Integer programming

Technology

Integer Programming 9 The linear-programming models that have been discussed thus far all have been continuous, in the sense that decision variables are allowed to be fractional. Often this is a realistic assumption. For instance, we might 3 easily produce 102 4 gallons of a divisible good such as wine. It also might be reasonable to accept a solution 1 giving an hourly production of automobiles at 58 2 if the model were based upon average hourly production, and the production had the interpretation of production rates. At other times, however, fractional solutions are not realistic, and we must consider the optimization problem: n

Maximize j= 1 cjxj, subject to: n j= 1 ai j x j = bi xj ? 0 x j integer (i = 1, 2, . . . , m), (j = 1, 2, . . . , n), (for some or all j = 1, 2, . . . , n). This problem is called the (linear) integer-programming problem. It is said to be a mixed integer program when some, but not all, variables are restricted to be integer, and is called a pure integer program when all decision variables must be integers. As we saw in the preceding chapter, if the constraints are of a network nature, then an integer solution can be obtained by ignoring the integrality restrictions and solving the resulting linear program.

In general, though, variables will be fractional in the linear-programming solution, and further measures must be taken to determine the integer-programming solution. The purpose of this chapter is twofold. First, we will discuss integer-programming formulations. This should provide insight into the scope of integer-programming applications and give some indication of why many practitioners feel that the integer-programming model is one of the most important models in managementscience. Second, we consider basic approaches that have been developed for solving integer and mixed-

integer programming problems. . 1 SOME INTEGER-PROGRAMMING MODELS Integer-programming models arise in practically every area of application of mathematical programming. To develop a preliminary appreciation for the importance of these models, we introduce, in this section, three areas where integer programming has played an important role in supporting managerial decisions. We do not provide the most intricate available formulations in each case, but rather give basic models and suggest possible extensions. 272 9. 1 Some Integer-Programming Models 273

Capital Budgeting In a typical capital-budgeting problem, decisions involve the selection of a number of potential investments. The investment decisions might be to choose among possible plant locations, to select a con? guration of capital equipment, or to settle upon a set of research-and-development projects. Often it makes no sense to consider partial investments in these activities, and so the problem becomes a go–no-go integer program, where the decision variables are taken to be x j = 0 or 1, indicating that the jth investment is rejected or accepted.

Assuming that c j is the contribution resulting from the jth investment and that ai j is the amount of resource i, such as cash or manpower, used on the jth investment, we can state the problem formally as: n Maximize j= 1 cjxj, subject to: n j= 1 ai j x j ? bi xj = 0 or (i = 1, 2, . . . , m), 1 (j = 1, 2, . . . , n). The objective is to maximize total contribution from all investments without exceeding the limited availability bi of any resource. One important special scenario for the capital-budgeting problem involves cash-? ow constraints.

In this case, the constraints n j= 1 ai j xi ? bi re? ect incremental cash balance in each period. The coef? cients ai j represent the net cash ? ow from investment j in period i. If the investment requires additional cash in period i, then ai j > 0, while if the investment generates cash in period i, then ai j < 0. The righthand-side coef? cients bi represent the incremental exogenous cash ? ows. If additional funds are made available in period i, then bi > 0, while if funds are withdrawn in period i, then bi < 0.

These constraints state that the funds required for investment must be less than or equal to the funds generated from prior investments plus exogenous funds made available (or minus exogenous funds withdrawn). The capital-budgeting model can be made much richer by including logical considerations. Suppose, for example, that investment in a new product line is contingent upon previous investment in a new plant. This contingency is modeled simply by the constraint x j ? xi , which states that if xi = 1 and project i (new product development) is accepted, then necessarily x j = 1 and project j (construction of a new plant) must be accepted.

Another example of this nature concerns con? icting projects. The constraint x1 + x2 + x3 + x4 ? 1, for example, states that only one of the ? rst four investments can be accepted. Constraints like this commonly are called multiple-choice constraints. By combining these logical constraints, the model can incorporate many complex interactions between projects, in addition to issues of resource allocation. The simplest of all capital-budgeting models has just one resource constraint, but has attracted much attention in the management-science literature. It is stated as: n Maximize j= 1 jxj, 274 Integer Programming 9. 1 subject to: n j= 1 a j x j ? b, xj = 0 or 1 (j = 1, 2, . .

. , n). Usually, this problem is called the 0–1 knapsack problem, since it is analogous to a situation in which a hiker must decide which goods to include on his trip. Here c j is the ‘‘ value’’ or utility of including good j, which weighs a j > 0 pounds; the objective is to maximize the ‘‘ pleasure of the trip,’’ subject to the weight limitation that the hiker can carry no more than b pounds. The model is altered somewhat by allowing more than one unit of any good to be taken, by writing x j ? and x j -integer in place of the 0–1 restrictions on the variables. The knapsack model is important because a number of integer programs can be shown to be equivalent to it, and further, because solution procedures for knapsack models have motivated procedures for solving general integer programs. Warehouse Location In modeling distribution systems, decisions must be made about tradeoffs between transportation costs and costs for operating distribution centers. As an example, suppose that a manager must decide which of n warehouses to use for meeting the demands of m customers for a good.

The decisions to be made are which warehouses to operate and how much to ship from any warehouse to any customer. Let yi = xi j 1 if warehouse i is opened, 0 if warehouse i is not opened; = Amount to be sent from warehouse i to customer j. The relevant costs are: f i = Fixed operating cost for warehouse i, ifopened (for example, a cost to lease the warehouse), ci j = Per-unit operating cost at warehouse i plus the transportation cost for shipping from warehouse i to customer j. There are two types of constraints for the model: i) the demand d j of each customer must be ? led from the warehouses; and ii) goods can be shipped from a warehouse only if it is opened. The model is: m n m Minimize i= 1 j= 1 ci j xi j + f i yi , i= 1 (1)

subject to: m xi j i= 1 n j= 1 = dj ? n j= 1 (j = 1, 2, . . . , n), (i = 1, 2, . . . , m), (i = 1, 2, . . . , m; j = 1, 2, . . . , n), (i = 1, 2, . . . , m). (2) xi j ? yi ? xi j ? 0 yi = 0 dj? ? 0 or 1 ? (3) 9. 1 Some Integer-Programming Models 275 The objective function incorporates transportation and variable warehousing costs, in addition to ? xed costs for operating warehouses.

The constraints (2) indicate that each customer’s demand must be met. The summation over the shipment variables xi j in the ith constraint of (3) is the amount of the good shipped from warehouse i. When the warehouse is not opened, yi = 0 and the constraint speci? es that nothing can be shipped from the warehouse. On the other hand, when the warehouse is opened and yi = 1, the constraint simply states that the amount to be shipped from warehouse i can be no larger than the total demand, which is always true. Consequently, constraints (3) imply restriction (ii) as proposed above.

Although oversimpli? ed, this model forms the core for sophisticated and realistic distribution models incorporating such features as: 1. multi-echelon distribution systems from plant to warehouse to customer; 2. capacity constraints on both plant production and warehouse throughput; 3. economies of scale in transportation and operating costs; 4. service considerations such as maximum distribution time from warehouses to customers; 5. multiple products; or 6. conditions preventing splitting of orders (in the model above, the demand for any customer can be supplied from several warehouses).

These features can be included in the model by changing it in several ways. For example, warehouse capacities are incorporated by replacing the term

involving yi in constraint (3) with yi K i , where K i is the throughput capacity of warehouse i; multi-echelon distribution may require triple-subscripted variables xi jk denoting the amount to be shipped, from plant i to customer k through warehouse j. Further examples of how the simple warehousing model described here can be modi? ed to incorporate the remaining features mentioned in this list are given in the exercises at the end of the chapter.

