A Tutorial on Integer Programming

Gérard Cornuéjols Michael A. Trick
Matthew J. Saltzman

1995

These notes are meant as an adjunct to Chapter 9 and 10 in Murty. You are responsible for what appears in these notes as well as Sections 9.1–9.7, 10.1–10.3, 10.5, 10.6, 10.8 in the text.

Contents

1 Introduction

Consider the manufacture of television sets. A linear programming model might give a production plan of 205.7 sets per week. In such a model, most people would have no trouble stating that production should be 205 sets per week (or even "roughly 200 sets per week"). On the other hand, suppose we were buying warehouses to store finished goods, where a warehouse comes in a set size. Then a model that suggests we purchase 0.7 warehouse at some location and 0.6 somewhere else would be of little value. Warehouses come in integer quantities, and we would like our model to reflect that fact.

This integrality restriction may seem rather innocuous, but in reality it has far reaching effects. On one hand, modeling with integer variables has turned out to be useful far beyond restrictions to integral production quantities. With integer variables, one can model logical requirements, fixed costs, sequencing and scheduling requirements, and many other problem aspects. In AMPL, one can easily change a linear programming problem into an integer program.

The downside of all this power, however, is that problems with as few as 40 variables can be beyond the abilities of even the most sophisticated computers. While these small problems are somewhat artificial, most real problems with more than 100 or so variables are not possible to solve unless they show specific exploitable structure. Despite the possibility (or even likelihood) of enormous computing times, there are methods that can be applied to solving integer programs. The CPLEX solver in AMPL is built on a combination of methods, but based on a method called branch and bound. The purpose of this chapter is to show some interesting integer programming applications and to describe some of these solution techniques as well as possible pitfalls.

First we introduce some terminology. An integer programming problem in which all variables are required to be integer is called a *pure integer programming problem*. If some variables are restricted to be integer and some are not then the problem is a *mixed integer programming problem*. The case where the integer variables are restricted to be 0 or 1 comes up surprising often. Such problems are called *pure (mixed) 0-1 programming problems* or *pure (mixed) binary integer programming problems*.

2 Modeling with Integer Variables

The use of integer variables in production when only integral quantities can be produced is the most obvious use of integer programs. In this section, we will look at some less obvious ones. The text also goes through a number of them (some are repeated here).

2.1 Capital Budgeting

Suppose we wish to invest \$14,000. We have identified four investment opportunities. Investment 1 requires an investment of \$5,000 and has a present value (a time-discounted value) of \$8,000; investment 2 requires \$7,000 and has a value of \$11,000; investment 3 requires \$4,000 and has a value of \$6,000; and investment 4 requires \$3,000 and has a value of \$4,000. Into which investments should we place our money so as to maximize our total present value?

As in linear programming, our first step is to decide on our variables. This can be much more difficult in integer programming because there are very clever ways to use integrality restrictions. In this case, we will use a 0-1 variable x_j for each investment. If x_j is 1 then we will make investment j. If it is 0, we will not make the investment. This leads to the 0-1 programming problem:

$$\begin{array}{rl} \text{Maximize} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0,1\}\ j = 1, \ldots 4. \end{array}$$

Now, a straightforward "bang for buck" suggests that investment 1 is the best choice. In fact, ignoring integrality constraints, the optimal linear programming solution is $x_1 = 1$, $x_2 = 1$, $x_3 = 0.5$, $x_4 = 0$ for a value of \$22,000. Unfortunately, this solution is not integral. Rounding x_3 down to 0 gives a feasible solution with a value of \$19,000. There is a better integer solution, however, of $x_1 = 0$, $x_2 = x_3 = x_4 = 1$ for a value of \$21,000. This example shows that rounding does not necessarily give an optimal value.

There are a number of additional constraints we might want to add. For instance, consider the following constraints:

1. We can only make two investments.

2. If investment 2 is made, investment 4 must also be made.

3. If investment 1 is made, investment 3 cannot be made.

All of these, and many more *logical restrictions*, can be enforced using 0-1 variables. In these cases, the constraints are:

1. $x_1 + x_2 + x_3 + x_4 \leq 2$
2. $x_2 - x_4 \leq 0$
3. $x_1 + x_3 \leq 1$.

Exercise 1 (Optional) *As the leader of an oil exploration drilling venture, you must determine the best selection of 5 out of 10 possible sites. Label the sites $s_1, s_2, \ldots, s_{10}$ and the expected profits associated with each as $p_1, p_2, \ldots, p_{10}$.*

(i) If site s_2 is explored, then site s_3 must also be explored. Furthermore, regional development restrictions are such that

(ii) Exploring sites s_1 and *s_7 will prevent you from exploring site s_8.*

(iii) Exploring sites s_3 or *s_4 will prevent you from exploring site s_5.*

Formulate an integer program to determine the best exploration scheme.

2.1.1 Multiperiod Capital Budgeting

In the preceding, we considered making a single investment in a project for the duration of some term, and receiving its return at the end of the term. In practice, we may face a choice among projects that require investments of different amounts in each of several periods (with possibly different budgets available in each period), with the return being realized over the life of the project. In this case, we can still model the problem with variables

$$x_j = \begin{cases} 1 & \text{if we invest in project } j \\ 0 & \text{otherwise,} \end{cases}$$

the objective is still to maximize the sum of the returns on the projects selected, and there is now a budget constraint for each period. For example, suppose we wish to invest \$14,000, \$12,000, and \$15,000 in each month of the next quarter. We have identified four investment opportunities. Investment 1 requires an investment of \$5,000, \$8,000, and \$2,000 in month 1, 2, and 3, respectivey, and has a present value (a time-discounted value) of \$8,000; investment 2 requires \$7,000 in month 1 and \$10,000 in period 3, and has a value of \$11,000; investment 3 requires \$4,000 in period 2 and \$6,000 in period 3, and has a value of \$6,000; and investment 4 requires \$3,000, \$ 4,000, and \$5,000 and has a value of \$4,000. The corresponding integer program is

$$\begin{array}{rl} \text{Maximize} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 \qquad\quad + 3x_4 \leq 14 \\ & 8x_1 \qquad\quad + 4x_3 + 4x_4 \leq 12 \\ & 2x_1 + 10x_2 + 6x_3 + 4x_4 \leq 15 \\ & x_j \in \{0,1\}. \end{array}$$

2.2 Knapsack

The knapsack problem is a particularly simple integer program: it has only one constraint. Furthermore, the coefficients of this constraint and the objective are all non-negative. For example, the following is a knapsack problem:

$$\begin{array}{rl} \text{Maximize} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0,1\}. \end{array}$$

The traditional story is that there is a knapsack (here of capacity 14). There are a number of items (here there are four items), each with a size and a value (here item 2 has size 7 and value 11). The objective is to maximize the total value of the items in the knapsack. Knapsack problems are nice because they are (usually) easy to solve, as we will see in the dynamic programming section of this course.

