methods for State Enumeration Integer Programming, Annals of Discrete Mathematics 1(1977) 273-285.

12 Guignard, M., and K. Spielberg, Alorithms for Exploiting the Structure of the Simple Plant Location Problem, Annals of Discrete Math. 1 (1977) 247-271.

13 Guignard, M., and K. Spielberg, Propagation, Penalty Improvement and Use of Logical Inequalities, Proc. of First Symp. on OR, Heidelberg '76, in: R. Henn et al, Methods of Operations Research, Vol. XXV, Verlag A. Hain, (1977) 157-171.

14 Guignard, M., Spielberg, K., and U. Suhl, Survey of Enumeration Methods for Integer Programming, Proceedings, SHARE 51, Boston, ACM (1978) 2161-2170.

15 Hammer, P.L., Boolean Procedures for Bivalent Programming, in Hammer P.L., Zoutendijk G., "Mathematical Programming in Theory and Practice", North-Holland (1975) 311-363.

16 Hammer, P.L., Johnson, E.L., and U.N. Peled, Facets of Regular 0-1 Polytopes, Math. Programming 8 (1975) 179-206.

17 Johnson, E.L., and K. Spielberg, Inequalities in Branch and Bound Programming, Proceedings, NATO Meeting, Copenhagen, 1971; in: R.W. Cottle, and J. Krarup, "Optimisation Methods for Resource Allocation", The English Univ. Press LTD, London (1974) 371-399.

18 Johnson, E.L., and U. Suhl, Experiments in Integer Programming, to

appear in Applied Discrete Mathematics 1, North Holland Publ. Co. (1980).

19 Krarup, J., and P.M. Pruzan, Selected Families of Discrete Location Problems, Part III : The Plant Location Family, Annals of Discrete Mathematics 5 (1979).

20 Lemke, C.E., and K. Spielberg, Direct Search Zero-One and Mixed Integer Programming, Operations Research 15 (1967) 892-914.

21 Mairs, T.G., Wakefield, G.W., Johnson, E.L. and K. Spielberg, On a Production Allocation and Distribution Problem, Management Science, 24 (1978) 1622-1630.

22 Salkin, H.M., On the Merit of the Generalized Origin and Restarts in Implicit Enumeration, Operations Research 18 (1970) 549-555.

23 Salkin, H.M., Integer Programming, Addison Wesley (1975).

24 Spielberg, K., and H.M. Salkin, An Adaptive Algorithm for Binary Programming, IBM N.Y. Sc. C. Rep. 3203-24 (1968).

25 Spielberg, K., Plant Location with Generalized Search Origin, Management Science 16 (1969) 165-178.

26 Spielberg, K., and U. Suhl, An Experimental Enumeration Software System for Large Scale 0-1 Problems with Efficient Data Structures and Access to MPSX/370, to appear as RC Report, IBM Research, Yorktown Heights, N.Y. (1979).

027 Williams, H.P., Experiments in the Formulation of Integer Programmin Problems, Math. Progr. Study 2 (1974) 180-197.

Fractional vertices, cuts and facets of the simple plant location problem

Monique Guignard

The Wharton School
Department of Statistics
University of Pennsylvania

Technical Report No.30

May 1978

Revised December 1978

Abstract

This paper investigates the structure of the Integer Programming Polytope of an uncapacitated (simple) plant location problem. One can describe families of fractional vertices and derive from them valid inequalities for the integer problem. Some of these will actually be shown to be facets of the integer polytope. Also some families of SPLP with large duality gaps will be described, together with facets which bridge these gaps. Much of the motivation stems from algorithmic work in which the exploitation of "good" cutting planes within a direct dual algorithm have been shown to be of crucial importance.

0. INTRODUCTION

This paper investigates the structure of the LP polytope of an uncapacitated location problem:

$$\text{SPLP}(m,n): \quad \min_{x,y} \sum_{i\in I}\sum_{j\in J} c_{ij}x_{ij} + \sum_{i\in I} f_i y_i$$

$$\forall j\in J: \quad \sum_{i\in I} x_{ij} = 1$$

$$\forall i\in I, \forall j\in J: \quad x_{ij} - y_i \le 0$$

$$x_{ij} \ge 0$$

$$y_i \ge 0, \text{ integer},$$

where I is a set of m possible plant locations, J a set of n destination points, x_{ij} the ratio of the requirement of $j\in J$ satisfied from location $i\in I$, and y_i, a 0-1 variable, is 1 if and only if location $i\in I$ is open. c_{ij} is a normalized cost of shipment from i to j, and f_i is a fixed charge associated with opening plant (or location) $i\in I$.

We assume $n \ge m \ge 3$.