Scheduling The entire class of problems referred to as sequencing, scheduling, and routing are inherently integer programs. Consider, for example, the scheduling of students, faculty, and classrooms in such a way that the number of students who cannot take their ? rst choice of classes is minimized. There are constraints on the number and size of classrooms available at any one time, the availability of faculty members at particular times, and the preferences of the students for particular schedules. Clearly, then, the ith student is scheduled for the jth class during the nth time period or not; hence, such a variable is either zero or one.

Other examples of this class of problems include line-balancing, critical-path scheduling with resource constraints, and vehicle dispatching. As a speci? c example, consider the scheduling of airline ? ight personnel. The airline has a number of routing ‘‘ legs’’ to be ? own, such as 10 A. M. New York to Chicago, or 6 P. M. Chicago to Los Angeles. The airline must schedule its personnel crews on routes to cover these ? ights. One crew, for example, might be scheduled to ? y a route containing the two legs just mentioned.

The decision variables, then, specify the scheduling of the crews to routes: 1 if a crew is assigned to route j, xj = 0 otherwise. Let 1 if leg i is included on

route j, ai j = 0 otherwise, and The coef? cients ai j de? ne the acceptable combinations of legs and routes, taking into account such characteristics as sequencing of legs for making connections between ? ights and for including in the routes ground time for maintenance. The model becomes: n c j = Cost for assigning a crew to route j. Minimize j= 1 cjxj, 276 Integer Programming 9. 1 subject to: n j= 1 ai j x j = 1 xj = 0 or 1 (i = 1, 2, . . , m), (j = 1, 2, . . . , n). (4) The ith constraint requires that one crew must be assigned on a route to ? y leg i. An alternative formulation permits a crew to ride as passengers on a leg. Then the constraints (4) become: n j= 1 ai j x j ? 1 n j= 1 (i = 1, 2, . . . , m). (5) If, for example, a1 j x j = 3, then two crews ? y as passengers on leg 1, possibly to make connections to other legs to which they have been assigned for duty. These airline-crew scheduling models arise in many other settings, such as vehicle delivery problems, political districting, and computer data processing.

Often model (4) is called a set-partitioning problem, since the set of legs will be divided, or partitioned, among the various crews. With constraints (5), it is called a set-covering problem, since the crews then will cover the set of legs. Another scheduling example is the so-called traveling salesman problem. Starting from his home, a salesman wishes to visit each of (n ? 1) other cities and return home at minimal cost. He must visit each city exactly once and it costs ci j to travel from city i to city j. What route should he select? If we let xi j = n n 1 0 if he goes from city i to city j, otherwise, e may be tempted to formulate his problem as the assignment problem: Minimize i= 1 j= 1 ci j xi j , subject to: n i= 1 n j= 1 xi j = 1 xi j = 1 xi j ? 0 (j = 1, 2, . . . , n), (i = 1, 2, . . . , n), (i = 1, 2, . . . , n; j = 1, 2, . . . , n). The constraints require that the

salesman must enter and leave each city exactly once. Unfortunately, the assignment model can lead to infeasible solutions. It is possible in a six-city problem, for example, for the assignment solution to route the salesman through two disjoint subtours of the cities instead of on a single trip or tour. See Fig. 9. 1.) Consequently, additional constraints must be included in order to eliminate subtour solutions. There are a number of ways to accomplish this. In this example, we can avoid the subtour solution of Fig. 9. 1 by including the constraint: x14 + x15 + x16 + x24 + x25 + x26 + x34 + x35 + x36 ? 1. 9. 2 Formulating Integer Programs 277 Figure 9. 1 Disjoint subtours. This inequality ensures that at least one leg of the tour connects cities 1, 2, and 3 with cities 4, 5, and 6.

In general, if a constraint of this form is included for each way in which the cities can be divided into two groups, then subtours will be eliminated. The problem with this and related approaches is that, with n cities, (2n ? 1) constraints of this nature must be added, so that the formulation becomes a very large integer-programming problem. For this reason the traveling salesman problem generally is regarded as dif? cult when there are many cities. The traveling salesman model is used as a central component of many vehicular routing and scheduling models. It also arises in production scheduling.

For example, suppose that we wish to sequence (n ? 1) jobs on a single machine, and that ci j is the cost for setting up the machine for job j, given that job i has just been completed. What scheduling sequence for the jobs gives the lowest total setup costs? The problem can be interpreted as a traveling salesman problem, in which the ‘‘ salesman’’ corresponds to the

machine which must ‘‘ visit’’ or perform each of the jobs. ‘‘ Home’’ is the initial setup of the machine, and, in some applications, the machine will have to be returned to this initial setup after completing all of the jobs.

That is, the ‘‘ salesman’’ must return to ‘‘ home’’ after visiting the ‘‘ cities. ’’ 9. 2 FORMULATING INTEGER PROGRAMS The illustrations in the previous section not only have indicated speci? c integer-programming applications, but also have suggested how integer variables can be used to provide broad modeling capabilities beyond those available in linear programming. In many applications, integrality restrictions re? ect natural indivisibilities of the problem under study. For example, when deciding how many nuclear aircraft carriers to have in the U. S.

Navy, fractional solutions clearly are meaningless, since the optimal number is on the order of one or two. In these situations, the decision variables are inherently integral by the nature of the decision-making problem. This is not necessarily the case in every integer-programming application, as illustrated by the capitalbudgeting and the warehouse-location models from the last section. In these models, integer variables arise from (i) logical conditions, such as if a new product is developed, then a new plant must be constructed, and from (ii) non-linearities such as ? ed costs for opening a warehouse. Considerations of this nature are so important for modeling that we devote this section to analyzing and consolidating speci? c integerprogramming formulation techniques, which can be used as tools for a broad range of applications. Binary (0–1) Variables Suppose that we are to determine whether or not to engage in the following activities: (i) to build a new plant, (ii) to undertake an advertising campaign, or (iii) to develop a new

product. In each case, we must make a yes–no or so-called go–no–go decision.

These choices are modeled easily by letting x j = 1 if we engage in the jth activity and x j = 0 otherwise. Variables that are restricted to 0 or 1 in this way are termed binary, bivalent, logical, or 0–1 variables. Binary variables are of great importance because they occur regularly in many model formulations, particularly in problems addressing long-range and high-cost strategic decisions associated with capital-investment planning. If, further, management had decided that at most one of the above three activities can be pursued, the 78 Integer Programming 9. 2 following constraint is appropriate: 3 j= 1 x j ? 1. As we have indicated in the capital-budgeting example in the previous section, this restriction usually is referred to as a multiple-choice constraint, since it limits our choice of investments to be at most one of the three available alternatives. Binary variables are useful whenever variables can assume one of two values, as in batch processing. For example, suppose that a drug manufacturer must decide whether or not to use a fermentation tank.