To solve the associated linear program, it is simply a matter of determining which variable gives the most "bang for the buck". If you take c_j/a_j (the objective coefficient/constraint coefficient) for each variable, the one with the highest ratio is the best item to place in the knapsack. Then the item with the second highest ratio is put in and so on until we reach an item that cannot fit. At this point, a fractional amount of that item is placed in the knapsack to completely fill it. In our example, the variables are already ordered by the ratio. We would first place item 1 in, then item 2, but item 3 doesn't fit: there are only 2 units left, but item 3 has size 4. Therefore, we take half of item 3. The solution $x_1 = 1$, $x_2 = 1$, $x_3 = 0.5$, $x_4 = 0$ is the optimal solution to the linear program.

As already observed in Section 2.1, this solution is quite different from the optimal solution to the knapsack problem. Nevertheless, it will play an important role in the solution of the problem by branch and bound as we will see shortly.

2.2.1 Converting a Single-Constraint 0-1 IP to a Knapsack Problem

The nonnegativity requirement on the coefficients in the knapsack problem is not really a restriction. As mathematicians say, we can assume this requirement holds *without loss of generality*, since any single-constraint 0-1 IP can be converted to knapsack form. The same cannot be said for general integer programs with one or more constraints. For 0-1 programs with more than one constraint, the technique applies to columns where *all* entries are negative.

Consider the general 0-1 IP with one constraint:

$$\begin{aligned} \text{Maximize} \quad & \sum_{j=1}^{n} c_j x_j \\ \text{subject to} \quad & \sum_{j=1}^{n} w_j x_j \leq w_0 \\ & x_j \in \{0,1\} \; j = 1, \dots, n \end{aligned}$$

If any x_j has $c_j < 0$ and $w_j \geq 0$ then it is clear that this x_j will be 0 in any optimal solution, so we can fix $x_j = 0$ and remove it from the problem. If any x_j has $c_j \geq 0$ and $w_j < 0$ then it is clear that $x_j = 1$ in any optimal solution (including item j actually increases capacity by $|w_j|$ and increases the objective as well), so we can fix $x_j = 1$ and remove it from the problem as well (adjusting the capacity and objective values accordingly).

If $c_j < 0$ and $w_j < 0$ then including item j incurs a cost but increases capacity, so there is no obious justification for fixing x_j to either 0 or 1. Instead, we can define a new variable $y_j = 1 - x_j$ and replace x_j with $1 - y_j$. After collecting the constants, we have a new, equivalent problem with nonnegative coefficients on y_j.

For example, consider the IP

$$\begin{aligned} \text{Maximize} \quad & 8x_1 + 11x_2 - 6x_3 + 4x_4 \\ \text{subject to} \quad & 5x_1 + 7x_2 - 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0,1\}. \end{aligned}$$

Defining $y_3 = 1 - x_3$ and substituting $x_3 = 1 - y_3$ gives

$$\begin{array}{rl} \text{Maximize} & 8x_1 + 11x_2 - 6(1 - y_3) + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 - 4(1 - y_3) + 3x_4 \leq 14 \\ & x_j, y_3 \in \{0, 1\}, \end{array}$$

or equivalently,

$$\begin{array}{rl} \text{Maximize} & 8x_1 + 11x_2 + 6y_3 + 4x_4 - 6 \\ \text{subject to} & 5x_1 + 7x_2 + 4y_3 + 3x_4 \leq 18 \\ & x_j, y_3 \in \{0, 1\}. \end{array}$$

2.2.2 Multidimensional and General Integer Knapsack Problems

A problem of the form

$$\begin{array}{rl} \text{Maximize} & cx \\ \text{subject to} & Ax \leq b \\ & x_j \in \{0, 1\}, \end{array}$$

where $c_j \geq 0$, $A_{ij} \geq 0$, and $b_i \geq 0$ is called a multidimensional knapsack problem.

If the x_js are allowed to take on arbitrary integer values (rather than being restricted to 0 or 1), then the problem is a general knapsack problem. In the story, this would correspond to being allowed to take multiple items of each type, with all items of a given type weighing the same and having value independent of the number of such items included.

2.3 The Lockbox Problem

Consider a national firm that receives checks from all over the United States. Due to the vagaries of the U.S. Postal Service, as well as the banking system, there is a variable delay from when the check is postmarked (and hence the customer has met her obligation) and when the check clears (and when the firm can use the money). For instance, a check mailed in Pittsburgh sent to a Pittsburgh address might clear in just two days. A similar check sent to Los Angeles might take eight days to clear. It is in the firm's interest to have the check clear as quickly as possible since then the firm can use the money. In order to speed up this clearing, firms open offices (called lockboxes) in different cities to handle the checks.

For example, suppose we receive payments from four regions (West, Midwest, East, and South). The average daily value from each region is as follows: \$70,000 from the West, \$50,000 from the Midwest, \$60,000 from the East, and \$40,000 from the South. We are considering opening lockboxes in L.A., Chicago, New York, and/or Atlanta. Operating a lockbox costs \$50,000 per year. The average days from mailing to clearing is given in the table. Which lockboxes should we open?

