Integer Programming and Combinatorics

by Fred Glover

Integer programming and combinatorics are closely linked. In the broadest sense their domains are identical, though in practice some problems are popularly viewed to fall more in the province of one than the other. "Combinatorics," as spoken of here, is the field of "combinatorial optimization," whose problems characteristically have the form of seeking a "best" subset of items (decisions, activities, etc.) satisfying particular criteria from a structured finite set of alternatives. "Best" is evaluated in terms of maximizing or minimizing some functional. The structure of the finite set, whatever it may be, is the feature to be exploited in devising a systematic method to solve the problem, rather than simply enumerating and comparing all alternatives (which in problems of real world significance can require astronomical amounts of computation). A remarkable number of practical problems fall into the integer programming/combinatorics area. Those associated with the "combinatorics" label usually are problems that are conveniently expressed in the specialized terminology of graph or matroid theory. Those associated with the "integer programming" label usually are more conveniently expressed in mathematical programming terminology. Then there is a no-man's-land of problems that fall under either or both labels according to the flavor the author wishes to impart to them.

This chapter focuses primarily (though not exclusively) on those problems and formulations that commonly find expression

Published in Handbook of Operations Research, Elmaghraby and Moder, eds, December 1977

in integer programming terminology. (The graph theoretic branch of combinatorics is covered in Chapter 3.)

Loosely speaking, integer programming is the domain of mathematical optimization in which some or all of the problem variables are required to assume integer (whole number) values. The nonlinear character of the integer requirement is subtle enough to accommodate an unexpected variety of other nonlinearities, permitting most "nonlinear" integer programs to be sorted into a linear component and an integer component. Consequently, integer programming is frequently regarded as the domain of linear mathematical optimization (specifically, "linear programming") in which some or all of the problem variables are integer-constrained.

An Illustration

A very simple example of an integer program is that of a gardener who needs 107 pounds of fertilizer for his lawn, and has the option of buying it either in 35-pound bags at $14 each or in 24-pound bags at $12 each. His goal is to buy (at least) the 107 pounds he needs at the cheapest cost. Letting x_1 be the number of 35-pound bags he buys and letting x_2 be the number of 24-pound bags he buys, he seeks to

$$\begin{aligned} \text{Minimize} \quad & 14x_1 + 12x_2 \\ \text{subject to} \quad & 35x_1 + 24x_2 \geq 107 \\ & x_1, x_2 \geq 0 \text{ and integer.} \end{aligned}$$

The "and integer" stipulation means that the gardener can't buy half or a third of a bag of fertilizer, but must buy a whole bag or none at all. Without this stipulation the

problem is an example of an ordinary linear programming problem.

Significance of the Integer Restriction

To get a very rudimentary grasp of how the integer restriction can affect the character of the problem, consider what would happen if the restriction were missing. The gardener would then observe that fertilizer in the 35-pound bags costs 40¢ a pound and fertilizer in the 24-pound bags costs 50¢ a pound, and immediately perceive that his best policy would be to buy 3-2/35 of the 35-pound bags. However, in the presence of the integer restriction, his best policy is to buy only 1 of the 35-pound bags and buy 3 of the 24-pound bags. Clearly, his best policy has changed drastically, a not at all unusual occurrence of requiring the variables to be integer valued. (In fact, an example exists [56] of a "conditional transportation problem" that can be summarized by a 5 x 5 cost matrix and whose linear programming solution can be rounded in more than a million ways, none of which yields a feasible--let alone optimal--solution.)

Areas of Application

In the foregoing example, the interpretation and relevance of the integer restriction was clear, as it would also be if one wished to determine an optimal number of airplanes, cargo ships, or human beings. However, a variety of problems that seem on the surface to have little or nothing to do with "integer constrained linear optimization" can nevertheless be

given an integer programming formulation. Indeed, the requirement of discreteness indirectly, if not directly, pervades many significant classes of problems and provides integer programming with application in an uncommonly wide variety of theoretical and practical disciplines. Production sequencing, job shop scheduling, logistics, plant location, assembly line balancing, mineral exploration, capital budgeting, resource allocation, and facilities planning constitute a few of the important industrial problem areas that frequently fall within the wider domain of integer programming.

The uses of integer programming are not confined to industry, however, and also find application in many other combinatorial optimization problems arising in engineering and scientific contexts. Computer design system reliability, prime implicant selection, signal coding, and energy storage system design all give rise to classes of problems with integer programming formulations.

Integer programming also has applications to economic analysis. The assumption of continuity underlying traditional economic theory is incompatible with the existence of indivisible plants, set up costs, and developmental expenditures. Conclusions reached by marginal analysis must, therefore, often be amended (or stated with appropriate qualification) to accommodate considerations which are the focus of integer programming.

Mathematical Formulation of Linear and Integer Programs

A standard formulation of the linear programming

problem is:

$$\text{Minimize} \quad \sum_{j=1}^{n} c_j x_j$$

$$\text{subject to} \quad \sum_{j=1}^{n} a_{ij} x_j \leq b_i \qquad i = 1, \ldots, m$$

$$x_j \geq 0 \qquad j = 1, \ldots, n$$

or, in matrix notation

$$\text{Minimize} \quad cx$$

$$\text{subject to} \quad Ax \leq b$$

$$x \geq 0$$

where $c = (c_j)_{1 \times n}$, $x = (x_j)_{n \times 1}$, $A = (a_{ij})_{m \times n}$ and $b = (b_i)_{m \times 1}$. The matrix inequality $Ax \leq b$ can be used to summarize constraints of the form $Dx \geq d$ and $Dx = d$ by the well known equivalences $Dx \geq d \Longleftrightarrow -Dx \leq -d$ and

$$Dx = d \Longleftrightarrow \begin{cases} Dx \leq d \\ -eDx \leq -ed \end{cases}$$

where e denotes the vector all of whose components equal 1, interpreted to be a row or column vector according to context. (Thus $-eDx \leq -ed$ sums the row inequalities of $Dx \leq d$ and reverses their sign.) Also, a problem with the objective Maximize dx can be accommodated by the foregoing minimization formulation by letting $c = -d$.

Pure and Mixed Integer Programs

A linear programming problem in which all components of x are additionally constrained to be integer is called a pure integer programming problem and one in which only some of the

components of x are additionally constrained to be integer is called a mixed integer programming problem. Variables that are not integer constrained are called continuous variables. (Thus, in the ordinary linear programming problems all variables are continuous.)

The words "integer programming" often have reference to the pure integer problem but we allow them also to refer to the more general mixed problem depending on the setting.

Examples of Integer Programming Problems

The Knapsack (or Loading) Problem

One of the simplest integer programming problems is the knapsack problem, so called because it can be interpreted as a problem of selecting a best set of items to go in a hiker's knapsack, given the value he attaches to these items and an upper limit on the amount of weight he can carry. A less well known interpretation of the knapsack problem goes back to the days of one of the early kings of Siam who was seeking an ideal composition for his harem.* The king's preferences were strongly influenced by hair color and, prompted by the royal economist, he established a set of subjective utilities (called "yoodles" in Siamese) expressing these preferences as follows:

Hair types:	Blonde	Brunette	Redhead	Albino	Bald
Yoodles:	7	5	9	6	3

*This historical information was contributed by Dr. Eugene Woolsey, who assures me it is true.