If he uses the tank, the processingtechnologyrequires that he make B units. Thus, his production y must be 0 or B, and the problem can be modeled with the binary variable x j = 0 or 1 by substituting Bx j for y everywhere in the model. Logical Constraints Frequently, problem settings impose logical constraints on the decision variables (like timing restrictions, contingencies, or con? icting alternatives), which lend themselves to integer-programming formulations. The following discussion reviews the most important instances of these logical relationships. Constraint Feasibility

Possibly the simplest logical question that can be asked in mathematical programming is whether a given choice of the decision variables satis? es a constraint. More precisely, when is the general constraint f (x1 , x2 , . . . , xn) ? b (6) satis? ed? We introduce a binary variable y with the interpretation: y= and write 0 1 if the constraint is known to be satis? ed, otherwise, f (x1 , x2 , . . . , xn) ? By ? b, (7) where the constant B is chosen to be large enough so that the constraint always is satis? ed if y = 1; that is, for every possible choice of the decision variables x1 , x2 , . . . xn at our disposal. Whenever y = 0 gives a feasible solution to constraint (7), we know that constraint (6) must be satis? ed. In practice, it is usually very easy to determine a large number to serve as B, although generally it is best to use the smallest possible value of B in order to avoid numerical dif? culties during computations. Alternative Constraints f (x1 , x2 , . . . , xn) ? b + B, Consider a situation with the alternative constraints: f 1 (x1 , x2 , . . . , xn) ? b1 , f 2 (x1 , x2 , . . . , xn) ? b2 . At least one, but not necessarily both, of these constraints must be satis? d. This restriction can be modeled by combining the technique just introduced with a multiple-choice constraint as follows: f 1 (x1 , x2 , . . . , xn) ? B1 y1 ? b1 , f 2 (x1 , x2 , . . . , xn) ? B2 y2 ? b2 , y1 + y2 ? 1, y1 , y2 binary. 9. 2 Formulating Integer Programs 279 The variables y1 and y2 and constants B1 and B2 are chosen as above to indicate when the constraints are satis? ed. The multiple-choice constraint y1 + y2 ? 1 implies that at least one variable y j equals 0, so that, as required, at least one constraint must be satis? ed.

We can save one integer variable in this formulation by noting that the multiple-choice constraint can be replaced by y1 + y2 = 1, or y2 = 1 ? y1 ,

since this constraint also implies that either y1 or y2 equals 0. The resulting formulation is given by: f 1 (x1 , x2 , . . . , xn) ? B1 y1 ? b1 , f 2 (x1 , x2 , . . . , xn) ? B2 (1 ? y1) ? b2 , y1 = 0 or 1. As an illustration of this technique, consider again the custom-molder example from Chapter 1. That example included the constraint 6x1 + 5x2 ? 60, (8) which represented the production capacity for producing x1 hundred cases of six-ounce glasses and x2 hundred cases of ten-ounce glasses.

Suppose that there were an alternative production process that could be used, having the capacity constraint 4x1 + 5x2 ? 50. (9) Then the decision variables x1 and x2 must satisfy either (8) or (9), depending upon which production process is selected. The integer-programming formulation replaces (8) and (9) with the constraints: 6x1 + 5x2 ? 100y ? 60, 4x1 + 5x2 ? 100(1 ? y) ? 50, y= 0 or 1. In this case, both B1 and B2 are set to 100, which is large enough so that the constraint is not limiting for the production process not used. Conditional Constraints These constraints have the form: f 1 (x1 , x2 , . . . xn) > b1 implies that f 2 (x1 , x2 , . . . , xn) ? b2 . Since this implication is not satis? ed only when both f 1 (x1 , x2 , . . . , xn) > b1 and f 2 (x1 , x2 , . . . , xn) > b2 , the conditional constraint is logically equivalent to the alternative constraints f 1 (x1 , x2 , . . . , xn) ? b1 and/or f 2 (x1 , x2 , . . . , xn) ? b2 , where at least one must be satis? ed. Hence, this situation can be modeled by alternative constraints as indicated above. k-Fold Alternatives Suppose that we must satisfy at least k of the constraints: f j (x1 , x2 , . . . , xn) ? b j (j = 1, 2, . . . , p).

For example, these restrictions may correspond to manpower constraints for p potential inspection systems for quality control in a production process. If

management has decided to adopt at least k inspection systems, then the k constraints specifying the manpower restrictions for these systems must be satis? ed, and the 280 Integer Programming 9. 2 remaining constraints can be ignored. Assuming that B j for j = 1, 2, . . . , p, are chosen so that the ignored constraints will not be binding, the general problem can be formulated as follows: f j (x1 , x2 , . . . , xn) ? B j (1 ? y j) ? b j p j= 1 (j = 1, 2, . . , p), y j ? k, yj = 0 or 1 (j = 1, 2, . . . , p). That is, y j = 1 if the jth constraint is to be satis? ed, and at least k of the constraints must be satis? ed. If we de? ne y j ? 1 ? y j , and substitute for y j in these constraints, the form of the resulting constraints is analogous to that given previously for modeling alternative constraints. Compound Alternatives The feasible region shown in Fig. 9. 2 consists of three disjoint regions, each speci? ed by a system of inequalities. The feasible region is de? ned by alternative sets of constraints, and can be modeled by the system: f 1 (x1 , x2) ?

B1 y1 ? b1 f 2 (x1 , x2) ? B2 y1 ? b2 Region 1 constraints Region 2 constraints Region 3 constraints ? f 5 (x1 , x2) ? B5 y3 ? b5 ? f 6 (x1 , x2) ? B6 y3 ? b6 ? f 7 (x1 , x2) ? B7 y3 ? b7 y1 + y2 + y3 ? 2, x1 ? 0, x2 ? 0, y1 , y2 , y3 binary. f 3 (x1 , x2) ? B3 y2 ? b3 f 4 (x1 , x2) ? B4 y2 ? b4 Note that we use the same binary variable y j for eachconstraint de? ning one of the regions, and that the Figure 9. 2 An example of compound alternatives. 9. 2 Formulating Integer Programs 281 Figure 9. 3 Geometry of alternative constraints. constraint y1 + y2 + y3 ? implies that the decision variables x1 and x2 lie in at least one of the required regions. Thus, for example, if y3 = 0, then each of the constraints f 5 (x1 , x2) ? b5 , f 6 (x1 , x2) ? b6 , and f 7 (x1 , x2) ? b7 is satis? ed. The regions do not have to be disjoint before we

can apply this technique. Even the simple alternative constraint f 1 (x1 , x2) ? b1 or f 2 (x1 , x2) ? b2 shown in Fig. 9. 3 contains overlapping regions. Representing Nonlinear Functions Nonlinear functions can be represented by integer-programming formulations. Let us analyze the most useful representations of this type.) Fixed Costs Frequently, the objective function for a minimization problem contains ? xed costs (preliminary design costs, ? xed investment costs, ? xed contracts, and so forth). For example, the cost of producing x units of a speci? c product might consist of a ? xed cost of setting up the equipment and a variable cost per unit produced on the equipment. An example of this type of cost is given in Fig. 9. 4. Assume that the equipment has a capacity of B units. De? ne y to be a binary variable that indicates when the ? xed cost is incurred, so that y = 1 when x > 0 and y = 0 when x = 0.