From	L.A.	Chicago	New York	Atlanta
West	2	6	8	8
Midwest	6	2	5	5
East	8	5	2	5
South	8	5	5	2

Table 1: Clearing Times

First we must calculate the losses due to lost interest for each possible assignment. For example, if the West sends to New York, then on average there will be \$560,000 ($= 8 \times \$70,000$) in process on any given day. Assuming an investment rate of 20%, this corresponds to a yearly loss of \$112,000. We can calculate the losses for the other possibilities in a similar fashion to get table 2.

From	L.A.	Chicago	New York	Atlanta
West	28	84	112	112
Midwest	60	20	50	50
East	96	60	24	60
South	64	40	40	16

Table 2: Lost Interest (\$1000)

The formulation takes a bit of thought. Let y_j be a 0-1 variable that is 1 if lockbox j is opened and 0 if it is not. Let x_{ij} be 1 if region i sends to lockbox j.

Our objective is to minimize our total yearly costs. This is:

$$28x_{11} + 84x_{12} + 112x_{13} + 112x_{14} + 60x_{21} + \ldots + 50y_1 + 50y_2 + 50y_3 + 50y_4.$$

One set of constraints is as follows:

$$\sum_j x_{ij} = 1 \text{ for all } i$$

(each region must be assigned to one lockbox).

A more difficult set of constraints is that a region can only be assigned to an open lockbox. For lockbox 1 (L.A.), this can be written

$$x_{11} + x_{21} + x_{31} + x_{41} \leq 100y_1$$

(There is nothing special about 100; any number at least 4 would do.) Suppose we do not open L.A. Then y_1 is 0, so all of x_{11}, x_{21}, x_{31}, and x_{41} must also be. If y_1 is 1 then there is no restriction on the x values.

We can create constraints for the other lockboxes to finish off the integer program. For this problem, we would have twenty variables (four y variables, sixteen x variables) and eight constraints. This gives the following 0-1 linear program:

$$\begin{array}{rl}
\text{Minimize} & 28x_{11} + 84x_{12} + 112x_{13} + 112x_{14} + 60x_{21} + 20x_{22} + 50x_{23} + 50x_{24} \\
& +96x_{31} + 60x_{32} + 24x_{33} + 60x_{34} + 64x_{41} + 40x_{42} + 40x_{43} + 16x_{44} \\
& +50y_1 + 50y_2 + 50y_3 + 50y_4 \\
\text{subject to} & x_{11} + x_{12} + x_{13} + x_{14} = 1 \\
& x_{21} + x_{22} + x_{23} + x_{24} = 1 \\
& x_{31} + x_{32} + x_{33} + x_{34} = 1 \\
& x_{41} + x_{42} + x_{43} + x_{44} = 1 \\
& x_{11} + x_{21} + x_{31} + x_{41} \leq 100y_1 \\
& x_{12} + x_{22} + x_{32} + x_{42} \leq 100y_2 \\
& x_{13} + x_{23} + x_{33} + x_{43} \leq 100y_3 \\
& x_{14} + x_{24} + x_{34} + x_{44} \leq 100y_4 \\
& x_{ij}, y_j \in \{0.1\} \;\; i = 1, \ldots, 4, \;\; j = 1, \ldots, 4
\end{array}$$

A possible AMPL model is

```
set Region;
set Box;

param cost{Box} >= 0;
param loss{Region, Box} >= 0;
param big > 0;
```

```
var open{Box} binary;
var assign{Region, Box} binary;

minimize Cost: sum{i in Region, j in Box} loss[i,j] * assign[i,j]
               + sum{j in Box} cost[j] * open[j];

subject to Assign{i in Region}: sum{j in Box} assign[i,j] = 1;
subject to Open{j in Box}: sum{i in Region} assign[i,j] <= big * open[j];

---------------------------

set Region := West Midwest East South;
set Box := LA Chi NY Atl;

param cost := LA 50 Chi 50 NY 50 Atl 50;
param loss :       LA Chi  NY Atl :=
         West      28  84 112 112
         Midwest   60  20  50  50
         East      96  60  24  60
         South     64  40  40  16;
param big := 100;
```

If we ignore integrality, we get the solution $x_{11} = x_{22} = x_{33} = x_{44} = 1$, $y_1 = y_2 = y_3 = y_4 = 0.01$ and the rest equal to 0. Note that we get no useful information out of this linear programming solution.

The above is a perfectly reasonable 0-1 programming formulation of the lockbox problem. Note that many variations are possible (New York costs more to operate in than other cities, South won't send to L.A., and so on).

There are other formulations, however. For instance, consider the sixteen constraints of the form

$$x_{ij} \leq y_j$$

These constraints also force a region to only use open lockboxes (check this!). It might seem that a larger formulation is less efficient and therefore should be avoided. This is not the case! If we solve the linear program with the above constraints, we get the solution $x_{11} = x_{24} = x_{34} = x_{44} = y_1 = y_4 = 1$ with the rest equal to zero. In fact, we have an integer solution, which must therefore be optimal! Different formulations can have very different properties with respect to their associated linear program. One very active research area is to take common problems and find good reformulations.

The above is an example of a *fixed charge* problem. There is a fixed charge for opening a lockbox, but then it can be used as much as desired. There are many other types of fixed charge problems, including the plant location problems described in Murty. Similar techniques also apply (as described in Murty) for a variety of constraints, such as batch size constraints (where if an activity is engaged in at all, it must have a minimum level, i.e., $x_j = 0$ or $x_j \geq l_j$) and other types of "either-or" or *disjunctive* constraints.

2.4 Set Covering

To illustrate this model, consider the following *location* problem: A city is reviewing the location of its fire stations. The city is made up of a number of neighborhoods, as illustrated in Figure 1.