We will first construct families of fractional vertices of the LP polytope, and generate from them valid cuts. We will then prove that some of these cuts are facets.

In a second part, we will study families of SPLP with large duality gaps, and show that for these one specific facet bridges the gap.

Fractional vertices have been characterized in Theorem 8 of [2], where certain cuts are proposed without proof. In this paper we construct similar inequalities and derive from them cuts which are shown to be facets of the integer polyhedron.

We have shown elsewhere, [4], how such cuts can be (and indeed must be) exploited via their associated dual variables within a direct dual algorithm (for the more difficult capacitated plant location problem and for pure integer cuts) and expect that a similar approach may be even more effective when suitable facets are constructed and utilized during the calculations.

I. SOME FRACTIONAL VERTICES OF SPLP(m,n).

Define $\bar{y}_i = 1 - y_i$ and $s_{ij} = 1 - x_{ij} - \bar{y}_i$, $\forall i \in I$, $\forall j \in J$.

SPLP(m,n) can be rewritten as a set partitioning problem:

$$m + \min_{x,\bar{y}} \sum_{i\in I} \sum_{j\in J} c_{ij}x_{ij} - \sum_{i\in I} f_i\bar{y}_i$$

$$\forall j \in J: \quad \sum_{i\in I} x_{ij} = 1 \qquad (\Sigma_j)$$

$$\forall i \in I, \forall j \in J: \quad x_{ij} + \bar{y}_i + s_{ij} = 1 \qquad (*_{ij})$$

$$x_{ij}\,, \quad \bar{y}_i,\ s_{ij} \geq 0$$

$$\bar{y}_i \text{ integer .}$$

LP(m,n) will denote the LP relaxation of SPLP(m,n).

let σ be a surjection (i.e. an onto function) from J into $I_p \triangleq I\backslash\bar{I}_p$ where $|\bar{I}_p| = p \geq 0$, $m > p + 2$, and let $M_i = \{j \in J | \sigma(j) = i\}$, $i \in I_p$, $\bar{M}_i = J\backslash M_i$. Let $J_q \subseteq J$ be such that the restriction of σ to J_q , $\sigma/_{J_q}$, is still a surjection into I_p. Let $\bar{J}_q = J\backslash J_q$, where $q = |\bar{J}_q| \geq 0$.

We will need the following results in the generation of the valid inequalities.

Lemma I.1

$$\forall i \neq i' \,,\ i,i' \in I_p,\ M_i \cap J_q \subsetneq \bar{M}_{i'} \cap J_q$$

Proof: Let $i, i' \in I_p$.

(1) $\forall i' \neq i$, $M_i \cap J_q \neq \bar{M}_{i'} \cap J_q$:

since σ is a function, ($\sigma(i)$ is unique for a given i), $\forall i'' \neq i, i'$, (i'' exists as $|I_p| > 2$), $i'' \in I_p$,

$$M_{i''} \cap J_q \subset J_q \backslash (M_i \cup M_{i'})$$

If $M_i \cap J_q = \bar{M}_{i'} \cap J_q$, then $M_{i''} \cap J_q = \emptyset$ which contradicts the fact that $\sigma/_{J_q}$ is a surjection.

(2) $\forall i \neq i'$, $M_i \cap J_q \subset \bar{M}_{i'} \cap J_q$

$$\forall j \in M_i \cap J_q,\ \forall i' \neq i,\ \sigma(j) \neq i' \Longrightarrow j \in \bar{M}_{i'} \cap J_q$$

Corollary I.2

Let n_j be a positive integer, $\forall j \in J_q$. Then

$$\forall i \in I_p,\ \sum_{j\in M_i \cap J_q} n_j < \sum_{j\in \bar{M}_{i'} \cap J_q} n_j .$$

Definition

Let $K \subset J_q$ with $|K| = m-p$ and $\sigma(K) = I_p$. Let $\bar{K} = J\backslash K$.

Let τ be a point-to-set mapping from $\bar{K}$ into I_p

such that $\begin{cases} \bar{\tau}(j) \cap \sigma(j) = \emptyset \quad \forall j \in \bar{K} \\ \text{and} \quad |\tau(j)| = m - p - 2. \end{cases}$

Let $X = (x, \bar{y}, s)$. Then $X(\sigma, K, \tau, p)$ will be defined as follows:

$\forall i \in \bar{I}_p$, $\forall j \in J$, $x_{ij} = 0$, and $\exists j = j(i)$: $s_{i,j(i)} = 0$.