Then the contribution to cost due to x may be written as K y + cx, with the constraints: x ? By, x ? 0, y = 0 or 1. As required, these constraints imply that x = 0 when the ? xed cost is not incurred, i. e. , when y = 0. The constraints themselves do not imply that y = 0 if x = 0. But when x = 0, the minimization will clearly 282 Integer Programming 9. 2 Figure 9. 4 A ? xed cost. Figure 9. 5 Modeling a piecewise linear curve. select y = 0, so that the ? xed cost is not incurred. Finally, observe that if y = 1, then the added constraint becomes x ? B, which re? ects the capacity limit on the production equipment. i) Piecewise Linear Representation Another type of nonlinear function that can be represented by integer variables is a piecewise linear curve. Figure 9. 5 illustrates a cost curve for plant expansion that contains three linear segments with variable costs of 5, 1, and 3 million dollars per

1000 items of expansion. To model the cost curve, we express any value of x as the sum of three variables ? 1 , ? 2 , ? 3 , so that the cost for each of these variables is linear. Hence, x = ? 1 + ? 2 + ? 3 , where 0 ? ? 1 ? 4, 0 ? ? 3 ? 5; and the total variable cost is given by: Cost = 5? 1 + ? 2 + 3? 3 . 0 ? ? 2 ? 6, 10) 9. 2 Formulating Integer Programs 283 Note that we have de? ned the variables so that: ? 1 corresponds to the amount by which x exceeds 0, but is less than or equal to 4; ? 2 is the amount by which x exceeds 4, but is less than or equal to 10; and ? 3 is the amount by which x exceeds 10, but is less than or equal to 15. If this interpretation is to be valid, we must also require that ? 1 = 4 whenever ? 2 > 0 and that ? 2 = 6 whenever ? 3 > 0. Otherwise, when x = 2, say, the cost would be minimized by selecting ? 1 = ? 3 = 0 and ? 2 = 2, since the variable ? 2 has the smallest variable cost.

However, these restrictions on the variables are simply conditional constraints and can be modeled by introducing binary variables, as before. If we let w1 = w2 = 1 0 1 0 if ? 1 is at its upper bound, otherwise, if ? 2 is at its upper bound, otherwise, then constraints (10) can be replaced by 4w1 ? ? 1 ? 4, 6w2 ? ? 2 ? 6w1 , 0 ? ? 3 ? 5w2 , w1 and w2 binary, (11) to ensure that the proper conditional constraints hold. Note that if w1 = 0, then w2 = 0, to maintain feasibility for the constraint imposed upon ? 2 , and (11) reduces to 0 ? ? 1 ? 4, If w1 = 1 and w2 = 0, then (11) reduces to ? 1 = 4, 0 ? ? 2 ? 6, and ? = 0. ? 2 = 0, and ? 3 = 0. Finally, if w1 = 1 and w2 = 1, then (11) reduces to ? 1 = 4, ? 2 = 6, and 0 ? ? 3 ? 5. Hence, we observe that there are three feasible combinations for the values of w1 and w2 : w1 = 0, w1 = 1, and w1 = 1, w2 = 1 corresponding to 10 ? x ? 15 since ? 1 = 4 and ? 2 = 6. The same general technique can be applied to piecewise linear curves with

any number of segments. The general constraint imposed upon the variable ? j for the jth segment will read: L j w j ? ? j ? L j w j? 1 , where L j is the length of the segment. w2 = 0 corresponding to 0 ? x ? 4 w2 = 0 corresponding to 4 ? x ? 0 since ? 2 = ? 3 = 0; since ? 1 = 4 and ? 3 = 0; 284 Integer Programming 9. 3 Figure 9. 6 Diseconomies of scale. iii) Diseconomies of Scale An important special case for representing nonlinear functions arises when only diseconomies of scale apply— that is, when marginal costs are increasing for a minimization problem or marginal returns are decreasing for a maximization problem. Suppose that the expansion cost in the previous example now is speci? ed by Fig. 9. 6. In this case, the cost is represented by Cost = ? 1 + 3? 2 + 6? 3 , subject only to the linear constraints without integer variables, 0 ? 1 ? 4 0 ? ? 2 ? 6, 0 ? ? 3 ? 5. The conditional constraints involving binary variables in the previous formulation can be ignored if the cost curve appears in a minimization objective function, since the coef? cients of ? 1 , ? 2 , and ? 3 imply that it is always best to set ? 1 = 4 before taking ? 2 > 0, and to set ? 2 = 6 before taking ? 3 > 0. As a consequence, the integer variables have been avoided completely. This representation without integer variables is not valid, however, if economies of scale are present; for example, if the function given in Fig. . 6 appears in a maximization problem. In such cases, it would be best to select the third segment with variable ? 3 before taking the ? rst two segments, since the returns are higher on this segment. In this instance, the model requires the binary-variable formulation of the previous section. iv) Approximation of Nonlinear Functions One of the most useful applications of the piecewise linear representation is for approximating nonlinear functions. Suppose, for

example, that the expansion cost in our illustration is given by the heavy curve in Fig. 9. 7.

If we draw linear segments joining selected points on the curve, we obtain a piecewise linear approximation, which can be used instead of the curve in the model. The piecewise approximation, of course, is represented by introducing integer variables as indicated above. By using more points on the curve, we can make the approximation as close as we desire. 9. 3 A Sample Formulation † 285 Figure 9. 7 Approximation of a nonlinear curve. 9. 3 A SAMPLE FORMULATION † Proper placement of service facilities such as schools, hospitals, and recreational areas is essential to ef? cient urban design. Here we will present a simpli? d model for ? rehouse location. Our purpose is to show formulation devices of the previous section arising together in a meaningful context, rather than to give a comprehensive model for the location problem per se. As a consequence, we shall ignore many relevant issues, including uncertainty. Assume that population is concentrated in I districts within the city and that district i contains pi people. Preliminary analysis (land surveys, politics, and so forth) has limited the potential location of ? rehouses to J sites. Let di j ? 0 be the distance from the center of district i to site j.

We are to determine the ‘‘ best’’ site selection and assignment of districts to ? rehouses. Let yj = and xi j = 1 0 if district i is assigned to site j, otherwise. 1 0 if site j is selected, otherwise; The basic constraints are that every district should be assigned to exactly one ? rehouse, that is, J j= 1 xi j = 1 (i = 1, 2, . . . , I), and that no district should be assigned to an unused site, that is, y j = 0 implies xi j = 0 (i = 1, 2, . . . , I). The latter restriction can

be modeled as alternative constraints, or more simply as: I i= 1 xi j ? y j I (j = 1, 2, . . . , J).

Since xi j are binary variables, their sum never exceeds I , so that if y j = 1, then constraint j is nonbinding. If y j = 0, then xi j = 0 for all i. † This section may be omitted without loss of continuity. 286 Integer Programming 9. 3 Next note that di , the distance from district i to its assigned ? rehouse, is given by: J di = since one xi j will be 1 and all others 0. Also, the total population serviced by site j is: di j xi j , j= 1 I sj = pi xi j . i= 1 Assume that a central district is particularly susceptible to ? re and that either sites 1 and 2 or sites 3 and 4 can be used to protect this district.