Thus, letting x_1 denote the number of blondes in his harem, x_2 the number of brunettes, etc., the king formulated his objective function to be

$$\text{Maximize } 7x_1 + 5x_2 + 9x_3 + 6x_4 + 3x_5$$

Siamese harem girls were known to be hearty eaters so the king got together with his royal cook and figured out how much it cost to feed each type of girl for a week, and counting on a maximum allowable food bill of one hundred Siamese dolors (the currency of that time), came up with a budget constraint of

$$54x_1 + 35x_2 + 57x_3 + 46x_4 + 19x_5 \leq 100$$

Not knowing what to do with a fractional part of a girl, and unable to recognize a negative girl if he saw one, the king added as an afterthought

$$x_j \geq 0 \text{ and integer, } j = 1, \ldots, 5$$

and thus arose the first example of the knapsack problem (or, if one prefers, the Siamese Harem problem).

In practice the variables of a knapsack problem are often constrained to satisfy

$$x_j \leq 1$$

(a restriction which the Siamese king himself was obliged to observe in later years), in which case it is sometimes called the "0-1" knapsack problem. More generally, the variables may be required to satisfy $x_j \leq U_j$, in which case it is called the "bounded variable" knapsack problem.

The knapsack problem is extremely easy to solve as an ordinary linear program simply by ranking the variables

according to the ratios of the objective function coefficients to the constraint coefficients. That is, writing the problem as

$$\text{Maximize} \quad \Sigma d_j x_j$$

$$\text{subject to} \quad \Sigma a_j x_j \leq b$$

$$x_j \geq 0 \text{ and integer}$$

where $d_j > 0$ and $a_j > 0$ for all j, the variables are ranked by a "primed indexing" so that

$$d_{1'}/a_{1'} \geq d_{2'}/a_{2'} \geq \cdots$$

where 1', 2', ... constitute a permutation of the numbers 1, 2, ... Then the problem is solved by taking $x_{1'}$ as large as possible without violating the constraint $\Sigma a_j x_j \leq b$ (or any bound imposed on $x_{1'}$, as in the 0-1 problem), then taking $x_{2'}$ as large as possible subject to the value already assigned $x_{1'}$, and so on. Thus, in the Siamese Harem example we have

$$1' = 5,\ 2' = 3,\ 3' = 2,\ 4' = 4,\ 5' = 1$$

and the optimal solution for the unbounded problem is

$$x_5 = 100/19,\ x_3 = x_2 = x_4 = x_1 = 0$$

while the optimal solution for the 0-1 problem is

$$x_5 = 1,\ x_3 = 1,\ x_2 = 24/35,\ x_4 = x_1 = 0$$

A frequently encountered generalization of the knapsack problem is the so called "multi-dimensional" knapsack problem whose formulation "Minimize cx subject to $Ax \leq b$, etc.," is characterized by $c \leq 0$ and $A \geq 0$. An obvious and sometimes useful property of the multidimensional knapsack problem is that any fractional (i.e., "not totally integer") x vector that satisfies $Ax \leq b$ can be rounded down to produce an integer x that

satisfies $Ax \leq b$. Practical applications of the multidimensional knapsack problem arise in capital budgeting, forestry, and cargo loading.

We have given an extended description of the knapsack problem not only for historical interest but also because, in spite of its apparent simplicity, this problem reappears in several guises and contexts in integer programming. (The problem of the gardener described at the beginning of this article is one form of the knapsack problem.) In fact, a number of solution techniques in integer programming can be interpreted as a direct extension of a "knapsack method," or involve the creation of "knapsack subproblems" whose solution provides valuable information for the solution of the original problem. More will be said about this subsequently.

The Fixed Charge Problem

The fixed charge problem is characterized by "one shot" outlays (or set up costs) that are incurred in the process of starting or renewing a business venture. For example, a manager who is faced with deciding which of several machines to buy, automobile plants to build, oil wells to drill, or land areas to develop, must account not only for the continuing costs of operation (once the projects are underway) but also for the initial fixed cost required to initiate the projects. This type of problem, especially in the presence of imbedded networks, is one of the most frequently occurring problems in practical applications.

A typical fixed charge problem has the form:

$$\text{Minimize } cx + \alpha y$$

$$\text{subject to } Ax \leq b$$

$$x \geq 0,\ y \geq 0$$

where the vectors x and y have the same dimension and $x_j > 0$ implies $y_j = 1$. (Here, of course, c, A and b are not intended to represent the c, A and b of the general integer programming formulation given earlier.) The stipulation that $x_j > 0$ implies $y_j = 1$ conveys the meaning that to make, buy, or process any positive amount of x_j incurs a fixed charge of α_j (> 0). It may be noted that $x_j = 0$ automatically implies $y_j = 0$ at any minimizing solution, since whenever $x_j = 0$ and $y_j = 1$ a better solution can always by obtained by setting $y_j = 0$ (without causing any of the constraints to become unsatisfied). However, the preceding formulation is not acceptable for integer programming since the constraint "$x_j > 0$ implies $y_j = 1$" is "logical" rather than linear. To put the problem in acceptable form, we assume the existence of a bound U_j so that $x_j \leq U_j$ is satisfied for all values of x_j compatible with $Ax \leq b$, $x \geq 0$. Then the constraint

$$U_j y_j \geq x_j$$

always holds for $y_j = 1$, and moreover, if y_j is integer valued, then $x_j > 0$ implies $y_j \geq 1$ and hence $y_j = 1$ in a minimizing solution (by the same reasoning that shows $x_j = 0$ implies $y_j = 0$). Thus, the integer programming formulation of the fixed charge problem may be written

$$\text{Minimize } cx + \alpha y$$

$$\text{subject to } Ax \leq b$$

$$x - Uy \leq 0$$

$$x \geq 0,\ y \geq 0,\ y \text{ integer}$$

where $U = (U_j)$. It is also possible to add the constraint $y \leq e$ (i.e., $y_j \leq 1$ for all j) to explicitly acknowledge that the y_j are 0-1 variables, but this is unnecessary under the assumption that $\alpha > 0$, as already noted.

The Harmonious Expedition Problem and the Combinatorial Matching Problem

The harmonious expedition problem involves a group of explorers who wish to embark on an expedition and take along as many of their members as possible. However, certain pairs of members do not get along together, and if one of the pair goes on the expedition the other will stay home. The problem may be formulated as a 0-1 integer program by defining

$$x_j = \begin{cases} 1 \text{ if member } j \text{ goes along} \\ 0 \text{ if member } j \text{ stays home} \end{cases}$$

Then the objective is to

$$\text{Maximize } \Sigma x_j$$

subject to the constraints

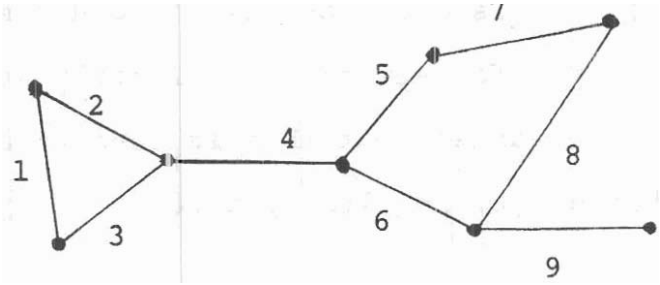
$$x_h + x_k \leq 1$$

for those pairs of members h and k that are not compatible. The formulation is completed by requiring $1 \geq x_j \geq 0$ and x_j integer for all j.

The harmonious expedition problem is closely related to the maximum cardinality matching problem from combinatorial

graph theory. Figure 1 shows a "typical" graph consisting of nodes (points) and edges (lines) connecting them.