$\forall i \in I_p$, $\begin{cases} \forall j \in M_i, \ x_{ij} = 0 . \\ \forall j \in \bar{M}_i \cap K, \ s_{ij} = 0 . \\ \forall j \in \bar{K}, \ s_{\tau(j),j} = 0 . \end{cases}$

Proposition I.3

$X(\sigma, K, \tau, p)$ is a vertex of $LP(m,n)$.

Proof:

One can easily compute the remaining components of $X(\sigma,K,\tau,p)$:

$$x_{ij} = \frac{1}{m-p-1} \qquad \forall i \in I_p, j \in \bar{M}_i$$

and $\quad \bar{y}_i = \frac{m-p-2}{m-p-1}$, or $y_i = \frac{1}{m-p-1} \quad \forall i \in I_p$

$$\bar{y}_i = 1 \text{ , or } y_i = 0 \quad \forall i \in \bar{I}_p.$$

By theorem 8 of [2], the non-integer solution $X(\sigma,K,\tau,p)$ is a vertex of $LP(m,n)$.

□

Example

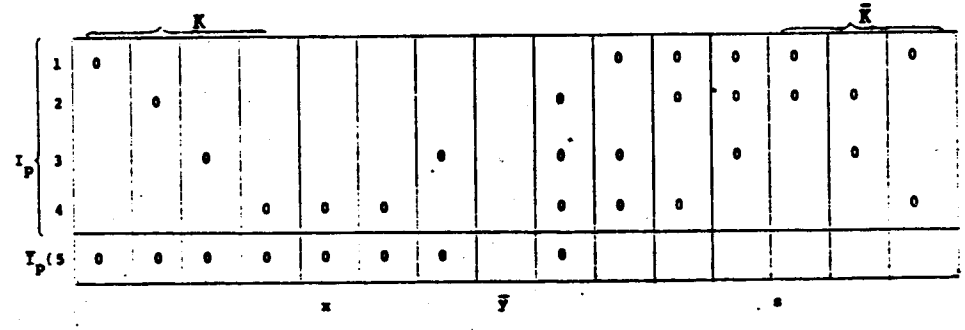
Let $m = 5$, $n = 7$, $p = 1$. $I_p = \{1,2,3,4\}$, $J = \{1,2,3,4,5,6,7\}$,

$\tau(5) = \{1,2\}$, $\tau(6) = \{2,3\}$, $\tau(7) = \{1,4\}$.

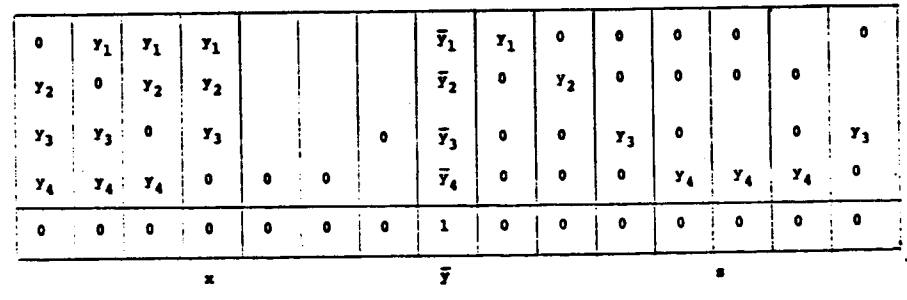
$M_1 = \{1\}$, $M_2 = \{2\}$, $M_3 = \{3,7\}$, $M_4 = \{4,5,6\}$.

$K = \{1,2,3,4\}$, $|K| = 4$, $\sigma(K) = I_p = \{1,2,3,4,\}$, $j(5) = 1$.

The known entries of $X(\sigma, K, \tau, p)$ are



We must determine the other entries.



$$\left.\begin{array}{l} y_1 + \quad y_3 + y_4 = 1 \\ y_1 + y_2 \quad + y_4 = 1 \\ y_1 + y_2 + y_3 \quad = 1 \\ \quad y_2 + y_3 + y_4 = 1 \end{array}\right\} \Longrightarrow y_1 = y_2 = y_3 = y_4 = \frac{1}{3} \ .$$

0	1/3	1/3	1/3				2/3	1/3	0	0	0	0		0
1/3	0	1/3	1/3				2/3	0	1/3	0	0	0	0	
1/3	1/3	0	1/3			0	2/3	0	0	1/3	0		0	1/3
1/3	1/3	1/3	0	0	0		2/3	0	0	0	1/3	1/3	1/3	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
			x				$\bar{y}$				s			

$$\forall j \notin K,\ x_{\tau(j),j} = 1 - \bar{y}_{\tau(j)} = y_{\tau(j)} = \frac{1}{3} \ .$$

$$\forall i \neq \tau(j),\ x_{ij} = 1 - \frac{2}{3} = \frac{1}{3} \ , \ s_{ij} = 1 - \frac{1}{3} - \frac{2}{3} = 0 \ .$$