Then one of a number of similar restrictions might be: y1 + y2 ? 2 or y3 + y4 ? 2. We let y be a binary variable; then these alternative constraints become: y1 + y2 ? 2y, y3 + y4 ? 2(1 ? y). Next assume that it costs f j (s j) to build a ? rehouse at site j to service s j people and that a total budget of B dollars has been allocated for ? rehouse construction. Then J j= 1 f j (s j) ? B. Finally, one possible social-welfare function might be to minimize the distance traveled to the district farthest from its assigned ? rehouse, that is, to: Minimize D, where or, equivalently,‡ to subject to: D ? i (i = 1, 2, . . . , I). J D = max di ; Minimize D, Collecting constraints and substituting above for di in terms of its de? ning relationship di = we set up the full model as: Minimize D, ‡ The inequalities D ? d imply that D ? max d . The minimization of D then ensures that it will actually be the i i maximum of the di . di j xi j , j= 1 9. 4 Some Characteristics Of Integer Programs—A Sample Problem 287 subject to: J D? j= 1 J j= 1 I i= 1 I di j xi j ? 0 xi j = 1 xi j ? y j I pi xi j = 0 f j (s j) ? B, (i = 1, 2, . . . , I), (i = 1, 2, . . . , I), (j = 1, 2, . . . , J), (j = 1, 2, . . . , J),

Sj? i= 1 J j= 1 y1 + y2 ? 2y ? 0, y3 + y4 + 2y ? 2, xi j , y j , y binary (i = 1, 2, . . . , I ; j = 1, 2, . . . , J). At this point we might replace each function f j (s j) by an integer-programming approximation to complete the model. Details are left to the reader. Note that if f j (s j) contains a ? xed cost, then new ? xed-cost variables need not be introduced—the variable y j serves this purpose. The last comment, and the way in which the conditional constraint ‘‘ y j = 0 implies xi j = 0 (i = 1, 2, . . . , I)’’ has been modeled above, indicate that the formulation techniques of Section 9. should not be applied without thought. Rather, they provide a common framework for modeling and should be used in conjunction with good modeling ‘‘ common sense. ’’ In general, it is best to introduce as few integer variables as possible. 9. 4 SOME CHARACTERISTICS OF INTEGER PROGRAMS—A SAMPLE PROBLEM Whereas the simplex method is effective for solving linear programs, there is no single technique for solving integer programs. Instead, a number of procedures have been developed, and the performance of any particular technique appears to be highly problem-dependent.

Methods to date can be classi? ed broadly as following one of three approaches: i) enumeration techniques, including the branch-and-bound procedure; ii) cutting-plane techniques; and iii) group-theoretic techniques. In addition, several composite procedures have been proposed, which combine techniques using several of these approaches. In fact, there is a trend in computer systems for integer programming to include a number of approaches and possibly utilize them all when analyzing a given problem. In the sections to follow, we shall consider the ? rst two approaches in some detail.

At this point, we shall introduce a speci? c problem and indicate some features of integer programs. Later we will use this example to illustrate and motivate the solution procedures. Many characteristics of this example are shared by the integer version of the custommolder problem presented in Chapter 1. The problem is to determine z ? where: z ? = max z = 5x1 + 8x2 , 288 Integer Programming 9. 5 subject to: x1 + x2 ? 6, 5x1 + 9x2 ? 45, x1 , x2 ? 0 and integer. The feasible region is sketched in Fig. 9. 8. Dots in the shaded region are feasible integer points. Figure 9. An integer programming example. If the integrality restrictions on variables are dropped, the resulting problem is a linear program. We will call it the associated linear program. We may easily determine its optimal solution graphically. Table 9. 1 depicts some of the features of the problem. Table 9. 1 Problem features. Continuous optimum x1 x2 z 9 4 = 2. 25 15 = 3. 75 4 Round off 2 4 Infeasible Nearest feasible point 2 3 34 Integer optimum 0 5 40 41. 25 Observe that the optimal integer-programming solution is not obtained by rounding the linear-programming solution.

The closest point to the optimal linear-program solution is not even feasible. Also, note that the nearest feasible integer point to the linear-program solution is far removed from the optimal integer point. Thus, it is not suf? cient simply to round linear-programming solutions. In fact, by scaling the righthand-side and cost coef? cients of this example properly, we can construct a problem for which the optimal integerprogramming solution lies as far as we like from the rounded linear-programming solution, in either z value or distance on the plane.

In an example as simple as this, almost any solution procedure will be effective. For instance, we could easily enumerate all the integer points with x1 ? 9, x2 ? 6, and select the best feasible point. In practice, the number of points to be considered is likely to prohibit such an exhaustive enumeration of potentially feasible points, and a more sophisticated procedure will have to be adopted. 9. 5 Branch-And-Bound 289 Figure 9. 9 Subdividing the feasible region. 9. 5 BRANCH-AND-BOUND Branch-and-bound is essentially a strategy of ‘‘ divide and conquer. ’ The idea is to partition the feasible region into more manageable subdivisions and then, if required, to further partition the subdivisions. In general, there are a number of ways to divide the feasible region, and as a consequence there are a number of branch-and-bound algorithms. We shall consider one such technique, for problems with only binary variables, in Section 9. 7. For historical reasons, the technique that will be described next usually is referred to as the branch-and-bound procedure. Basic Procedure An integer linear program is a linear program further constrained by the integrality restrictions.

Thus, in a maximization problem, the value of the objective function, at the linear-program optimum, will always be an upper bound on the optimal integer-programming objective. In addition, any integer feasible point is always a lower bound on the optimal linear-program objective value. The idea of branch-and-bound is to utilize these observations to systematically subdivide the linearprogramming feasible region and make assessments of the integer-programming problem based upon these subdivisions. The method can be described easily by considering the example from the previous section.

At ? rst, the linear-programming region is not subdivided: The integrality restrictions are dropped and the associated linear program is solved, giving an optimal value z 0 . From our remark above, this gives the upper 1 bound on z ? , z ? ? z 0 = 41 4 . Since the coef? cients in the objective function are integral, z ? must be integral ? ? 41. and this implies that z 1 3 Next note that the linear-programming solution has x1 = 2 4 and x2 = 3 4 . Both of these variables must be integer in the optimal solution, and we can divide the feasible region in an attempt to make either integral.

We know that, in any integer programming solution, x2 must be either an integer ? 3 or an integer ? 4. Thus, our ? rst subdivision is into the regions where x2 ? 3 and x2 ? 4 as displayed by the shaded regions L 1 and L 2 in Fig. 9. 9. Observe that, by making the subdivisions, we have excluded the old linear-program solution. (If we selected x1 instead, the region would be subdivided with x1 ? 2 and x1 ? 3.) The results up to this point are pictured conveniently in an enumeration tree (Fig. 9. 10). Here L 0 represents the associated linear program, whose optimal solution has been included within the L 0 box, and the upper bound on z ? ppears to the right of the box. The boxes below correspond to the new subdivisions; the constraints that subdivide L 0 are included next to the lines joining the boxes. Thus, the constraints of L 1 are those of L 0 together with the constraint x2 ? 4, while the constraints of L 2 are those of L 0 together with the constraint x2 ? 3. The strategy to be pursued now may be apparent: Simply treat each subdivision as we did the original problem. Consider L 1 ? rst. Graphically, from Fig. 9. 9 we see that the optimal linear-programming solution 290 Integer Programming 9. 5 Figure 9. 10 Enumeration tree.