Figure 1



A matching in such a graph is defined to be a set of edges that share no common endpoints. Thus, by the numbering of the edges shown in Figure 1, edges 1 and 4 constitute a matching, while edges 1, 5 and 7 do not (since edges 5 and 7 share a node in common). The maximum cardinality matching problem is to determine a matching containing the largest number of edges. Its formulation is the same as that of the harmonious expedition problem by defining

$$x_j = \begin{cases} 1 \text{ if edge } j \text{ is in the matching} \\ 0 \text{ if edge } j \text{ is not in the matching} \end{cases}$$

introducing the constraints $x_h + x_k \leq 1$ for those pairs of edges h, k that share a common endpoint.

A simple generalization of the harmonious expedition problem arises by attaching a "worth" d_j to each member j, and seeking to maximize the "total worth" $\Sigma d_j x_j$ of the expedition rather than the number of members that go along. The corresponding generalization of the graph theory problem is called the maximum weight matching problem, and arises by assigning

a weight to each edge and seeking a matching whose edge weights sum to the greatest value. Other generalizations readily come to mind. For example, in the harmonious expedition problem, one may identify sets rather than pairs of members that constitute an "inharmonious group." If S denotes such a set, and $|S|$ denotes the number of its elements, then one may enforce

$$\sum_{j \in S} x_j \leq |S| - 1$$

to assure that at least one member of S stays home.

The Efficacious Expedition Problem and the Combinatorial Covering Problem

In this problem a group of explorers seek to minimize the number of members who embark on an expedition. However, for each of several critical skills there must be at least one member possessing that skill who goes on the expedition in order for the expedition to be carried out successfully. Defining the variables x_j as for the harmonious expedition problem, and letting S_i denote the set of members possessing the ith skill, the problem is to

$$\text{Minimize } \Sigma x_j$$

subject to the constraints

$$\sum_{j \in S_i} x_j \geq 1$$

Again the formulation is completed by stipulating $1 \geq x_j \geq 0$ and x_j integer.

There is, as might be suspected, a closely related problem from combinatorial graph theory, in this case called the minimum cardinality covering problem. A cover (or

covering) is a set of edges whose endpoints include all nodes of the graph. Thus, in Figure 1, the edges 1, 2, 3, 4, 5, 8, 9 constitute a cover, but the edges 1, 3, 5 and 9 do not. (No edge covers the node at the intersection of edges 7 and 8.) A minimum cardinality cover is one containing the fewest number of edges, and its formulation is the same as that of the efficacious expedition problem by defining the x_j as for the matching problem and letting S_i denote the set of edges that intersect at a node i (thus yielding one constraint for each node of the graph).

Immediate generalizations of these problems arise by assigning weights to edges and requiring the nodes to be multiple covered.

A Delivery Problem

A company delivers merchandise to its customers each day by truck (or railroad, air, barge, etc.). There are a variety of feasible delivery routes, each one accommodating a specified subset of customers and capable of being traveled by a single carrier in a day. Each route also has a "cost" associated with it, which may be just the length of the route, or the cost of fuel to travel the route, etc. The goal is to select a set of routes that will make it possible to provide a delivery to each customer and, subject to this, minimize total cost.

Define

$$\begin{cases} 1 \text{ if route } j \text{ is selected} \end{cases}$$

and create an A matrix whose jth column has (constant) entries

$$a_{ij} = \begin{cases} 1 \text{ if customer is on route } j \\ 0 \text{ if not} \end{cases}$$

Denoting the cost of route j by c_j, the delivery problem may be written

$$\text{Minimize } cx$$

$$\text{subject to } Ax = e$$

$$x \geq 0 \text{ and integer,}$$

where as before e is a vector of 1's (a column vector in this case).

The delivery problem may also include a constraint of the form

$$\Sigma x_j \leq k,$$

which restricts the total number of routes selected to be no more than k (because, for example, the company can operate at most k trucks on a given day).

Note that for the specified form of A, the constraints "Ax = e, $x \geq 0$ and integer" automatically imply $x_j = 0$ or 1 for each j (this would also be true if Ax = e were replaced by $Ax \leq e$). We assume of course that there are no columns of A that are all 0's (which would correspond to a route without any customers). The matrix equation Ax = e stipulates that each customer will receive exactly one delivery each day.

Multiple Alternative Problems

A variety of "dichotomous" or "multiple alternative" situations can be accommodated by the introduction of appropriately defined zero-one variables. Problems to which such

multiple alternative situations are relevant range from the design of a nuclear reactor complex to the determination of demand reservoir locations for a water resource allocation project.

The common ingredient in these problems is a set of constraining relations

$$A^{(1)}x \leq b^{(1)}$$
$$A^{(2)}x \leq b^{(2)}$$
$$\vdots$$
$$A^{(r)}x \leq b^{(r)}$$

out of which at least q are required to be satisfied while the remaining r - q may be satisfied or not. Corresponding to each $b^{(k)}$ let $M^{(k)}$ denote a vector of the same dimension (which may be different for different k) whose components are sufficiently large that $A^{(k)}x \leq b^{(k)} + M^{(k)}$ will be satisfied for all x vectors relevant to the problem under consideration. Thus, the constraint

$$A^{(k)}x \leq b^{(k)} + y_k M^{(k)}$$

where y_k is a 0-1 variable, corresponds to $A^{(k)}x \leq b^{(k)}$ for $y_k = 0$ and to $A^{(k)}x \leq b^{(k)} + M^{(k)}$ for $y_k = 1$. For at least q of the inequalities $A^{(k)}x \leq b^{(k)}$ to hold, at least q of the y_k must be 0. This is accommodated by introducing the constraints $A^{(k)}x \leq b^{(k)} + y_k M^{(k)}$ for k = 1, ..., r and requiring

$$\Sigma y_k \leq r - q.$$

This latter constraint may also be replaced by

$$\Sigma y_k = r - q$$

to accomplish the same result.

Zero-One Polynomial Problems

Nonlinear 0-1 programming problems of the form

$$\text{Minimize } f(x)$$

$$\text{subject to } g_i(x) \leq 0 \qquad i = 1, \ldots, m$$

$$x_j = 0 \text{ or } 1 \text{ for all } j$$

where f and g_i are polynomials, can be replaced by equivalent linear 0-1 programming problems by the rules

1. Replace all nonzero powers of x_j by x_j itself (since for $x_j = 0$ or 1, $x_j = x_j^2 = x_j^3$ etc.)
2. Replace each cross product $\pi_{j\in Q} x_j$ by a 0-1 variable x_Q which is required to satisfy

$$x_Q \geq \Sigma_{j\in Q} x_j + 1 - |Q|$$

and

$$x_Q \leq (\Sigma_{j\in Q} x_j)/|Q|$$

where $|Q|$ denotes the number of elements in Q.

The justification of the second rule follows from the fact that the indicated lower bound for x_Q is always redundant until $\Sigma_{j\in Q} x_j = |Q|$, (which occurs when $x_j = 1$ for all $j\in Q$), in which case it implies $x_Q \geq 1$, and the indicated upper bound always requires $x_Q < 1$ (hence $x_Q = 0$) until $\Sigma_{j\in Q} x_j = |Q|$, in which case it permits $x_Q = 1$. Thus $x_Q = 1$ if $x_j = 1$ for all $j\in Q$ (and $x_Q = 0$ otherwise), which identifies the value of x_Q to be the same as that of $\pi_{j\in Q} x_j$. Ways have been developed for expressing these polynomials still more effectively [28].

Equivalence of a Bounded Variable Integer Program to a 0-1 Integer Program

There are several ways to express an integer program in bounded variables as a problem in 0-1 variables.