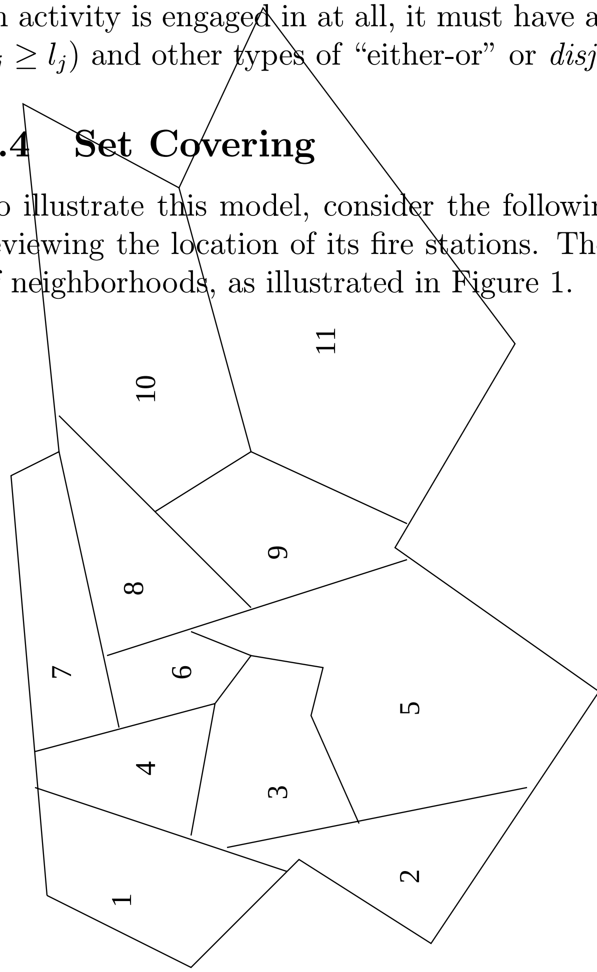

Figure 1: Map of the city

A fire station can be placed in any neighborhood. It is able to handle the fires for both its neighborhood and any adjacent neighborhood (any neighborhood with a non-zero border with its home neighborhood). The objective is to minimize the number of fire stations used.

We can create one variable x_j for each neighborhood j. This variable will be 1 if we place a station in the neighborhood, and will be 0 otherwise. This leads to the following formulation

$$
\begin{array}{rlr}
\text{Minimize} & x_1 + x_2 + x_3 + x_4 + x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} + x_{11} & \\
\text{subject to} & x_1 + x_2 + x_3 + x_4 & \geq 1 \\
& x_1 + x_2 + x_3 + x_5 & \geq 1 \\
& x_1 + x_2 + x_3 + x_4 + x_5 + x_6 & \geq 1 \\
& x_1 + x_3 + x_4 + x_6 + x_7 & \geq 1 \\
& x_2 + x_3 + x_5 + x_6 + x_8 + x_9 & \geq 1 \\
& x_3 + x_4 + x_5 + x_6 + x_7 + x_8 & \geq 1 \\
& x_4 + x_6 + x_7 + x_8 & \geq 1 \\
& x_5 + x_6 + x_7 + x_8 + x_9 + x_{10} & \geq 1 \\
& x_5 + x_8 + x_9 + x_{10} + x_{11} & \geq 1 \\
& x_8 + x_9 + x_{10} + x_{11} & \geq 1 \\
& x_9 + x_{10} + x_{11} & \geq 1 \\
& x_j \in \{0,1\} \quad j = 1, \ldots, 11 &
\end{array}
$$

The first constraint states that there must be a station either in neighborhood 1 or in some adjacent neighborhood. The next constraint is for neighborhood 2 and so on. Notice that the constraint coefficient a_{ij} is 1 if neighborhood i is adjacent to neighborhood j or if $i = j$ and 0 otherwise. The jth column of the constraint matrix represents the set of neighborhoods that can be served by a fire station in neighborhood j. We are asked to find a set of such subsets j that *covers* the set of all neighborhoods in the sense that every neighborhood appears in the service subset associated with *at least* one fire station.

One optimal solution to this is $x_3 = x_8 = x_9 = 1$ and the rest equal to 0.

This is an example of the *set covering problem*. The set covering problem is characterized by having binary variables, $\geq$ constraints each with a right hand side of 1, and having simply sums of variables as constraints. In general, the objective function can have any coefficients, though here it is of a particularly simple form.

2.4.1 Set Packing and Partitioning

Other problems involving relationships between a set and some of its subsets are related to set covering. The *set packing problem* arises when each set element must appear in *at most* one subset. In this case, the constraints are of the less-than-or-equal form. The *set partitioning problem* arises when each set element must appear in *exactly* one subset, and the constraints in this problem are equality constraints.

The airline crew scheduling problem discussed in Murty is usually solved in practice as a set partitioning problem, with exactly one crew assigned to each flight leg. After all, there is only one crew actually flying the plane on each leg. A crew that must ride a plane to another destination is said to be *deadheading*. Since the costs associated with deadheading are different from those associated with active duty, deadhead legs should not appear as a consequence of the solution, but instead need to be factored into the duty periods by the software that generates the pairings.

2.5 Traveling Salesperson Problem

Consider a traveling salesperson who must visit each of 20 cities before returning home. She knows the distance between each of the cities and wishes to minimize the total distance traveled while visiting all of the cities. In what order should she visit the cities?

Let there be n cities, numbered from 1 up to n. For each pair of cities (i, j) let c_{ij} be the cost of going from city i to city j or from city j to city i. Let's let x_{ij} be 1 if the person travels between cities i and j (either from city i to city j or from j to i). This problem is known as the *symmetric* TSP. In the *assymetric* TSP, the cost to travel in one direction may differ from the cost to travel in the other, and the decision variables must distinguish between the two directions. Clearly the assymetric problem is the more general. Murty discusses the assymetric formulation, but we will concentrate on the symmetric problem.

The objective is to minimize $\sum_{i=1}^{n} \sum_{j=1}^{i-1} c_{ij} x_{ij}$. The constraints are harder to find. Consider the following set:

$$\sum_{j \neq i} x_{ij} = 2 \text{ for all } i.$$

These constraints say that every city must be visited. These constraints are not enough, however, since it is possible to have multiple cycles (*subtours*), rather than one big cycle (*tour*) through all the points. To handle this condition, we can use the following set of *subtour elimination constraints*:

$$\sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \text{ for all } S \subset N.$$

This set states that, for any subset of cities S, the tour must enter and exit that set. These, together with $x_{ij} \in \{0, 1\}$ is sufficient to formulate the traveling salesperson problem (TSP) as an integer program.