Figure 9. 11 Subdividing the region L 1 . 9 1 lies on the second constraint with x2 = 4, giving x1 = 5 (45 ? 9(4)) = 5 and an objective value z = 5 9 +8(4) = 41. Since x1 is not integer, we subdivide L 1 further, into the regions L 3 with x1 ? 2 and L 4 with 5 x1 ? 1. L 3 is an infeasible problem and so this branch of the enumeration tree no longer needs to be considered. The enumeration tree now becomes that shown in Fig. 9. 12. Note that the constraints of any subdivision are obtained by tracing back to L 0 . For example, L 4 contains the original constraints together with x2 ? 4 and x1 ? . The asterisk (?) below box L 3 indicates that the region need not be subdivided or, equivalently, that the tree will not be extended from this box. At this point, subdivisions L 2 and L 4 must be considered. We may select one arbitrarily; however, in practice, a number of useful heuristics are applied to make this choice. For simplicity, let us select the subdivision most recently generated, here L 4 . Analyzing the region, we ? nd that its optimal solution has x1 = 1, 1 x2 = 9 (45 ? 5) = 40 9. Since x2 is not integer, L 4 must be further subdivided into L 5 with x2 ? 4, and L 6 with x2 ? , leaving L 2 , L 5 and L 6 yet to be considered. Treating L 5 ? rst (see Fig. 9. 13), we see that its optimum has x1 = 1, x2 = 4, and z = 37. Since this is the best linear-programming solution for L 5 and the linear program contains every integer solution in L 5 , no integer point in that subdivision can give a larger objective value than this point. Consequently, other points 9. 5 Branch-And-Bound 291 Figure 9. 12 Figure 9. 13 Final subdivisions for the example. in L 5 need never be considered and L 5 need not be subdivided further. In fact, since x1 = 1, x2 = 4, z = 37, is a feasible solution to the original problem, z ? 37 and we now have the bounds 37 ? z ? ? 41. Without further analysis, we could terminate with the integer solution x1 = 1, x2 = 4, knowing that the

objective value of this point is within 10 percent of the true optimum. For convenience, the lower bound z ? ? 37 just determined has been appended to the right of the L 5 box in the enumeration tree (Fig. 9. 14). Although x1 = 1, x2 = 4 is the best integer point in L 5 , the regions L 2 and L 6 might contain better feasible solutions, and we must continue the procedure by analyzing these regions.

In L 6 , the only feasible point is x1 = 0, x2 = 5, giving an objective value z = +40. This is better than the previous integer point and thus the lower bound on z ? improves, so that 40 ? z ? ? 41. We could terminate with this integer solution knowing that it is within 2. 5 percent of the true optimum. However, L 2 could contain an even better integer solution. The linear-programming solution in L 2 has x1 = x2 = 3 and z = 39. This is the best integer point in L 2 but is not as good as x1 = 0, x2 = 5, so the later point (in L 6) must indeed be optimal.

It is interesting to note that, even if the solution to L 2 did not give x1 and x2 integer, but had z < 40, then no feasible (and, in particular, no integer point) in L 2 could be as good as x1 = 0, x2 = 5, with z = 40. Thus, again x1 = 0, x2 = 5 would be known to be optimal. Thisobservationhas important computational implications, 292 Integer Programming 9. 5 Figure 9. 14 since it is not necessary to drive every branch in the enumeration tree to an integer or infeasible solution, but only to an objective value below the best integer solution.

The problem now is solved and the entire solution procedure can be summarized by the enumeration tree in Fig. 9. 15. Figure 9. 15 Further

Considerations There are three points that have yet to be considered withrespectto the branch-and-bound procedure: i) Can the linear programs corresponding to the subdivisions be solved ef? ciently? ii) What is the best way to subdivide a given region, and which unanalyzed subdivision should be considered next? 9. 5 Branch-And-Bound 293 iii) Can the upper bound (z = 41, in the example) on the optimal value z ? of the integer program be improved while the problem is being solved?

The answer to the ? rst question is an unquali? ed yes. When moving from a region to one of its subdivisions, we add one constraint that is not satis? ed by the optimal linear-programming solution over the parent region. Moreover, this was onemotivationfor the dual simplex algorithm, and it is natural to adopt that algorithm here. Referring to the sample problem will illustrate the method. The ? rst two subdivisions L 1 and L 2 in that example were generated by adding the following constraints to the original problem: For subdivision 1 : For subdivision 2 : x2 ? 4 x2 ? or or x2 ? s3 = 4 x2 + s4 = 3 (s3 ? 0); (s4 ? 0). In either case we add the new constraint to the optimal linear-programming tableau. For subdivision 1, this gives: 3 1? 5 (? z) ? 4 s1 ? 4 s2 = ? 41 4 ? Constraints from the ? 1 9 9 = x1 + 4 s1 ? 4 s2 4 ? optimal canonical 5 1 15 ? form xj? 4 s1 + 4 s2 = 2 4 ? x2 x1 , x2 , s1 , s2 , s3 ? 0, + s3 = ? 4, Added constraint where s1 and s2 are slack variables for the two constraints in the original problem formulation. Note that the new constraint has been multiplied by ? 1, so that the slack variable s3 can be used as a basic variable.

Since the basic variable x2 appears with a nonzero coef? cient in the new constraint, though, we must pivot to isolate this variable in the second

constraint to re-express the system as: (? z) x1 5 3 ? 4 s1 ? 4 s2 1 5 x2 ? 4 s1 + 4 s2 1 9 + 4 s1 ? 4 s2 = = = 1 ? 41 4 , 15 4, 1 ? 4, x1 , x2 , s1 , s2 , s3 ? 0. 5 1 ? 4 s1 + 4 s2 +s3 = 9 4, These constraints are expressed in the proper form for applying the dual simplex algorithm, which will pivot next to make s1 the basic variable in the third constraint. The resulting system is given by: (? z) x1 x2 ? s2 ? s3 + 1 s2 + 9 s3 5 5 ? 3 s1 ? 1 s2 ? 4 s3 5 5 = ? 41, 9 = 5, = 4, 1 = 5, This tableau is optimal and gives the optimal linear-programming solution over the region L 1 as x1 = 9 , x2 = 5 4, and z = 41. The same procedure can be used to determine the optimal solution in L 2 . When the linear-programming problem contains many constraints, this approach for recovering an optimal solution is very effective. After adding a new constraint and making the slack variable for that constraint basic, we always have a starting solution for the dual-simplex algorithm with only one basic variable negative.

Usually, only a few dual-simplex pivoting operations are required to obtain the optimal solution. Using the primal-simplex algorithm generally would require many more computations. x1 , x2 , s1 , s2 , s3 ? 0. 294 Integer Programming 9. 5 Figure 9. 16 Issue (ii) raised above is very important since, if we can make our choice of subdivisions in such a way as to rapidly obtain a good (with luck, near-optimal) integer solution z , then we can eliminate many potential ? subdivisions immediately. Indeed, if any region has its linear programming value z ? , then the objective ? value of no integer point in that region can exceed z and the region need not be subdivided. There is no ? universal method for making the required choice, although several heuristic procedures have been suggested, such as selecting the subdivision

with the largest optimal linear-programming value. † Rules for determining which fractional variables to use in constructing subdivisions are more subtle. Recall that any fractional variable can be used to generate a subdivision.

One procedure utilized is to look ahead one step in the dual-simplex method for every possible subdivision to see which is most promising. The details are somewhat involved and are omitted here. For expository purposes, we have selected the fractional variable arbitrarily. Finally, the upper bound z on the value z ? of the integer program can be improved as we solve the problem. Suppose for example, that subdivision L 2 was analyzed before subdivisions L 5 or L 6 in our sample problem. The enumeration tree would be as shown in Fig. 9. 16. At this point, the optimal solution must lie in either L 2 or L 4 .

Since, however, the largest value for 5 any feasible point in either of these regions is 40 9 , the optimal value for the problem z ? cannot exceed 40 5 . 9 Because z ? must be integral, this implies that z ? ? 40 and the upper bound has been improved from the value 41 provided by the solution to the linear program on L 0 . In general, the upper bound is given in this way as the largest value of any ‘‘ hanging’’ box (one that has not been divided) in the enumeration tree. Summary The essential idea of branch-and-bound is to subdivide the feasible region to develop bounds z < z ? lt; z on z ? . For a maximization problem, the lower bound z is the highest value of any feasible integer point encountered. The upper bound is given by the optimal value of the associated linear program or by the largest value for the objective function at any ‘‘ hanging’’ box. After considering a subdivision, we must branch to (move to) another subdivision and analyze it. Also, if either One

common method used in practice is to consider subdivisions on a last-generated–? rst-analyzed basis. We used this rule in our previous example.