To illustrate, suppose each integer variable x_j of the original problem can be required to satisfy $0 \leq x_j \leq U_j$ for some finite upper bound U_j. Then we may replace x_j by the linear expression

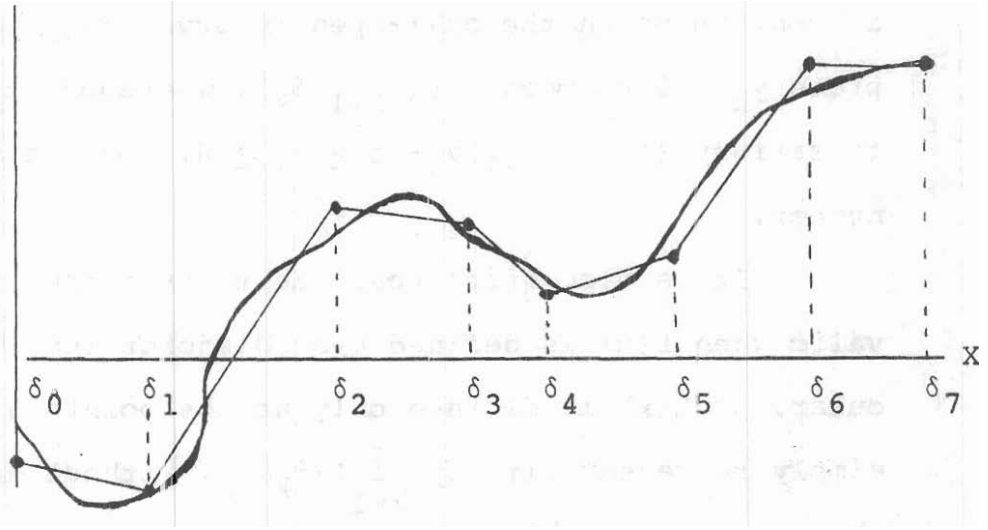
$$x_{0j} + 2x_{1j} + 4x_{2j} + \ldots + 2^r x_{rj}$$

where each x_{ij} is a 0-1 variable and r is the unique integer for which $2^r \leq U_j < 2^{r+1}$. Two other ways to replace an integer variable by a weighted sum of 0-1 variables arise simply by taking all weights equal 1 (yielding an ordinary sum of U_j different variables) and by taking the weights to be the positive integers 1, 2, 3, ..., U_j, with the added restriction that the sum of the 0-1 variables associated with these latter weights not exceed 1.

Because of the generality of the 0-1 problem, and because a high percentage of practical problems exhibit "naturally occurring" 0-1 variables, a good deal of attention has been devoted to methods and results for the 0-1 case. In fact, a variety of nonlinear programming problems can be expressed as integer programming problems (within a desired degree of approximation) by the use of 0-1 variables. Two of the more significant ways of doing this are as follows:

Piecewise Linear Approximation of a Separable Nonlinear Function

A nonlinear function $f(x)$ $(x = (x_j)_{n \times 1})$ is called separable if it can be written in the form $f(x) = f_1(x_1) + f_2(x_2) + \ldots + f_n(x_n)$, where each f_j is a function of the single variable x_j. If the functions f_j are sufficiently well-behaved, it may be possible to fit them with reasonable piecewise linear approximations without too much difficulty. In this context we shall for convenience drop the j subscript and understand $f(x)$ to designate one of the functions $f_j(x_j)$ where now x is a single variable rather than a vector. For example, such a function $f(x)$ and its linear approximation $L(x)$ might be as shown in the following diagram, where $f(x)$ is the curved line and $L(x)$ is the broken straight line.



The domain of these functions has been divided into intervals $[\delta_{k-1}, \delta_k]$ $k = 1, \ldots \delta$ over which $L(x)$ is an unbroken straight line. We let α_k denote the slope of $L(x)$ on the interval $[\delta_{k-1}, \delta_k]$ and observe that if $x \in [\delta_{k-1}, \delta_k]$ then $L(x) = L(\delta_{k-1}) + \alpha_k(x - \delta_{k-1})$. This fact makes it possible to represent $L(x)$ by a linear function as follows. Let y_k be a

0-1 variable and let z_k be a continuous variable with the interpretation that $y_k = 1$ corresponds to $x \in [\delta_{k-1}, \delta_k]$ and z_k is nonzero only if $y_k = 1$, in which case it represents the quantity $x - \delta_{k-1}$. Then, L(x) may be expressed as

$$\sum_{k=1} (L(\delta_{k-1})y_k + \alpha_k z_k)$$

given the constraints

$$\sum_{k=1}^{r} y_k = 1$$

and

$$(\delta_k - \delta_{k-1})y_k \geq z_k \geq 0,$$

$$y_k \geq 0 \text{ and integer} \qquad k = 1, \ldots, r$$

This representation can be made valid even if L(x) is not continuous. To do this, one selects the δ_k so that L(x) is continuous on the half-open interval $[\delta_{k-1}, \delta_k)$, and interprets $y_k = 1$ to mean $x \in [\delta_{k-1}, \delta_k)$, whereupon z_k is constrained to satisfy $(\delta_k - \delta_{k-1})y_k - \epsilon \geq z_k \geq 0$, for ϵ a small positive number.

It is also quite possible to make the representation valid when L(x) is defined over disjoint intervals. In particular, if L(x) is defined only at the points δ_k, then one can simply represent L(x) by $\sum_{k=1}^{r} L(\delta_k)y_k$, without any reference to the z_k variables.

There is another way to represent L(x) which, however, requires L(x) tc be continuous. Under this assumption, L(x) may be written

$$L(0) + \sum_{k=1}^{r} \alpha_k z_k$$

where now the z_k have the interpretation of equaling the length

of the interval $[\delta_{k-1},\delta_k]$ if $x > \delta_k$, equaling $x - \delta_{k-1}$ (as before) if $x \in [\delta_{k-1},\delta_k]$, and equaling 0 if $x < \delta_{k-1}$. (Thus, $\Sigma z_k = x$.) This interpretation is assured by imposing the constraints

$$y_{k-1}(\delta_k - \delta_{k-1}) \geq z_k \geq y_k(\delta_k - \delta_{k-1})$$
$$y_k \geq 0 \text{ and integer} \quad k = 1, \ldots, r$$

where by convention $y_o = 1$. Then "$y_k = 1$" corresponds to "$x \geq \delta_k$." The constraints imply that $1 \geq y_1 \geq y_2 \cdots \geq y_r$, and that each z_k assumes exactly the value indicated by the foregoing discussion.

This second way of representing L(x) can be slightly modified to accommodate discontinuous functions and functions defined over disjoint intervals by including the 0-1 variables in the expression for L(x).

If the function L(x) that approximates f(x) is convex, and if the problem objective is to minimize f(x), then the quantity $L(0) + \Sigma\alpha_k z_k$ provides an acceptable linear expression for L(x) without introducing 0-1 variables y_k. The only stipulation required is that

$$\delta_k - \delta_{k-1} \geq z_k \geq 0 \qquad k = 1, \ldots, r$$

This follows from the fact that

$$\alpha_1 \geq \alpha_2 \geq \cdots \geq \alpha_k$$

and is implied by the convexity of L(x).

Solution Methods for Integer Programming Problems

The number of solution methods--and special variations for special cases--that have been proposed for integer programming problems are legion. However, nearly all of them can

be described in terms of a single framework. We will indicate this framework, and then provide illustrations of a few of the more popular methods that are special instances of it.