Note however, that there are a tremendous number of constraints: for our 20-city problem, that number is roughly 524,288. For a 300-city problem, this would amount to 101851798816724304313422284420468908052573419683296 8125318070224677190649881668353091698688 constraints. Try putting that into AMPL!

Despite the apparent complexity of this formulation, it lies at the heart of the most promising current approach to solving medium (100–1000 city) TSPs. We will see this again in the cutting plane section. Other forms of subtour elimination constraints are possible, but they suffer from the same explosive rate of growth in number. Subtour elimination constraints for the assymetric problem are equally difficult. In fact, Murty does not explicitly write them down at all!

Exercise 2 (Optional) *Orders from 5 different destinations are delivered from a central warehouse. Routes are assigned to different carriers. Suppose that there are 9 feasible routes with each route specifying the destinations to which orders are delivered.*

Route 1: 1,3,4
Route 2: 1,4,5
Route 3: 1,2,5
Route 4: 2,3,5
Route 5: 2,4,5
Route 6: 3,4,5
Route 7: 1
Route 8: 2,3
Route 9: 4,5

The length of the routes are 10, 12, 12, 13, 11, 9, 7, 8 and 8 respectively. Formulate an integer program that will result in the shortest total distance routing. Each destination must be on at least one of the chosen routes.

Exercise 3 (Optional) *Consider a job-shop scheduling problem involving eight operations on a single machine with a total of two end products. Let* p_j *be the processing time for the* j*th operation. Since each operation requires a machine setup, it is assumed that any operation once started must be completed before a new operation can be undertaken on the machine. The sequencing of the operations must satisfy the following precedence relationships: Operations 1 and 2 must precede Operation 4 (i.e. both Operations 1 and 2 must be completed before Operation 4 can be started), Operations 4 and 5 precede 7, Operation 3 precedes 6 and, finally, Operations 5 and 6 precede 8. Once Operation 7 is completed, Product 1 is available. Similarly, once Operation 8 is completed, Product 2 is available. The due date for Product 1 is* d_1*. The objective is to minimize the delivery date of Product 2 while meeting the due date on Product 1.*

3 Solving Integer Programs

We have gone through a number of examples of integer programs. The text gives a few others. A natural question is "How can we get solutions to these models?" There are two common approaches. Historically, the first method developed was based on cutting planes (adding constraints to force integrality). In the last twenty years or so, however, the most effective technique has been based on dividing the problem into a number of smaller problems in a method called *branch and bound*. Recently (the last ten years or so), cutting planes have made a resurgence in the form of facets and polyhedral characterizations. All these approaches involve solving a series of *linear* programs. So that is where we will begin.

3.1 Relationship to Linear Programming

Given an integer program

$$\begin{array}{rl} \text{Minimize (or maximize)} & cx \\ \text{subject to} & Ax = b \\ & x \geq 0 \text{ and integer,} \end{array} \qquad \text{(IP)}$$

there is an associated linear program called the *linear relaxation* formed by dropping the integrality restrictions:

$$\begin{array}{rl} \text{Minimize (or maximize)} & cx \\ \text{subject to} & Ax = b \\ & x \geq 0. \end{array} \tag{LR}$$

Since (LR) is less constrained than (IP), the following are immediate:

- If (IP) is a minimization, the optimal objective value for (LR) is less than or equal to the optimal objective for (IP).
- If (IP) is a maximization, the optimal objective value for (LR) is greater than or equal to that of (IP).
- If (LR) is infeasible, then so is (IP).
- If (LR) is optimized by integer variables, then that solution is feasible and optimal for (IP).
- If the objective function coefficients are integer, then for minimization, the optimal objective for (IP) is greater than or equal to the "round up" of the optimal objective for (LR). For maximization, the optimal objective for (IP) is less than or equal to the "round down" of the optimal objective for (LR).

So solving (LR) does give some information: it gives a bound on the optimal value, and, if we are lucky, may give the optimal solution to IP. We saw, however, that rounding the solution of LR will not in general give the optimal solution of (IP). In fact, for some problems it is difficult to round and even get a feasible solution.

Exercise 4 (Optional) *Consider the problem*

$$\begin{array}{rl} \textit{Maximize} & 20x_1 + 10x_2 + 10x_3 \\ \textit{subject to} & 2x_1 + 20x_2 + 4x_3 \leq 15 \\ & 6x_1 + 20x_2 + 4x_3 = 20 \\ & x_1, x_2, x_3 \geq 0 \ \textit{integer.} \end{array}$$

Solve this problem as a linear program. Then, show that it is impossible to obtain a feasible integer solution by rounding the values of the variables.

3.2 Branch and Bound

We will explain branch and bound by using the capital budgeting example from the previous section. In that problem, the model is

$$\begin{array}{rl} \text{Maximize} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \text{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0, 1\}\ j = 1, \ldots 4. \end{array}$$

The linear relaxation solution is $x_1 = 1, x_2 = 1, x_3 = 0.5, x_4 = 0$ with a value of 22. We know that no integer solution will have value more than 22. Unfortunately, since x_3 is not integer, we do not have an integer solution yet.

We want to force x_3 to be integer. To do so, we *branch* on x_3, creating two new problems. In one, we will add the constraint $x_3 = 0$. In the other, we add the constraint $x_3 = 1$. This is illustrated in Figure 2.