Note that data to initiate the dual-simplex method mentioned above must be stored for each subdivision that has yet to be analyzed. This data usually is stored in a list, with new information being added to the top of the list. When required, data then is extracted from the top of this list, leading to the last-generated–? rst-analyzed rule. Observe that when we subdivide a region into two subdivisions, one of these subdivisions will be analyzed next. The data required for this analysis already will be in the computer core and need not be extracted from the list. † 9. 6 Branch-And-Bound 95 i) the linear program over L j is infeasible; ii) the optimal linear-programming solution over L j is integer; or iii) the value of the linear-programming solution z j over L j satis? es z j ? z (if maximizing), then L j need not be subdivided. In these cases, integer-programming terminology says that L j has been fathomed. † Case (i) is termed fathoming by infeasibility, (ii) fathoming by integrality, and (iii) fathoming by bounds. The ? ow chart in Fig. 9. 17 summarizes the general procedure. Figure 9. 17 Branch-and-bound for integer-programming maximization. † To fathom is de? ed as ‘‘ to get to the bottom of; to understand thoroughly. ’’ In this chapter, fathomed might be more appropriately de? ned as ‘‘ understood enough or already considered. ’’ 296 Integer Programming 9. 7 Figure 9. 18 9. 6 BRANCH-AND-BOUND FOR MIXED-INTEGER PROGRAMS The branch-and-bound approach just described is easily extended to solve problems in which some, but not all, variables are constrained to be integral. Subdivisions then are generated solely by the integral variables. In every other way, the procedure is the same as that

speci? ed above. A brief example will illustrate the method. ? = max z = ? 3x1 ? 2x2 + 10, subject to: x1 ? 2x2+ x3 5 = 2, 2x1 + x2 xj ? 0 3 + x4 = 2 , (j = 1, 2, 3, 4), x2 and x3 integer. The problem, as stated, is in canonical form, with x3 and x4 optimal basic variables for the associated linear program. The continuous variable x4 cannot be used to generate subdivisions since any value of x4 ? 0 potentially can be optimal. Consequently, the subdivisions must be de? ned by x3 ? 2 and x3 ? 3. The complete procedure is summarized by the enumeration tree in Fig. 9. 18. 1 The solution in L 1 satis? es the integrality restrictions, so z ? ? z = 8 2 .

The only integral variable with a fractional value in the optimal solution of L 2 is x2 , so subdivisions L 3 and L 4 are generated from this variable. Finally, the optimal linear-programming value of L 4 is 8, so no feasible mixed-integer solution in that region 1 can be better than the value 8 2 already generated. Consequently, that region need not be subdivided and the solution in L 1 is optimal. The dual-simplex iterations that solve the linear programs in L 1 , L 2 , L 3 , and L 4 are given below in Tableau 1. The variables s j in the tableaus are the slack variables for the constraints added to generate the subdivisions.

The coef? cients in the appended constraints are determined as we mentioned in the last section, by eliminating the basic variables x j from the new constraint that is introduced. To follow the iterations, recall that in the dual-simplex method, pivots are made on negative elements in the generating row; if all elements in this row are positive, as in region L 3 , then the problem is infeasible. 9. 7 Implicit Enumeration 297 9. 7 IMPLICIT

ENUMERATION A special branch-and-bound procedure can be given for integer programs with only binary variables.

The algorithm has the advantage that it requires no linear-programming solutions. It is illustrated by the following example: z ? = max z = ? 8x1 ? 2x2 ? 4x3 ? 7x4 ? 5x5 + 10, subject to: ? 3x1 ? 3x2 + x3 + 2x4 + 3x5 ? ? 2, ? 5x1 ? 3x2 ? 2x3 ? x4 + x5 ? ? 4, xj = 0 or 1 (j = 1, 2, . . . , 5). One way to solve such problems is complete enumeration. List all possible binary combinations of the variables and select the best such point that is feasible. The approach works very well on a small problem such as this, where there are only a few potential 0–1 combinations for the variables, here 32.

In general, though, an n-variable problem contains 2n 0–1 combinations; for large values of n, the exhaustive approach is prohibitive. Instead, one might implicitly consider every binary combination, just as every integer point was implicitly considered, but not necessarily evaluated, for the general problem via branch-and-bound. Recall that in the ordinary branch-and-bound procedure, subdivisions were analyzed by maintaining the linear constraints and dropping the integrality restrictions. Here, we adopt the opposite tactic of always 298 Integer Programming 9. 7 maintaining the 0–1 restrictions, but ignoring the linear inequalities.

The idea is to utilize a branch-and-bound (or subdivision) process to ? x some of the variables at 0 or 1. The variables remaining to be speci? ed are called free variables. Note that, if the inequality constraints are ignored, the objective function is maximized by setting the free variables to zero, since their objectivefunction coef? cients are negative. For example, if x1 and x4

are ? xed at 1 and x5 at 0, then the free variables are x2 and x3 . Ignoring the inequality constraints, the resulting problem is: max [? 8(1) ? 2x2 ? 4x3 ? 7(1) ? 5(0) + 10] = max [? 2x2 ? 4x3 ? 5], subject to: x2 and x3 binary.

Since the free variables have negative objective-function coef? cients, the maximization sets x2 = x3 = 0. The simplicity of this trivial optimization, as compared to a more formidable linear program, is what we would like to exploit. Returning to the example, we start with no ? xed variables, and consequently every variable is free and set to zero. The solution does not satisfy the inequality constraints, and we must subdivide to search for feasible solutions. One subdivision choice might be: For subdivision 1 : x1 = 1, For subdivision 2 : x1 = 0. Now variable x1 is ? xed in each subdivision.

By our observations above, if the inequalities are ignored, the optimal solution over each subdivision has x2 = x3 = x4 = x5 = 0. The resulting solution in subdivision 1 gives z = ? 8(1) ? 2(0) ? 4(0) ? 7(0) ? 5(0) + 10 = 2, 9. 7 Implicit Enumeration 299 and happens to satisfy the inequalities, so that the optimal solution to the original problem is at least 2, z ? ? 2. Also, subdivision 1 has been fathomed: The above solution is best among all 0–1 combinations with x1 = 1; thus it must be best among those satisfying the inequalities. No other feasible 0–1 combination in subdivision 1 needs to be evaluated explicitly.

These combinations have been considered implicitly. The solution with x2 = x3 = x4 = x5 = 0 in subdivision 2 is the same as the original solution with every variable at zero, and is infeasible. Consequently, the region must be subdivided further, say with x2 = 1 or x2 = 0, giving: The enumeration tree

to this point is as given in Fig. 9. 19. For subdivision 3 : For subdivision 4 : x1 = 0, x2 = 1; x1 = 0, x2 = 0. Figure 9. 19 Observe that this tree differs from the enumeration trees of the previous sections. For the earlier procedures, the linear-programming solution used to analyze each subdivision was speci? ed explicitly in a box.