0	1/3	1/3	1/3	1/3	1/3	1/3	2/3	1/3	0	0	0	0	0	0
1/3	0	1/3	1/3	1/3	1/3	1/3	2/3	0	1/3	0	0	0	0	0
1/3	1/3	0	1/3	1/3	1/3	0	2/3	0	0	1/3	0	0	0	1/3
1/3	1/3	1/3	0	0	0	1/3	2/3	0	0	0	1/3	1/3	1/3	0
0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
			x				$\bar{y}$				s			

II. VALID INEQUALITIES ASSOCIATED WITH $X(\sigma, K, \tau, P)$

Consider the inequalities

$$x_{ij} + \bar{y}_i \leq 1 \qquad \forall i \in I_p \ , \ \forall j \in \bar{M}_i \cap J_q \qquad (*_{ij})$$

$$\sum_{\substack{i \in I_p \\ i \neq \sigma(j)}} x_{ij} \leq 1 \qquad \forall j \in J_q \qquad (\Sigma_j)$$

satisfied as equalities by $X(\sigma, K, \tau, P)$.

Any nonnegative integer combination of these inequalities will be a valid inequality for SPLP(m,n).

If the resulting left-hand side coefficients are all congruent to 0 modulo an integer and if the right-hand side is not, one can divide all coefficients by that integer and round down the right-hand side. The result will still be a valid inequality.

Multiply every $(*_{ij})$ inequality by a positive integer n_j and every (Σ_j) inequality by $(\alpha - n_j)$, where α is an integer larger than n_j. The composite inequality will be such that $\forall i \in I_p$, $\bar{y}_i$ will have a coefficient $\sum_{j \in \bar{M}_i \cap J_q} n_j$ and x_{ij} will appear only if $j \notin M_i$, i.e. if $i \neq \sigma(j)$, with the coefficient $n_j + (\alpha - n_j) = \alpha$.

Proposition II.1

If the system

$$\sum_{j\in\bar{M}_i \cap J_q} n_j = \alpha \quad \forall i\in I_p$$

has a positive integer solution (n_j,α), then

$$\sum_{j\in J_q} \sum_{\substack{i\in I_p \\ i\neq\sigma(j)}} x_{ij} + \sum_{i\in I_p} \bar{y}_i \leq m + n - p - q - 2 \qquad (F_{\sigma,p,q})$$

is a valid inequality for SPLP(m,n) .

Proof:

Assume that (n_j,α), positive integers, satisfy

$$\sum_{j\in\bar{M}_i \cap J_q} n_j = \alpha \qquad \forall i\in I_p .$$

Then the composite inequality will read

$$\alpha\left[\sum_{j\in J_q} \sum_{\substack{i\in I_p \\ i\neq\sigma(j)}} x_{ij} + \sum_{i\in I_p} \bar{y}_i\right] \leq \alpha(m-p) + \sum_{j\in J_q} (\alpha - n_j)$$

$$\leq \alpha(m-p) + (n-q)\alpha - \sum_{j\in J_q} n_j$$

We know that $\forall i\in I_p$, $M_i \cap J_q \neq \emptyset$

and $\sum_{j\in M_i \cap J_q} n_j < \sum_{j\in\bar{M}_i \cap J_q} n_j = \alpha$

therefore

$$\alpha < \sum_{j\in J_q} n_j = \sum_{j\in\bar{M}_i \cap J_q} n_j + \sum_{j\in M_i \cap J_q} n_j < 2\alpha$$

and $\quad 1 < \frac{1}{\alpha} \sum_{j\in J_q} n_j < 2$, i.e. $\left\lfloor -\frac{1}{\alpha} \sum_{j\in J_q} n_j \right\rfloor = -2$.

Then $\quad \sum_{j\in J_q} \sum_{i\in I_p\setminus\sigma(j)} x_{ij} + \sum_{i\in I_p} \bar{y}_i \leq m + n - p - q - 2$

is a valid inequality for SPLP(m,n).

Proposition II.2

The system $\sum_{j\in\bar{M}_i \cap J_q} n_j = \alpha$, $\forall i\in I_p$

has a positive integer solution .

Proof:

$$\alpha = \sum_{j\in J_q} n_j - \sum_{j\in M_i \cap J_q} n_j$$

is a constant for all $i\in I_p$, so that n_j , $j\in J_q$, must be such that $\forall i \neq i'$, $i, i'\in I_p$:

$$\sum_{j\in M_i \cap J_q} n_j = \sum_{j\in M_{i'} \cap J_q} n_j .$$

Consider $\beta = \mathrm{LCM}\{(|M_i \cap J_q|),\ i\in I_p\}$.