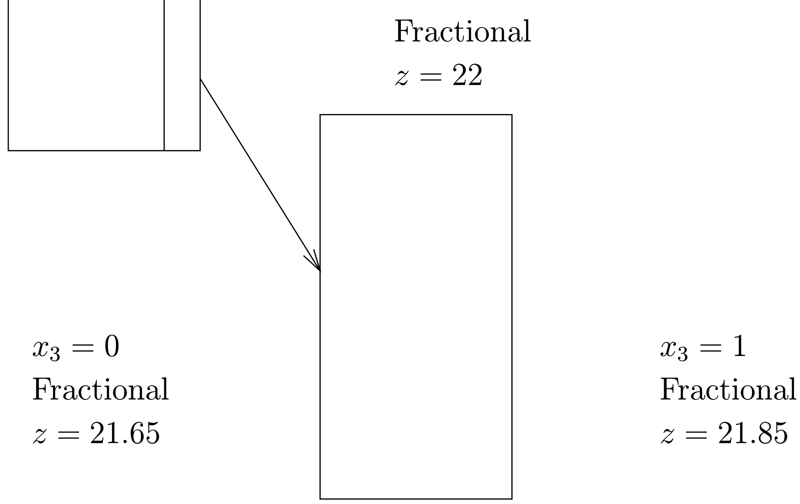

Figure 2: First Branching

Note that any optimal solution to the overall problem must be feasible to one of the subproblems. If we solve the linear relaxations of the subproblems, we get the following solutions:

- $x_3 = 0$: objective 21.65, $x_1 = 1$, $x_2 = 1$, $x_3 = 0$, $x_4 = 0.667$;

- $x_3 = 1$: objective 21.85, $x_1 = 1$, $x_2 = 0.714$, $x_3 = 1$, $x_4 = 0$.

At this point we know that the optimal integer solution is no more than 21.85 (we actually know it is less than or equal to 21 (Why?)), but we still do not have any feasible integer solution. So, we will take a subproblem and branch on one of its variables. In general, we will choose the subproblem as follows:

- We will choose an *active* subproblem, which so far only means one we have not chosen before, and
- We will choose the subproblem with the highest solution value (for maximization) (lowest for minimization).

In this case, we will choose the subproblem with $x_3 = 1$, and branch on x_2. After solving the resulting subproblems, we have the branch and bound tree in Figure 3.

The solutions are:

- $x_3 = 1$, $x_2 = 0$: objective 18, $x_1 = 1$, $x_2 = 0$, $x_3 = 1$, $x_4 = 1$;
- $x_3 = 1$, $x_2 = 1$: objective 21.8, $x_1 = 0.6$, $x_2 = 1$, $x_3 = 1$, $x_4 = 0$.

We now have a feasible integer solution with value 18. Furthermore, since the $x_3 = 1, x_2 = 0$ problem gave an integer solution, no further branching on that problem is necessary. It is not active due to integrality of solution. There are still active subproblems that might give values more than 18. Using our rules, we will branch on problem $x_3 = 1, x_2 = 1$ by branching on x_1 to get Figure 4.

The solutions are:

- $x_3 = 1$, $x_2 = 1$, $x_1 = 0$: objective 21, $x_1 = 0$, $x_2 = 1$, $x_3 = 1$, $x_4 = 1$;
- $x_3 = 1$, $x_2 = 1$, $x_1 = 1$: infeasible.

Our best integer solution now has value 21. The subproblem that generates that is not active due to integrality of solution. The other subproblem generated is not active due to infeasibility. There is still a subproblem that is active. It is the subproblem with solution value 21.65. By our "round-down" result, there is no better solution for this subproblem than 21. But we already have a solution with value 21. It is not useful to search for another such solution. We can *fathom* this subproblem based on the above