Here the 0–1 solution (ignoring the inequalities) used to analyze subdivisions is not stated explicitly, since i it is known simply by setting free variables to zero. In subdivision 3 , for example, x1 = 0 and x2 = 1 are ? xed, and the free variables x3 , x4 andx5 are set to zero. Continuing to ? x variables and subdivide in this fashion produces the complete tree shown in Fig. 9. 20. The tree is not extended after analyzing subdivisions 4, 5, 7, 9, and 10, for the following reasons. i i) At 5 , the solution x1 = 0, x2 = x3 = 1 , with free variables x4 = x5 = 0, is feasible, with z = 4 , thus providing an improved lower bound on z ? i ii) At 7 , the solution x1 = x3 = 0, x2 = x4 = 1, and free variable x5 = 0, has z = 1 < 4, so that no i solution in that subdivision can be as good as that generated at 5 . i i iii) At 9 and 10 , every free variable is ? xed. In each case, the subdivisions contain only a single point, which is infeasible, and further subdivision is not possible. i iv) At 4 , the second inequality (with ? xed variables x1 = x2 = 0) reads: No 0–1 values of x3 , x4 , or x5 ‘‘ completing’’ the ? xed variables x1 = x2 = 0 satisfy this constraint, since the lowest value for the lefthand side of this equation is ? when x3 = x4 = 1 and x5 = 0. The subdivision then has no feasible solution and need not be analyzed further. The last observation is completely general. If, at any point after substituting for the ? xed variables, the sum of the remaining negative coef? cients in any constraint exceeds the righthand side, then the

region de? ned by these ? xed variables has no feasible solution. Due to the special nature of the 0–1 problem, there are a number of other such tests that can be utilized to reduce the number of subdivisions generated. The ef? iency of these tests is measured by weighing the time needed to perform them against the time saved by fewer subdivisions. The techniques used here apply to any integer-programming problem involving only binary variables, so that implicit enumeration is an alternative branch-and-bound procedure for this class of problems. In this case, subdivisions are fathomed if any of three conditions hold: ? 2x3 ? x4 + x5 ? ? 4. 300 Integer Programming 9. 7 Figure 9. 20 i) the integer program is known to be infeasible over the subdivision, for example, by the above infeasibility test; ii) the 0–1 solution obtained by setting free variables to zero satis? s the linear inequalities; or iii) the objective value obtained by setting free variables to zero is no larger than the best feasible 0–1 solution previously generated. These conditions correspond to the three stated earlier for fathoming in the usual branch-and-bound procedure. If a region is not fathomed by one of these tests, implicit enumeration subdivides that region by selecting any free variable and ? xing its values to 0 or 1. Our arguments leading to the algorithm were based upon stating the original 0–1 problem in the following standard form: 1. the objective is a maximization with all coef? cients negative; and 2. onstraints are speci? ed as ‘‘ less than or equal to’’ inequalities. As usual, minimization problems are transformed to maximization by multiplying cost coef? cients by ? 1. If x j appears in the maximization form with a positive coef? cient, then the variable substitution x j = 1 ? x j everywhere in the model leaves the binary variable x j with a negative objective-function coef? cient. Finally, ‘‘ greater than or equal to’’

constraints can be multiplied by ? 1 to become ‘‘ less than or equal to’’ constraints; and generalequalityconstraints are converted to inequalities by the special technique discussed in Exercise 17 of Chapter 2.

Like the branch-and-bound procedure for general integer programs, the way we choose to subdivide regions can have a profound effect upon computations. In implicit enumeration, we begin with the zero solution x1 = x2 = · · · = xn = 0 and generate other solutions by setting variables to 1. One natural approach is to subdivide based upon the variable with highest objective contribution. For the sample problem, this would imply subdividing initially with x2 = 1 or x2 = 0. Another approach often used in practice is to try to drive toward feasibility as soon as possible.

For instance, when x1 = 0, x2 = 1, and x3 = 0 are ? xed in the example problem, we could subdivide based upon either x4 or x5 . Setting x4 or x5 to 1 and substituting for the ? xed variables, we ? nd that the constraints become: 9. 8 Cutting Planes 301 x4 = 1, x5 (free) = 0 : ? 3(0) ? 3(1) + (0) + 2(1) + 3(0) ? ? 2, ? 5(0) ? 3(1) ? 2(0) ? 1(1) + (0) ? ? 4, x5 = 1, x4 (free) = 0 : ? 3(0) ? 3(1) + (0) + 2(0) + 3(1) ? ? 2, ? 5(0) ? 3(1) ? 2(0) ? 1(0) + (1) ? ? 4. For x4 = 1, the ? rst constraint is infeasible by 1 unit and the second constraint is feasible, giving 1 total unit of infeasibility.

For x5 = 1, the ? rst constraint is infeasible by 2 units and the second by 2 units, giving 4 total units of infeasibility. Thus x4 = 1 appears more favorable, and we would subdivide based upon that variable. In general, the variable giving the least total infeasibilities by this approach would be chosen next. Reviewing the example problem the reader will see that this

approach has been used in our solution. 9. 8 CUTTING PLANES The cutting-plane algorithm solves integer programs by modifying linear-programming solutions until the integer solution is obtained.

It does not partition the feasible region into subdivisions, as in branch-and-bound approaches, but instead works with a single linear program, which it re? nes by adding new constraints. The new constraints successively reduce the feasible region until an integer optimal solution is found. In practice, the branch-and-bound procedures almost always outperform the cutting-plane algorithm. Nevertheless, the algorithm has been important to the evolution of integer programming. Historically, it was the ? rst algorithm developed for integer programming that could be proved to converge in a ? nite number of steps.

In addition, even though the algorithm generally is considered to be very inef? cient, it has provided insights into integer programming that have led to other, more ef? cient, algorithms. Again, we shall discuss the method by considering the sample problem of the previous sections: z ? = max 5x1 + 8x2 , subject to: x1 + x2 + s1 = 6, 5x1 + 9x2 + s2 = 45, x1 , x2 , s1 , s2 ? 0. s1 and s2 are, respectively, slack variables for the ? rst and second constraints. Solving the problem by the simplex method produces the following optimal tableau: (? z) x1 3 1 5 ? 4 s1 ? 4 s2 = ? 41 4 , 1 5 x2 ? s1 + 4 s2 = 9 1 + 4 s1 ? 4 s2 = 9 4, 15 4, (11) x1 , x2 , s1 , s2 , s3 ? 0. Let us rewrite these equations in an equivalent but somewhat altered form: (? z) x1 ? 2s1 ? s2 +42 = x2 ? 2s1 3 4 1 4 3 4 3 1 ? 4 s1 ? 4 s2 , 1 3 ? 4 s1 ? 4 s2 , 3 1 ? 4 s1 ? 4 s2 , +2s1 ? s2 ? 2 = ? 3= x1 , x2 , s1 , s2 ? 0. These algebraic manipulations have isolated integer coef? cients to one side of the equalities

and fractions to the other, in such a way that the constant terms on the righthand side are all nonnegative and the slack variable coef? cients on the righthand side are all nonpositive. 302 Integer Programming 9. 8

In any integer solution, the lefthand side of each equation in the last tableau must be integer. Since s1 and s2 are nonnegative and appear to the right with negative coef? cients, each righthand side necessarily must be less than or equal to the fractional constant term. Taken together, these two observations show that both sides of every equation must be an integer less than or equal to zero (if an integer is less than or equal to a fraction, it necessarily must be 0 or negative). Thus, from the ? rst equation, we may write: 3 4 3 1 ? 4 s1 ? 4 s2 ? 0 and integer, or, introducing a slack variable s3 , 3 4 3 1 ? 4 s1 ? 4 s2 + s3 = 0, 3 ? 0 s4 ? 0 s5 ? 0 and integer. (C1) Similarly, other conditions can be generated from the remaining constraints: 1 4 3 4 1 3 ? 4 s1 ? 4 s2 + s4 = 0