Then, set $n_j = \dfrac{\beta}{|M_{\sigma(j)} \cap J_q|}$, $j\in J_q$.

n_j will be a positive integer. As the sets $M_i \cap J_q$ form a partition of J_q, all n_j are determined in this way, and

$$\alpha = \sum_{j \in J_q} n_j - \sum_{j \in M_i \cap J_q} n_j = \beta(m - p - 1)$$

is a positive integer.

Corollary II.3

$F_{\sigma,p,q}$ is a valid inequality which eliminates $X(\sigma, K, \tau, p)$.

Example:

Consider again the example of section 1. Suppose $q=0$.

We must find α, n_j, positive integers such that

$$n_2 + n_3 + n_4 + n_5 + n_6 + n_7 - \alpha = 0 \qquad (1)$$

$$n_1 + n_3 + n_4 + n_5 + n_6 + n_7 - \alpha = 0 \qquad (2)$$

$$n_1 + n_2 + n_4 + n_5 + n_6 + n_7 - \alpha = 0 \qquad (3)$$

$$n_1 + n_2 + n_3 + n_7 - \alpha = 0 \qquad (4)$$

It is clean that $n_1 = n_2 = n_3 + n_7 = n_4 + n_5 + n_6$.

Thus $\alpha = n_1 + n_3 + n_7 + n_3 + n_7$

$= n_1 + 2n_3 + 2n_7 \qquad (1)$

$= n_1 + 2n_3 + 2n_7 \qquad (2)$

$= 2n_1 + n_3 + n_7 \qquad (3)$

$= 2n_1 + n_3 + n_7 \qquad (4)$

and finally $\alpha = 3n_1$.

Choose $\beta = \text{LCM}(|M_i|) = \text{LCM}(1,2,3) = 6$.

Then:

j	1	2	3	4	5	6	7
n_j	6/1=6	6/1=6	6/2=3	6/3=2	6/3=2	6/3=2	6/2=3

and $\alpha = 18$.

The valid inequality $F_{\sigma,p,q}$ reads:

$$\left.\begin{array}{llllll} x_{12}+x_{13}+x_{14}+x_{15}+x_{16}+x_{17} & & & & & \\ +x_{21} & +x_{23}+x_{24}+x_{25}+x_{26}+x_{27} & & & & \\ +x_{31}+x_{32} & & +x_{34}+x_{35}+x_{36} & & & \\ +x_{41}+x_{42}+x_{43} & & & +x_{47} & & \end{array}\right\} + \bar{y}_1+\bar{y}_2+\bar{y}_3+\bar{y}_4 \leq 9 \ .$$

For $X(\sigma, K, \tau, p)$ the left-hand side is $7 + 4 \times 2/3 = 9\ 2/3$ and $F_{\sigma,p,q}$ is violated.

III. SOME FACETS OF SPLP(m,n) .

Consider now $p = 0$, $q = n - m$ and $J_q = K$.

$F_{\sigma,p,q}$ becomes:

$$\sum_{j \in K} \sum_{i \neq \sigma(j)} x_{ij} \leq \sum_{i} y_i + m - 2 \qquad (F_{\sigma,K}) \ .$$

Rather than being derived from the previously treated valid inequalities, these facets could also be generated more directly with $n_j = 1$. However, the simple form of the valid

inequalities, and their direct relationship with fractional vertices, may make them important in their own rights, for example in a simplex-based algorithm.

Proposition III.1

$F_{\sigma, K}$ is a facet of SPLP(m,n) .

Proof:

We must show that $F_{\sigma, K}$ contains $mn + m - n$ linearly independent integer vertices of SPLP(m,n), and that there exists at least one integer vector satisfying $F_{\sigma,K}$ as a strict inequality.

a) (1) Let $\overset{i}{X}$ be defined by $x_{ij} = 1$, $\forall j$, $y_i = 1$, $y_k = 0$, $\forall k \neq i$.

$\overset{i}{X}$ satisfies $F_{\sigma, K}$ as an equality:

$$\sum_{j \in K} \sum_{i \neq \sigma(j)} x_{ij} = m - 1 = \sum_{i} y_i + m - 2 .$$

There are m such vertices $\overset{i}{X}$.

(2) Consider $j^0 < j^1 < j^2 < \ldots < j^h$,

such that $\nexists\ k < j^0$ with $\sigma(k) = \sigma(j^0)$

and $\sigma(j^0) = \sigma(j^1) = \ldots = \sigma(j^h)$.