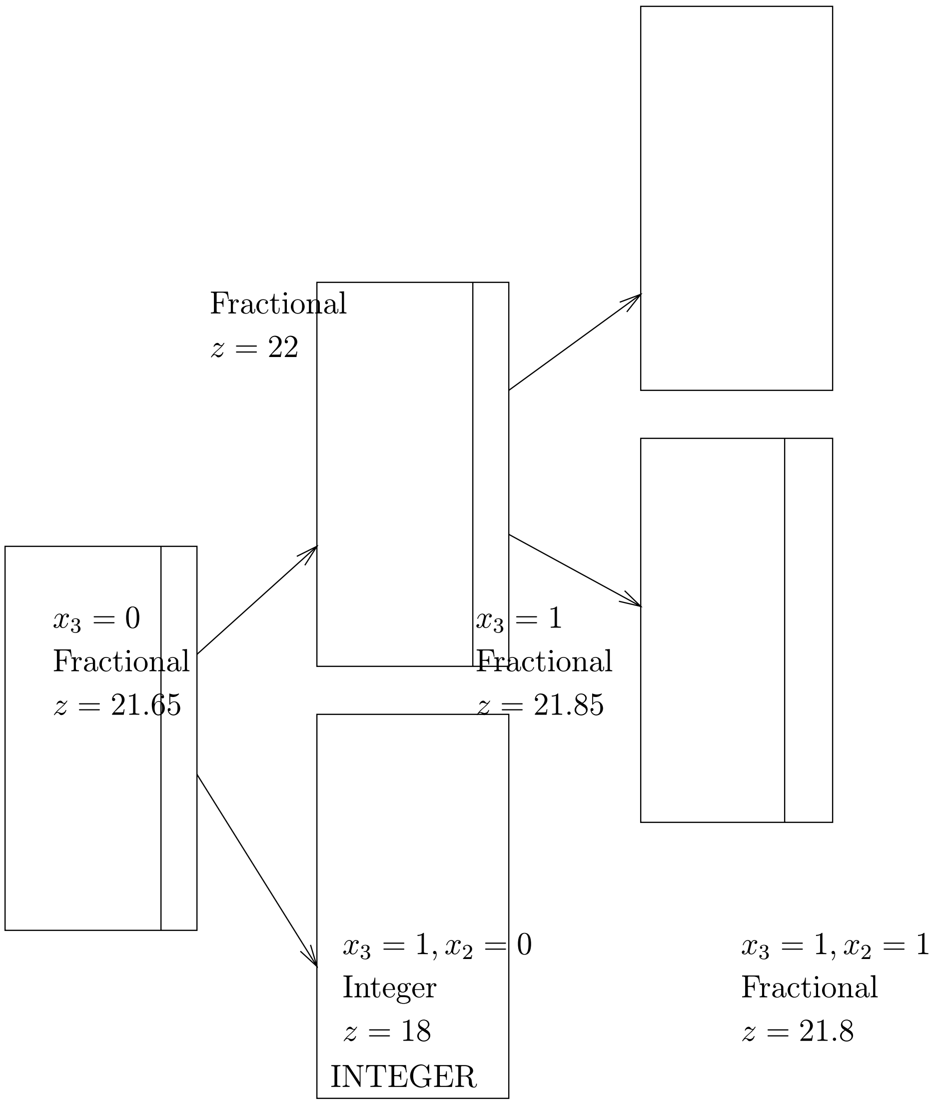

Figure 3: Second Branching

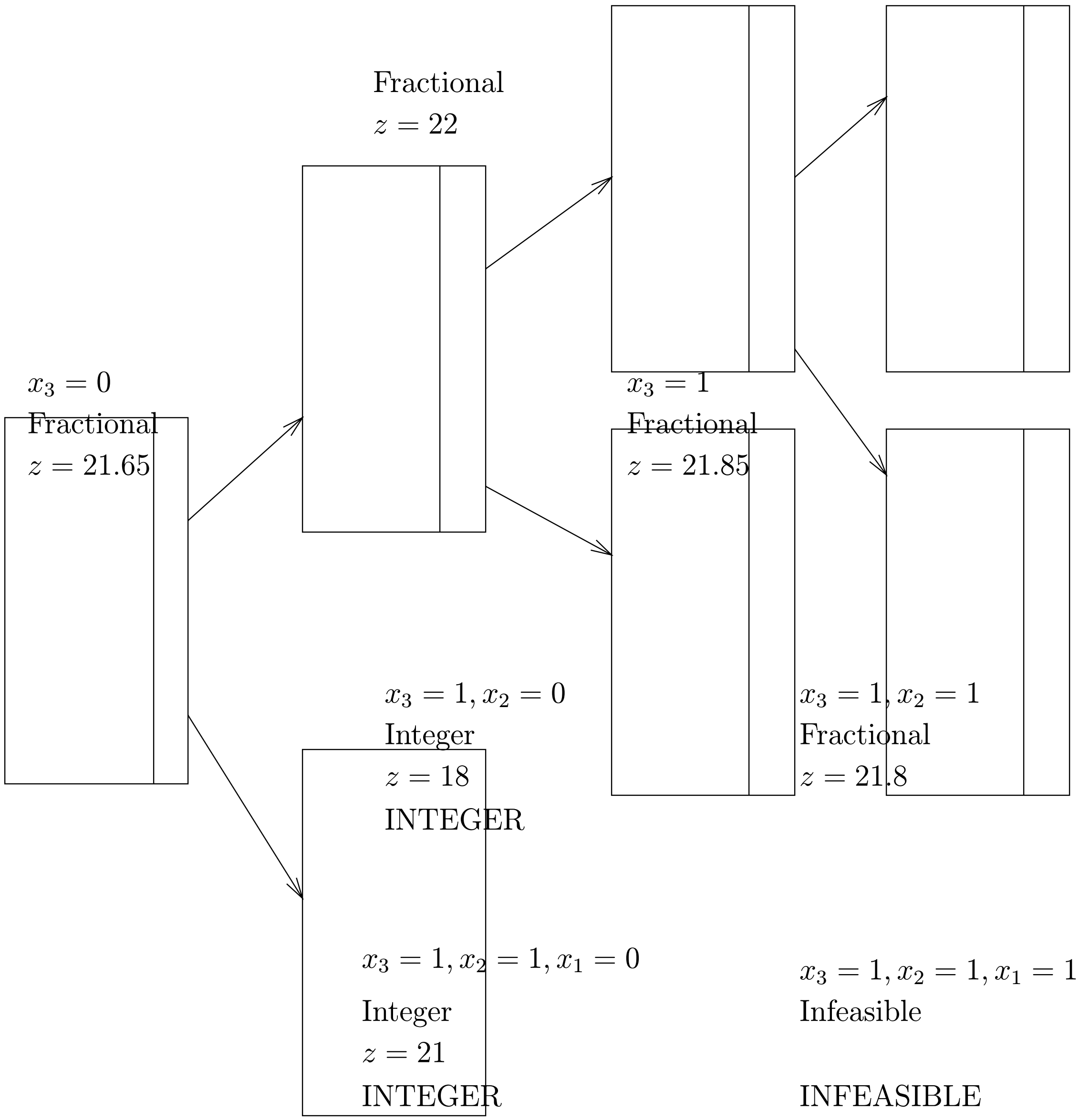

Figure 4: Third Branching

bounding argument and mark it not active. There are no longer any active subproblems, so the optimal solution value is 21.

We have seen all parts of the branch and bound algorithm. The essence of the algorithm is as follows:

1. Solve the linear relaxation of the problem. If the solution is integer, then we are done. Otherwise create two new subproblems by branching on a fractional variable.
2. A subproblem is not active when any of the following occurs:
   (a) You used the subproblem to branch on,
   (b) All variables in the solution are integer,
   (c) The subproblem is infeasible,
   (d) You can fathom the subproblem by a bounding argument.
3. Choose an active subproblem and branch on a fractional variable. Repeat until there are no active subproblems.

That's all there is to branch and bound! Depending on the type of problem, the branching rule may change somewhat. For instance, if x is restricted to be integer (but not necessarily 0 or 1), then if $x = 4.27$ your would branch with the constraints $x \leq 4$ and $x \geq 5$ (not on $x = 4$ and $x = 5$).

In the worst case, the number of subproblems can get huge. For many problems in practice, however, the number of subproblems is quite reasonable.

For an example of a huge number of subproblems, try the following in AMPL:

```
var x0 binary;
var x{1..17} binary;

maximize z: -x0 + sum{j in 1..17} 2 * x[j];
subject to c:  x0 + sum{j in 1..17} 2 * x[j] <= 17;
```

Note that this problem has only 18 variables and only a single constraint. CPLEX looks at 48,619 subproblems, taking about 90 seconds on a Sun Sparc 10 workstation, before deciding the optimal objective is 16. LINGO (another math programming package) on a 16MHz 386 PC (with math coprocessor) looks at 48,000+ subproblems and takes about five hours. The 100-variable version of this problem would take about 10^{29} subproblems or about 3×10^{18} years (at 1000 subproblems per second). Luckily, most problems take far less time.