Methods for solving integer programming problems typically proceed by generating a succession of related problems which we will call descendants of the original problem. For each descendant, an associated relaxed problem is identified that is easier to solve than the problem it came from (which will be called its source problem). The solution to the relaxed problem determines the resolution of its source; i.e., whether the source may be discarded or replaced by one or more descendants of its own (hence, which are also descendants of the original problem). Thereupon, one of the descendants is selected which has not yet been discarded or replaced and the process repeats until no more unresolved descendants remain. In a variety of methods (as, for example, cutting methods), the resolution of a problem gives rise to exactly one descendant, though there is typically a choice among several alternatives to provide the identity of this descendant.

We will now characterize the nature of descendants and relaxed problems more precisely.

Descendants

In the same way that a relaxed problem has a source, a descendant problem has a parent (an immediate predecessor from which the descendant "issues"). A collection of immediate descendants of a given problem is required to have the property that at least one of these descendants has the same optimal

solution as its parent. In case the parent problem has multiple optima, the restriction is that some nonempty subset of these optima must constitute the set of optimal solutions for one of its descendants.

Two common examples of descendant problems with these properties will be useful. In the so-called cutting approach, an inequality or equation--called a "cut"--is deduced that is satisfied by all solutions to the parent. (To be useful, the cut usually must not be satisfied by the optimum for the relaxed problem which has been solved in lieu of the parent.) The cut is then appended to the parent to yield a single new problem which has exactly the same set of optimal solutions as the problem from which it descended.

In the branch and bound approach, for the second example, a collection of inequalities (or equations) is deduced, so that each admits some subset of the solutions to the parent problem as feasible, and so that the union of these subsets includes all feasible solutions to the parent. Each of the derived inequalities--or "provisional cuts"--is adjoined to the parent problem separately from the others (since it is not known in advance which one will admit an optimum solution to the parent as feasible), thus creating a collection of descendants having the properties indicated.

In still other approaches (such as certain graph theory methods), nothing is added to the parent problem, but the manner in which the problem is "represented" is modified, based on the solution to the relaxed problem, to produce a single descendant for which the process is repeated.

Relaxed Problems:

The principal characteristic of a relaxed problem is that its constraints are less restrictive--i.e., admit a larger (or no smaller) range of feasible solutions--than the constraints of its source problem. Frequently, a relaxed problem is created simply by discarding some of the constraining conditions of its source. For example, in integer programming, a commonly employed relaxed problem is the ordinary linear programming problem (which arises by dropping the integrality requirements). A relaxed problem may also have a different objective function than its source, provided the optimum objective function value for the relaxed problem does not exceed that for the source (in the minimization context).

These stipulations immediately give rise to the following "double inequality": optimal objective function value for the relaxed problem $\leq$ optimal objective function value for the source problem $\leq$ objective function value for the source problem for any feasible solution to the source.

It follows that, whenever solutions can be found that make the first and the last quantities in this double inequality equal, then the middle quantity is compelled to equal the same value as the other two quantities, and an optimal solution has been found for the source problem.

Duality theories for linear, nonlinear and combinatorial programming involve the specification of a problem which is an instance of a relaxed problem as just defined, and thus which causes the double inequality to hold. The dual problem in

each of these duality theories is to find a "strongest" relaxed problem from among the problems in its category--i.e., a relaxed problem that makes the first quantity in the double inequality the largest.

Only a strongest relaxed problem can possibly force the double inequality to hold as an equality and thereby serve as a tool for identifying an optimal solution to the source problem. (However, even a strongest relaxed problem--from those in the category specified by a particular duality theory--may not be able to force equality, in which case the resulting condition is known as a "duality gap.")

In view of these remarks, the methods that derive from the framework under consideration may properly be called duality exploiting methods. In fact, one of the important features of these methods is that they give a means for solving the source problem even when duality gaps occur. The manner in which they do this relies on the iterative creation of descendants to serve as source problems, and on making use of three useful properties of their associated relaxed problems:

(1) Whenever a relaxed problem lacks a feasible solution, then so does its source;

(2) the optimum objective function value for a relaxed problem provides a lower bound for the optimum objective function value for the source problem; and

(3) if an optimal solution to a relaxed problem yields the same objective function value for the source problem as for the relaxed problem (which it automatically does if the

two objective functions are the same), and if this solution is feasible for the source problem, then it is also optimal for the source problem.

Consequently, whenever conditions (1) or (3) hold, the process of solving the relaxed problem "disposes of" the source problem, thereby accomplishing a hard task (solving the source) by undertaking an easier one (solving the relaxed problem). Moreover, if a ceiling has been established for the optimum objective function value of the source problem, so that the optimum must fall below this ceiling if it is to be acceptable (as commonly occurs in branch and bound methods), then the solution of the relaxed problem also disposes of the source problem under condition (2), provided the lower bound identified by this condition equals or exceeds the imposed ceiling.

Utilizing these observations, we now state the general procedural format for "duality exploiting" methods in integer programming and combinatorial optimization.

Duality Exploiting Methods: General Framework

1. Begin with a "problem list" that contains the original problem as its only member.

2. Select a problem from the list. (If there are no problems on the list at this point, the method stops and the best solution so far found is optimal for the original problem. If no such "candidate" solutions were found, the original problem has no feasible solution.)

3. Solve a relaxed problem--or a collection of relaxed problems--corresponding to the selected problem

--If a relaxed problem lacks a feasible solution (cf. condition (1)), discard the selected problem and return to Step 2.

--If the optimum objective function value for a relaxed problem equals or exceeds the ceiling given by the objective function value for the best candidate solution for the original problem so far found (cf. condition (2)), discard the selected problem and return to Step 2.

--If neither of the preceding situations apply, and if an optimal solution to a relaxed problem is optimal for the selected problem (cf. condition (3)), then check this solution to see if it is feasible for the original problem. (The check is unnecessary for standard branch and bound and cutting approaches--as previously characterized--since for these the set of feasible solutions to each selected problem is a subset of the feasible solutions to the original problem.) If feasibility for the original problem is established, then record this solution as the new best candidate solution for the original problem, and return to Step 2.

--If none of the preceding circumstances apply, treat the selected problem as a parent problem and generate a set of one or more descendants (as by the use of cuts, provisional cuts, an updated problem representation, etc.), so that these descendants have the properties previously stipulated. Add these descendant problems to the list (in place

of their parent) and return to Step 2.

Some comments about the foregoing framework are in order. First, standard techniques for obtaining successively stronger relaxed problems can be used in Step 3. These techniques apply to relaxed problems that are created by replacing some subset of the constraints of the source problem with a nonnegative linear combination of these constraints, where this linear combination is either absorbed into the objective function, as in generalized Lagranzean approaches, or used to form one or more "summarizing" constraints, as in the surrogate constraint apprcaches. (These two types of relaxation approaches will be discussed in more detail later.) All of the techniques for creating successively stronger relaxed problems utilize some variant of the following simple strategy:

(i) solve the current relaxed problem

(ii) if the solution is not feasible for the source problem, identify a subset of the violated constraints and a subset of the "oversatisfied" constraints (so that at least one of these subsets is nonempty) and modify the relaxed problem by increasing the weights of the linear combination associated with the first set and by decreasing the weights associated with the second set. The amount of increase or decrease may vary for each constraint, in a given set, but the net change is made sufficiently great so that the previous solution to the relaxed problem becomes nonoptimal relative to the modified form of the relaxed problem. (The Frank-Wolfe algorithm [19, 43] is a popular technique of this type. In fact, many

of the penalty function methods for nonlinear programming utilize essentially the same strategy, except that the "weights" may apply to functions other than linear ones.)