Let $i \neq \sigma(j^0)$. Then

$X_i^{j^0}$ will be defined by $x_{ij^0} = 1$, $x_{ik} = 0$, $\forall k \neq j^0$

$X_i^{j^1}$ by $x_{ij^0} = x_{ij^1} = 1$, $x_{ik} = 0$, $\forall k \neq j^0, j^1$.

$\vdots$

$X_i^{j^h}$ by $x_{ij^0} = x_{ij^1} = \ldots x_{ij^h} = 1$,
$x_{ik} = 0$, $\forall k \neq j^0, \ldots, j^h$,

and for all $X_i^{j^\ell}$, $\ell = 0, \ldots, h$, all other shipments are made in row $\sigma(j^0)$. $x_{\sigma(j^0),j} = 1$ $\forall j \neq j^0, j^1, \ldots, j^\ell$ and $y_i = 1$; $y_{\sigma(j^0)} = 1$ if $\sigma(j^0) \neq i$ $\forall i$. There are $n(m-1)$ such X_i^j .

(3) We want to prove that $\{\overset{i}{X}\}$ $i \in I$, $\{X_i^j\}$ $j \in J$, $i \neq \sigma(j)$ are linearly independent by showing that

$$\sum_i \alpha_i \overset{i}{X} + \sum_j \sum_{i \neq \sigma(j)} \beta_{ij} X_i^j = 0 \implies \begin{cases} \beta_{ij} = 0, \forall i, \forall j : i \neq \sigma(j) \\ \alpha_i = 0, \forall i \end{cases}$$

Let us form a matrix whose rows are indexed by $\bar{y}_1, \ldots, \bar{y}_m$, $x_{11}, \ldots, x_{mn}$, and whose columns are $\{\overset{i}{X}\}$, $i \in I$, and $\{X_i^j\}$, $j \in J$, $i \neq \sigma(j)$.

The rows $x_{i,\sigma/\overset{-1}{X}(i)}$ have only one 1 in column $\overset{i}{X}$, so that $\alpha_i = 0$, $\forall i$.

With the same notation as above, we obtain

$$\beta_{ij^0} + \beta_{ij^1} + \ldots + \beta_{ij^h} = \beta_{ij^1} + \ldots + \beta_{ij^h} = \beta_{ij^h} ,$$

$$\forall j^0 < j^1 < \ldots < j^h \text{ with } \sigma(j^0) = \ldots = \sigma(j^h) \neq i$$

so that $\beta_{ij^0} = \beta_{ij^1} = \ldots = \beta_{ij^{h-1}} = 0$.

The only remaining columns are $X_i^{j^h}$ with $i \neq \sigma(j^h)$, say $X_i^{j_i}$, $X_i^{j_i'}$, $\ldots$, $X_i^{j_i^{(k)}}$, $i \in I$.

It is easy to see that

$\forall i : \beta_{ij_i} = \beta_{ij_i'} = \ldots = \beta_{ij_i^{(k)}} = \beta_i$, say,

from the x_{ij} rows, since they now appear independently in equal sums whose other terms are identical. Finally, by substituting into the $\bar{y}_i$ rows, one must solve the system

$$\begin{bmatrix} 0 & 1 & & 1 \\ 1 & & & \\ & & & 1 \\ 1 & & 1 & 0 \end{bmatrix} \begin{pmatrix} \beta_1 \\ \\ \beta_m \end{pmatrix} = 0 \quad .$$

We know that the matrix of this system is nonsingular and that all β_i must be zero.

The $m + n(m-1) = mn + m - n$ vertices are linearly independent.

b) The vector $(x,\bar{y})$, defined by

$$x_{ij} = \begin{cases} 1 & i=\sigma(j),\ j \in J, \\ 0 & \text{otherwise} \end{cases}$$

$$\bar{y}_i = 0 \quad \forall i \ ,$$

satisfies $F_{\sigma,K}$ as a strict inequality.

c) $F_{\sigma,K}$ is therefore a facet of SPLP (m,n).

Example m =3, n=7, $M_1 = \{1\}$, $M_2 = \{2,6,7\}$, $M_3 = \{3,4,5\}$.