Another comment is that, as an adjunct to Step 2 or 3, one can optionally employ heuristic approaches to generate "trial solutions" to the selected problem, whereupon these solutions can be tested to see whether they provide improved candidate solutions for the original problem.

Further, in addition to this option, there are three main places in the duality exploiting framework where choice enters: in selecting the next problem from the list; in choosing a relaxed problem (or collection of relaxed problems) for a given source problem; in deciding which cut or which set of provisional cuts, etc., should be generated to produce the new descendant problems. Each of these choice areas is critical to efficient implementation (see [4 , 20 , 22 , 52 , 53 55 , 57]).

Another thing to be noted is that provisional cuts must tend to be a great deal stronger (more restrictive) than ordinary cuts if, as often happens, the branch and bound option is to be preferred to cutting, since the use of provisional cuts leads to two or more descendants of each given problem, and these descendants must somehow be easier to solve than the single descendant of the cutting approach if the total solution time of branch and bound is not to suffer. But there is one advantage to branch and bound that may be extracted even if some of the provisional cuts are rather weak, provided at

least some of them are sufficiently strong. For problems that require excessive amounts of computer time to solve optimally, it may nevertheless be possible that relatively good candidate solutions can be found early in the game using branch and bound, and hence the method may be arbitrarily stopped at a reasonable cut-off point and the best solution found to that time used in lieu of a guaranteed optimum.

Accommodating Primal and Primal-Dual Methods in the Duality Exploiting Framework

A class of methods not subsumed by the preceding framework is the class of "primal" methods. Nevertheless, these methods are easily characterized, and can be incorporated into the foregoing framework as an alternative means of providing trial solutions to the problem selected in Step 2, or as a means of improving a candidate solution generated in Step 3, or even in some cases, as a means of completely solving a relaxed problem or its source.

A method classified as "primal" proceeds by producing a succession of feasible solutions, each better (or no worse) than its predecessor. The "primal strategy" operates by imposing all constraints of the source problem, plus a number of additional constraints which require admissible solutions to be "nearly the same" as the current feasible solution to the source. (In the primal simplex method, for example, the additional constraints compel all current nonbasic variables except one to equal 0, thus allowing a very limited deviation from the solution associated with the current basis. In the

primal maximum flow method, similarly, all flow changes are restricted to occur on an elementary path which currently has a positive "net capacity," again compelling a modified solution to lie close to the current one.) The resulting more highly constrained problem is typically much easier to solve than its source (just as an appropriately generated relaxed problem is much easier to solve than its source), and will always yield a feasible solution at least as good or better than the one currently at hand.

Primal methods that are guaranteed to obtain an optimal solution are much harder to come by in integer programming than in linear programming [44 , 62, 63 , 64] but the "primal strategy" can still be a viable solution technique. In fact, the primal methods for integer programming are actually "primal-dual" methods, insofar as they may be interpreted as combining problem relaxation with problem restriction.

The duality exploiting framework becomes a framework for both primal and primal-dual methods simply by incorporating the primal strategy in Step 3. Moreover, the use of cuts (and provisional cuts) in a primal strategy can be carried out in a manner that assures these cuts are valid for the source problem. Specifically, cuts derived in the process of generating a succession of restricted problems by a primal approach may be inferred relative only to constraints shared in common with the source problem. (This is in fact one way of interpreting the known primal IP methods.) Thus in the general primal-dual framework, the generation of cuts or

provisional cuts to create descendant problems can be guided by an imbedded primal strategy.

Some Examples of Specific Methods

For a clearer understanding of how the foregoing framework can be applied, illustrations will be provided of some of the more popular methods that occur as variants of this framework.

A Cutting Plane Method

For the pure integer programming problem, where all variables are integer-valued, a commonly employed method is that proposed by Gomory [31]. An equation from the Tucker form of a linear programming tableau (see Chapter 1) may be written as

$$x_i = a_{io} + \sum_{j \in N} a_{ij} (-t_j),$$

where x is an integer variable and the t_j, $j \in N$, are the current nonbasic variables, assumed to be constrained to nonnegative integer values. A cut equation implied by this equation is

$$S = -f_{io} - \sum_{j \in N} f_{ij} (-t_j)$$

where S is a nonnegative integer variable, and the constants f_{ij} are the "positive fractional parts" of the constants a_{ij}; that is $f_{ij} = a_{ij} - [a_{ij}]$, where the square brackets represent the largest integer not exceeding the quantity inside. Since x_i does not have to be nonnegative, the cut can be taken from any equation obtained as some integer linear combination of tableau equations.

A standard method for using such cuts results from the general framework by selecting the relaxed problem to be the ordinary LP problem at Step 3. A cut is then obtained from an equation for which a_{io} is not an integer (hence $f_{io} > 0$), thereby producing a single descendant (having the same set of feasible solutions as its parent). For each successive descendant selected at Step 2, the relaxed LP problem is solved in Step 3 by "postoptimizing" with the dual simplex method.

Recently, a number of advances have been made in cutting theory by reference to subadditive functions [11, 34, 35, 46] and specially constructed convex domains [2, 3, 10, 27, 45, 61] Stronger cuts obtained from these advances may permit cutting methods to be applied with greater efficiency than in the past.

A Branch and Bound Method

A simple procedure that has had some success in certain mixed-integer problem applications is the "Dakin branching scheme" [16] of branch and bound, which again takes the relaxed problem of Step 3 to be the ordinary linear program. This approach generates two descendants from a given problem by reference to an integer variable x_i whose current LP solution value, x_i^*, is noninteger. These descendants arise by respectively (i.e., separately) adjoining the two constraints $x_i \leq [x_i^*]$ and $x_i \geq [x_i^*] + 1$ to the parent problem.

More elaborate branch and bound schemes employ "penalty calculations" from relaxed problems that may differ from the LP problem. These penalties yield a lower bound on the amount by which compelling $x_i \leq [x_i^*]$ or $x_i \geq [x_i^*] + 1$ will cause

the optimum objective function value for a descendant problem to differ from that of its parent, thereby allowing some descendants to be discarded immediately, and allowing others to be put "on the bottom" of the problem list as unlikely possibilities for yielding an improved candidate solution.

Some of the relaxed problems from which penalties and other typical uses of relaxed problems derive will now be discussed.

Problem Relaxation

Perhaps the most common generic form of problem relaxation is "generalized Lagrangean relaxation," which incorporates a subset of the problem constraints into the objective function.

Using the notation by which the linear and integer programming problems were earlier defined, the Lagrangean approach associates nonnegative weights u_i with a subset of the problem inequalities (with index set P, say) to create the modified objective function

$$\text{Minimize } \sum_{j=1}^{n} c_j x_j + \sum_{i \in P} u_i \left(\sum_{j=1}^{n} a_{ij} x_j - b_i \right)$$

The constraints thus absorbed into the objective function are "dropped" from the constraint set for the relaxed problem. The updated objective function coefficients of the simplex method for LP problems arise by exactly this technique (here the constraints "taken up" are those corresponding to the nonbasic variables). In fact, most duality theory of mathematical programming is based on this type of relaxation, so it

covers a very wide swath indeed. Variations in the application of this approach are covered in [17, 18, 22, 23, 40, 54].

A second form of problem relaxation that has found considerable application in integer programming is "surrogate constraint relaxation," which replaces subsets of problem constraints by one or more "surrogate constraints." A surrogate constraint likewise may be expressed in terms of nonnegative weights u_i, $i \in P$, used to define the Lagrangean (though the values of these weights that give a strongest relaxation are usually different for the two approaches). The original objective function remains unchanged, but the constraints associated with the index set P are replaced by

$$\sum_{i \in P} u_i \sum_{j=1}^{n} a_{ij} x_j \leq \sum_{i \in P} u_i b_i$$

Although of recent vintage [1, 21, 25] surrogate constraint relaxation in some cases provides stronger penalties than Lagrangean relaxation, and a new mathematical programming duality theory has emerged from this type of relaxation [29, 36, 37] that may lead to increased use. The use of knapsack methods to solve more general problems than knapsack problems occurs primarily in this setting.

The remaining form of relaxation commonly employed is a "group theoretic" relaxation that arises by dropping the nonnegativity conditions on the variables that are basic in an optimal LP solution, but retaining all integer restrictions. This relaxation can actually be viewed as an instance of generalized Lagrangean relaxation, and can be supplemented by further use of this type of relaxation, but is of unique

interest due to the special group theory structure it provides. Studies of this form of relaxation may be found in [33 , 34 , 35 44 , 54].

Finally, a duality that accommodates both generalized Lagrangean and surrogate constraint relaxation--and their composite--in a single framework is now available [29], and combinations of these approaches are beginning to find use in practice [52].

Practical Considerations

Some aspects of solving integer programming problems in practice deserve mention, together with a few accompanying cautions.

First of all, some integer programming problems seem to be inherently difficult to solve, regardless of the method employed. Thus, while simple approaches such as rounding may be clearly futile for such problems, there may also be no other method (currently known) that is capable of producing an optimal or even a feasible integer solution in a reasonable length of time.

Secondly, even for those problems that can be solved relatively efficiently by existing integer programming methods, it is a useful precaution to be skeptical about the presumed optimality of an "optimal" solution. The real world situations which integer programming formulations are sometimes called upon to model can be rather complex, and, the question of imperfect data entirely aside, it is easy to overlook certain aspects of these situations that should be reflected in

the constraints or the objective function. In such instances an optimal integer programming solution may be hazardous to implement unless a number of checks and safeguards are used to make sure that the solution exhibits characteristics appropriate to the situation modeled. Indeed, on the positive side, obtaining absurd or inappropriate optimal solutions is a very useful way to identify unreasonable assumptions and inadequate information that may have gone into the model, thereby permitting an improved model to be developed.

Integer Programs with Special Structures

Some integer programming problems, such as the classical "transportation" and "assignment" problems discussed in many linear programming text books, have the fortunate property that the linear programming solution, or more precisely, every extreme point solution automatically assigns integer values to the variables. Consequently, the standard linear programming methods are entirely sufficient to solve these problems and no special integer programming techniques are required.

There are also other integer programming problems with very special structures, but that do not have the "integer extreme point" property. Most of those discussed earlier in this chapter are of such a type. For these, the linear programming solution generally does not provide an integer solution, and other approaches must be sought. Precisely what approach should be used to solve a specially structured problem? This question is a very difficult one, and a great deal less is known about matching integer programming methods to problem

structures than one would prefer. Nevertheless, ignorance is not total on this issue, and guidelines are in the process of being established [22 , 24 , 48 , 52 , 55 , 59]. These guidelines must be taken cautiously, however, for approaches that initially seem ineffective in certain contexts are sometimes found to be far more effective when slightly modified or when implemented in a slightly different way. But the question of matching methods to problems also has other facets. Some problems have more than one integer programming formulation, and a problem that appears almost wholly intractable under one formulation may be readily solved under another. Recent disclosures in this area can be found in [18 , 24 , 28 , 55 , 59 , 60].

A still more basic issue for some problems is whether they should be formulated as integer programs at all. Virtually every mathematical programming problem whose functions map into the field of real numbers can be approximated to an arbitrary degree of accuracy (if not precisely) by a corresponding integer programming problem. However, it is a pertinent question whether such "artificial" integer programs are worth the bother to formulate, or whether it would be better to tackle these problems in their "natural" form with an appropriately designed solution technique. To make matters murkier, the question of what constitutes a "natural" or an "artificial" form for a problem is itself a rather difficult one. Part of the issue is clearly empirical; a new algorithm may change an "artificial" formulation into a "natural" one by readily

solving problems posed in that formulation. Important advances in this area are provided by the special solution methods developed for "latice point" and "disjunctive" problems [13 , 47 , 48].

On the other hand, it is also true that part of the question of whether an integer programming formulation is natural or not has to do with the issue of sheer size. A variety of nonlinear programs and combinatorial optimization problems tend to "blow-up" when formulated as integer programs. A problem that appears to involve a modest number of parameters and restrictions in a nonlinear or combinatorial formulation may easily turn out to have a staggering number of variables and constraints in an integer programming guise. Another caution at this point: it's not entirely clear that size should be the bugbear it's sometimes taken to be. For, frequently accompanying the problem "blow up" is a corresponding reduction in its "density." That is, the problem matrices corresponding to these large integer programs are usually exceedingly sparse, containing only a small number of nonzero elements. The development of special computer techniques for handling sparse matrices has been very active in recent years and may conceivably revise some of our opinions about the formidability of certain classes of "large" integer programs in the near future. This is particularly true for problems involving imbedded networks, due to substantial recent advances in handling network structures [14 , 30].

On the other hand, special problem structures often suggest special approaches for accommodating them. It is a truism that there nearly always exists a special method that is more efficient for such problems than a general method. (Again, applications involving imbedded networks offer a case in point.) Indeed, the general method itself, adapted and tailored to the special structure, is usually an example of a "more efficient special method." It frequently occurs that special methods devised for combinatorial (and even some non-linear) problems are in fact refinements of more general integer programming methods.

Finally, a truly significant area that is forever being rediscovered, fleetingly heralded for its importance, and then somehow submerged in the rush to devise more foolproof and rigorously founded approaches, is that of heuristics. In the present context, heuristics refer to intelligent schemes for obtaining "good" solutions. Such schemes typically provide no assurance of obtaining optimal solutions, or even ultimately of obtaining any solution. A systematic cataloging of the features of good heuristic methods in integer programming and combinatorics is not an easy thing, though a few attempts have been made [8 , 15 , 26 , 41].

In the real world, the mathematical guarantee of "convergence to an optimum in a finite number of steps" can amount to a warranty that a solution will be obtained one day before doomsday (optimistically speaking, in the case of some integer programming problems). Consequently, the usefulness of a

particular algorithm can well be said to depend on its "heuristic content." In this view, which gradually seems to be gaining in favor, the active pursuit of heuristic principles may hold promise of unlocking doors that presently remain closed, leading to the efficient implementation of integer programming in what are presently "hard" problem areas.

References

1. Balas, Egon, "Discrete Programming by the Filter Method," Operations Research, 19-5, 915-957, September-October 1967.

2. Balas, E., "The Intersection Cut--A New Cutting Plane for Integer Programming," Operations Research, 19, 19-39, 1970.

3. Balas, E., "Integer Programming and Convex Analysis: Intersection Cuts from Outer Polars," Mathematical Programming, 2-3, 1972.

4. Balas, Egon, "On the Use of Intersection Cuts in Branch and Bound," paper presented at the 8th International Symposium on Mathematical Programming, Stanford, August 1973.

Beale, E. M. L., "Sparseness in Linear Programming," in Large Sparse Sets of Linear Equations, ed. J. K. Reid, Academic Press, London, 1971, pp. 1-15.

Benders, J. F., "Partitioning Procedures for Solving Mixed-Variables Programming Problems," Numerische Mathematik, 4, 238-252, 1962.