The facet is $x_{12}+x_{13}+x_{21}+x_{23}+x_{31}+x_{32} \leq y_1+y_2+y_3+1$

	x_1^2	x_1^3	x_1^4	x_1^5	x_1^6	x_1^7	x_2^1	x_2^3	x_2^4	x_2^5	x_3^1	x_3^2	x_3^6	x_3^7	$\bar{x}^1$	$\bar{x}^2$	$\bar{x}^3$
$\bar{y}_1$							1	1	1		1	1	1			1	1
2		1	1	1							1				1		1
3	1				1	1	1								1	1	
x_{12}	1				1	1	1				1				1		
13		1	1	1			1				1				1		
14			1	1			1				1				1		
15				1			1				1				1		
16					1	1	1				1				1		
17						1	1				1				1		
21	1				1	1	1					1	1	1		1	
23	1				1	1		1	1	1		1	1	1		1	
24	1				1	1			1	1		1	1	1		1	
25	1				1	1				1		1	1	1		1	
31		1	1	1				1	1	1	1						1
32		1	1	1				1	1	1		1	1	1			1
36		1	1	1				1	1	1			1	1			1
37		1	1	1				1	1	1				1			1
11															1		
22																1	
26	1											1				1	
27	1				1							1	1			1	
33																	1
34		1						1									1
35		1	1					1	1								1

Remarks.

In the $(x,\bar{y})$ space, by varying p, K, and σ, one obtains quite a number of vertices of the form $X(\sigma, K, \tau, \rho)$.

For instance, for m=3, n=4, one has 36 fractional vertices of the form $X(\sigma,K,\tau,\rho)$ vs. 81 integer vertices.

Similarly, one obtains $\binom{n}{m}.m!$ facets $F_{\sigma,K}$.

For example, with m=3 and n=4, one has 24 facets.

Notice that some fractional points are eliminated by several facets. For instance, the fractional solution

$$x = \begin{bmatrix} 1/2 & 0 & 1/2 & 1/2 \\ 0 & 1/2 & 1/2 & 0 \\ 1/2 & 1/2 & 0 & 1/2 \end{bmatrix} \qquad y = \begin{bmatrix} 1/2 \\ 1/2 \\ 1/2 \end{bmatrix}$$

is eliminated by both

$$x_{11} + x_{31} + x_{22} + x_{32} + x_{13} + x_{23} \leq y_1 + y_2 + y_3 + 1 \quad , \text{ and}$$

$$x_{14} + x_{34} + x_{22} + x_{32} + x_{13} + x_{23} \leq y_1 + y_2 + y_3 + 1 \quad .$$

On the other hand, each facet eliminates m^{n-m} fractional vertices with $p = 0$, and many more with $p > 0$.

IV. DUALITY GAPS AND FACETS.

We shall now condider a special family of SPLP(m,n)'s, for which the structure of the cost matrix is such that the continuous and the integer optimal solutions are known explicitly and the duality gaps are very large. For these problems we shall show that a facet of the form $F_{\sigma, K}$ bridges the duality gap.

Let σ be a surjection from J into I. Let $c_{\sigma(j),j}$ be infinite, and let $c_{ij} = c$, $\forall i \neq \sigma(j)$. In other words, the shipment costs are either infinite or equal to constant c. Let $f_i = L$, $\forall i \in I$. We shall denote this problem by SPLP(σ,L).

Proposition IV.1

The relative duality gap for SPLP(σ,L) cannot exceed 1/2.

Proof:

A continuous optimal solution will be determined by:

$\forall j \in J$: $x_{\sigma(j)j} = 0$,

$\forall i \in I$, $i \neq \sigma(j)$, $x_{ij} = 1/(m-1)$

and $\forall i \in I$, $y_i = 1/(m-1)$.

The cost of this solution is

$$z_C = m \cdot \frac{L}{m-1} + n.(m-1).\frac{c}{m-1} = \frac{mL}{m-1} + n.c.$$

An integer solution will correspond to two open plants (given that at least one route is prohibited from an open plant by means of an infinite cost), and an optimal solution will, for instance, be defined by:

$y_{i_1} = y_{i_2} = 1$, $i_1, i_2 \in I$, and say $i_1 < i_2$

$x_{i_1 j} = 1 \quad \forall j \in M_{i_1}$, $x_{i_2 j} = 1 \; \forall j \in M_{i_1} \subset \overline{M_{i_2}}$,

with a total cost of

$z_I = 2L + nc$.

Then the relative duality gap α will be

$$\alpha = \frac{z_I - z_C}{z_I} = \frac{2L + nc - \frac{mL}{m-1} - nc}{2L+nc} = \frac{(m-2)L}{(m-1)(2L+nc)}$$

$$< \frac{(m-1)L}{(m-1)(2L)} = \frac{1}{2}$$

In order to generate a problem with a given relative duality gap α , one must choose

$$(m-2)\,L = 2\,\alpha\,(m-1)\,L + \alpha\,(m-1)\,nc$$

or $mL - 2\alpha mL + 2\alpha L - 2L = \alpha(m-1)nc$,

i.e. $L = \frac{\alpha(m-1)nc}{m(1-2\alpha)+2(\alpha-1)}$

This makes sense only if $m(1 - 2\alpha) + 2(\alpha - 1) > 0$, i.e. if

$$m > \frac{2(1-\alpha)}{1-2\alpha} ,$$

since we know that $\alpha < 1/2$.