7. Bowman, V. J., and G. L. Nemhauser, "Deep Cuts in Integer Programming," Opsearch, 8, 89-111, 1971.

8. Bradley, G. H., "Heuristic Solution Methods and Transformed Integer Linear Programming Problems," Yale Report No. 43, March 1971.

Bradley, G. H., and L. A. Wolsey, "Coefficient Reduction for Inequalities in 0-1 Variables," Report CORR 73-6, University of Waterloo, Ontario, Canada, March 1973.

10. Burdet, C. A., "Polaroids: A New Tool in Non-convex and in Integer Programming," W. P. 78-72-1; Graduate School of Industrial Administration, Carnegie-Mellon University, submitted to Naval Research Logistics Quarterly, 1972.

11. Burdet, C. A., and E. L. Johnson, "A Subadditive Approach to the Group Problem of Integer Programming," released by the Mathematical Sciences Department, IBM Watson Research Center, Yorktown Heights, New York, September 1973.

12. Cabot, V. A., "An Enumeration Algorithm for Knapsack Problems," Opns. Res., 18, 306-311, 1970.

13. Cabot, V. A., and A. P. Hurter, "An Approach to 0-1 Integer Programming," Opns. Res., 16, 1206-1211, 1968.

14. Charnes, A., F. Glover, D. Karney, D. Klingman and J. Stutz, "Past, Present and Future of Development, Computational Efficiency, and Practical Use of Large-Scale Transportation and Transhipment Computer Codes," Research Report C.S. 131, Center for Cybernetic Studies, University of Texas, Austin, July 1973.

15. Cooper, L., and C. Drebes, "Investigations in Integer Linear Programming by Direct Search Methods," Report No. COO-1493-10, Washington University, 1967.

16. Dakin, R. J., "A Tree Search Algorithm for Mixed Integer Programming Problems," Computer Journal, 8-3, 250-255, 1965.

17. Everett, H., "Generalized Lagrange Multiplier Method for Solving Problems of Optimum Allocation of Resources," Operations Research, 11, 399-417, 1963.

18. Fisher, M. L., and J. F. Shapiro, "Constructive Duality for Discrete Optimization," CMBSE Report 7321, Graduate School of Business, University of Chicago, April 1973.

19. Frank, M., and P. Wolfe, "An Algorithm for Quadratic Programming," Naval Res. Log. Quart., 3, 95-110, 1956.

20. Garfinkel, R. S., and G. L. Nemhauser, Integer Programming, John Wiley and Sons, Inc., New York, 1972.

21. Geoffrion, A. M., "An Improved Implicit Enumeration Approach for Integer Programming," Operations Research, 17-3, 437-454, May-June 1969.

22. Geoffrion, A. M., "Lagrangean Relaxation for Integer Programming," Working Paper No. 195, Western Management Science Institute, UCLA, revised December 1973.

23. Geoffrion, A. M., "Duality in Nonlinear Programming," SIAM Review, 13-1, 1-37, January 1971.

24. Geoffrion, A. M., and R. E. Marsten, "Integer Programming Algorithms: A Framework and State-of-the-Art Survey," Management Science, 18-9, 465-491, May 1972.

25. Glover, F., "Surrogate Constraints," Operations Research, 16-4, 741-749, July-August 1968.

26. Glover, Fred, "Heuristics in Integer Programming," University of Texas, August 1967.

27. Glover, Fred, "Convexity Cuts and Cut Search," December 1969, published in Operations Research, 21-1, 123-134, January-February 1973.

28. Glover, F., "Improved Linear Representations of Discrete Mathematical Programs," Management Science Report No. 72-8, School of Business, University of Colorado, May 1972.

29. Glover, F., "Surrogate Constraint Duality in Mathematical Programming," MSRS 73-5, University of Colorado, May 1973.

30. Glover, F., D. Karney, D. Klingman, and A. Napier, "A Computation Study on Start Procedures, Basis Change Criteria, and Solution Algorithms for Transportation Problems," Management Science, 20-5, 793-813, January 1974.

31. Gomory, R. E., "An Algorithm for Integer Solutions for Linear Programs," in Recent Advances in Mathematical Programming, ed. Graves and Wolfe, 1963, pp. 269-302.

32. Gomory, R. E., "An Algorithm for the Mixed Integer Problem," RM-2597, RAND Corporation, 1960.

33. Gomory, R. E., "Faces of an Integer Polyhedron," Proceedings of the National Academy of Sciences, 57, 16-18, 1967.

34. Gomory, R. E., and E. L. Johnson, "Some Continuous Functions Related to Corner Polyhedra," Mathematical Programming, 3, 23-85, 1972.

35. Gomory, R. E., and E. L. Johnson, "Some Continuous Functions Related to Corner Polyhedra, II," Mathematical Programming, 3, 359-389, 1972.

36. Greenberg, Harvey J., "The Generalized Penalty Function Surrogate Model," Operations Research, 21-1, January-February 1973.

37. Greenberg, Harvey J., and W. P. Pierskalla, "Surrogate Mathematical Programs," Operations Research, 18, 924-939, 1970.

38. Hammer, P. L., and S. Rudeanu, "Boolean Methods in Operations Research and Related Areas," Springer-Verlag, Berlin-Heidelberg-New York, 1968.

39. Hammer, P. L., E. L. Johnson, and U. N. Peled, "Facets of Regular 0-1 Polytopes," University of Waterloo, Combinatorics and Optimization, Research Report No. CORR 73-19, September 1973.

40. Held, M., and R. M, Karp, "The Traveling Salesman Problem and Minimum Spanning Trees," Operations Research, 18-6, 1138-1162, November-December 1970.

41. Hillier, F. S., "Efficient Heuristic Procedures for Integer Linear Programming with an Interior," Opns. Res., 17, 600-637, 1969.

42. Hillier, F. S., "A Bound-and-Scan Algorithm for Pure Integer Linear Programming with General Variables," Opns, Res., 17, 638-679, 1969.

43. Hogan, W. W., "Convergence Results for Some Extensions of the Frank-Wolfe Method," Working Paper No. 169, Western Management Science Institute, University of California, Los Angeles, January 1971.

44. Hu, T. C., Integer Programming and Network Flows, Addison-Wesley Publishing Company, Menlo Park, California, 1969. 432+ pp.

45. Jeroslow, R. G., "The Either-Or Cut from Convex Domains," Carnegie-Mellon University, September 1973.

46. Jeroslow, R. G., "The Principles of Cutting-Plane Theory: Part I," preliminary report, Carnegie-Mellon University, February 1974.

47. Klingman, D., and F. Glover, "The Generalized Lattice Point Problem," Operations Research, 21, 141-156, 1973.

48. Klingman, D., F. Glover, and J. Stutz, "The Disjunctive Facet Problem: Formulation and Solution Techniques," MSRS 72-10, University of Colorado, June 1972, to appear in Management Science.

49. McDaniel, D., and M. Devine, "Alternative Benders-Based Partitioning Procedures for Mixed Integer Programming," School of INdustrial Engineering, University of Oklahoma, May 1973.

50. Lemke, C. E., and K. Spielberg, "Direct Search Zero-One and Mixed Integer Programming," Opns. Res., 15, 892-914, 1967.

51. Padberg, Manfred W., "Equivalent Knapsack-type Formulations of Bounded Integer Linear Programs," Management Sciences Research Report No. 227, Carnegie-Mellon University, September 1970.

52. Ross, T., and F. Glover, "Strong Penalty Calculations for a Class of 0-1 Programs," MSRS 74-5, University of Colorado, April 1974.