Example:

For $\alpha = 10/21$ (very close to 1/2), we must have

$m > (2(1- \frac{10}{21}))/(1-2 \times \frac{10}{21}) = (2 \times 11)/1 = 22$.

Take m = 23. Then L should be equal to

$L = ((10/21)(22)nc)/(23(1-(20/21))+2((10/21)-1)) = 220\,nc.$

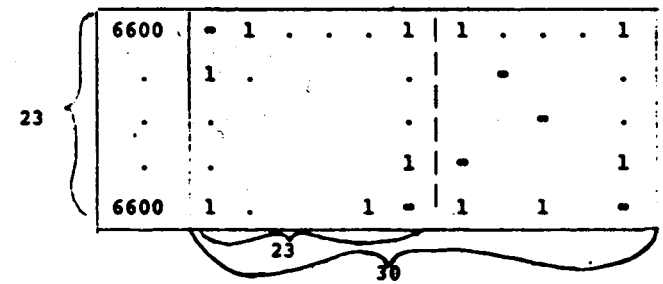
If n = 30 and c = 1 , for instance, we obtain L = 6600.

Then, $z_I = 2.\ 6600 + 30 = 13230$, and

$z_C = 30 + (23/22)\ 6600 = 6930$, so that indeed the gap is

$\alpha = (z_I - z_C)/z_I = (13230 - 6930)/13230 = 10/21$.

The cost matrix looks as follows:



Proposition IV.2

Given a subset $K \subset J$ such that $\sigma(K) = I$, $|K| = m$

$F_{\sigma, K}$ bridges the gap for SPLP (σ, L).

Proof:

Add $F_{\sigma, K}$ to the constraints of SPLP(σ,L) and write

the corresponding dual problem:

Min $\sum_i f_i \cdot y_i + \sum_j \sum_i c_{ij} \cdot x_{ij}$	Max $\sum_j v_j - (m-2)\,t = z_d$
$\forall j$: $\sum_i x_{ij} = 1$	v_j unrestricted in sign
$\forall i,\forall j$: $y_i - x_{ij} \geq 0$	$w_{ij} \geq 0$
$\sum_i y_i - \sum_{j\in K} \sum_{i\neq\sigma(j)} x_{ij} \geq 2-m$	$t \geq 0$
$\forall i$: $y_i \geq 0$	$\sum_j w_{ij} + t \leq f_i$
$\forall j\in K, \forall i \neq \sigma(j)$: $x_{ij} \geq 0$	$v_j - w_{ij} - t \leq c_{ij}$
$\forall j\in K$: $x_{\sigma(j)j} \geq 0$	$v_j - w_{\sigma(j)j} \leq c_{\sigma(j)j}$
$\forall j\in J\setminus K, \forall i$: $x_{ij} \geq 0$	$v_j - w_{ij} \leq c_{ij}$

The following solution is feasible for the dual:

$w_{ij} = 0 \quad \forall i, \forall j$

$t = f_i = L,$

$v_j = c + L \quad \forall j\in K$

$v_j = c \quad \forall j\in J\setminus K$.

Then $z_d = \sum_{j\in K} (c + L) + \left(\sum_{j\in J\setminus K} c\right) - L\,(m-2)$

$= m\,(c + L) + (n - m)\,c - L\,m + 2\,L$,

that is $z_d = 2\,L + n\,c$.

Since we know a primal solution with the same objective function value, this is the optimal value for the dual and there is no more duality gap.

References:

Pertinent references are either the following papers, or can be found referenced in them.

[1] E. Balas, M. Padberg, Set Partitioning, in B. Roy (ed), Combinatorial Programming: Methods and Applications, D. Reidel Pub. Co. (1975).

[2] G. Cornuejols, M. Fisher, G.L. Nemhauser, On the Uncapacitated Location Proble, Annals of Discrete Mathematics 1 (1977), 163 - 177.

[3] M. Guignard, K. Spielberg, Algorithms for Exploiting the Structure of the Simple Plant Location Problem, Annals of Discrete Mathematics 1 (1977), 247 - 271.

[4] M. Guignard, K. Spielberg, A Dual Method for the Mixed Plant Location Problem, Wharton School, Department of Statistics, Report #20, January 1977, revised December, 1978.

[5] J. Krarup, P. Pruzan, Selected Families of Discrete Location Problems, Part III: The Plant Location Family, University of Calgary Working Paper # WP.12.77.