Exercise 5 (Optional) *Solve the following problem by the branch and bound algorithm. For convenience, always select x_1 as the branching variable when both x_1 and x_2 are fractional.*

$$\begin{array}{ll} \textit{Maximize} & x_1 + x_2 \\ \textit{subject to} & 2x_1 + 5x_2 \leq 16 \\ & 6x_1 + 5x_2 \leq 30 \\ & x_1, x_2 \geq 0 \textit{ and integer.} \end{array}$$

Exercise 6 (Optional) *Repeat the preceeding exercise assuming that x_1 only is restricted to integer values.*

Exercise 7 (Optional) *Consider the following* cargo-loading *problem, where five items are to be loaded on a vessel. The weights w_i and the volume v_i per unit of the different items as well as their corresponding values r_i are tabulated as follows.*

Item i	w_i	v_i	r_i
1	*5*	*1*	*4*
2	*8*	*8*	*7*
3	*3*	*6*	*6*
4	*2*	*5*	*5*
5	*7*	*4*	*4*

The maximum cargo weight and volume are given by $W = 112$ and $V = 109$, respectively. It is required to determine the most valuable cargo load in discrete units of each item. Formulate the problem as an integer program and solve by AMPL.

3.3 Cutting Plane Techniques

There is an alternative to branch and bound called *cutting planes* which can also be used to solve integer programs. The fundamental idea behind cutting planes is to add constraints to a linear program until the optimal basic feasible solution takes on integer values. Of course, we have to be careful which constraints we add: we would not want to change the problem by adding the constraints. We will add a special type of constraint called a *cut*. A cut relative to a current fractional solution satisfies the following criteria:

1. every feasible integer solution is feasible for the cut, and

2. the current fractional solution is not feasible for the cut.

This is illustrated in Figure 5.

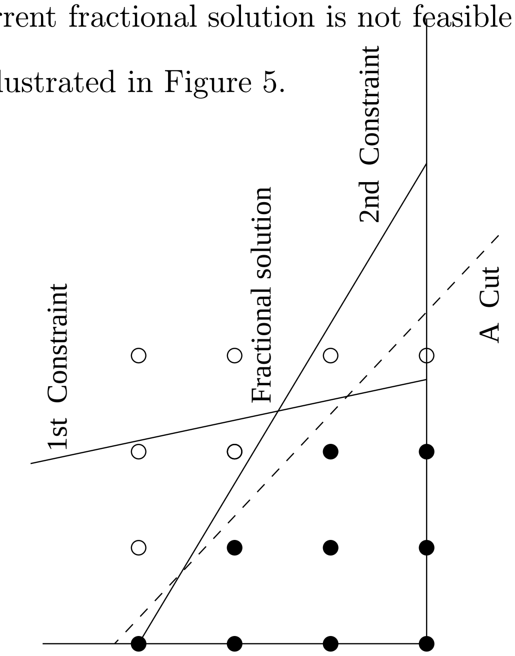

Figure 5: A cut

There are two ways to generate cuts. The first, called Gomory cuts, generates cuts from any linear programming tableau. This has the advantage of "solving" any problem but has the disadvantage that the method can be very slow. The second approach is to use the structure of the problem to generate very good cuts. The approach needs a problem-by-problem analysis, but can provide very efficient solution techniques.

3.3.1 General Cutting Planes

Consider the following integer program:

$$\begin{array}{rrcrcl} \text{Maximize} & 7x_1 & + & 9x_2 & & \\ \text{subject to} & -x_1 & + & 3x_2 & \leq & 6 \\ & 7x_1 & + & x_2 & \leq & 35 \\ & x_1, & & x_2 & \geq & 0 \text{ integer.} \end{array}$$

If we ignore integrality, we get the following optimal tableau (with the updated columns and reduced costs shown for nonbasic variables):

Variable	x_1	x_2	s_1	s_2	$-z$	RHS
x_2	0	1	7/22	1/22	0	7/2
x_1	1	0	-1/22	3/22	0	9/2
$-z$	0	0	28/11	15/11	1	63

Let's look at the first constraint:

$$x_2 + 7/22s_1 + 1/22s_2 = 7/2$$

We can manipulate this to put all of the integer parts on the left side, and all the fractional parts on the right to get:

$$x_2 - 3 = 1/2 - 7/22s_1 - 1/22s_2$$

Now, note that the left hand side consists only of integers, so the right hand side must add up to an integer. Which integer can it be? Well, it consists of some positive fraction minus a series of positive values. Therefore, the right hand side can only be $0, -1, -2, \ldots$; it cannot be a positive value. Therefore, we have derived the following constraint:

$$1/2 - 7/22s_1 - 1/22s_2 \leq 0.$$

This constraint is satisfied by every feasible integer solution to our original problem. But, in our current solution, s_1 and s_2 both equal 0, which is infeasible to the above constraint. This means the above constraint is a cut, called the *Gomory cut* after its discoverer. We can now add this constraint to the linear program and be guaranteed to find a different solution, one that might be integer.

We can also generate a cut from the other constraint. Here we have to be careful to get the signs right:

$$x_1 - 1/22s_1 + 3/22s_2 = 9/2$$
$$x_1 + (-1 + 21/22)s_1 + 3/22s_2 = 4 + 1/2$$
$$x_1 - s_1 - 4 = 1/2 - 21/22s_1 - 3/22s_2$$

gives the constraint

$$1/2 - 21/22s_1 - 3/22s_2 \leq 0.$$

In general, let $\lfloor a \rfloor$ be defined as the largest integer less than or equal to a. For example, $\lfloor 3.9 \rfloor = 3$, $\lfloor 5 \rfloor = 5$, and $\lfloor -1.3 \rfloor = -2$.

If we have a constraint

$$x_k + \sum a_i x_i = b$$

with b not an integer, we can write each $a_i = \lfloor a_i \rfloor + a'_i$, for some $0 \leq a'_i < 1$, and $b = \lfloor b \rfloor + b'$ for some $0 < b' < 1$. Using the same steps we get:

$$x_k + \sum \lfloor a_i \rfloor x_i - \lfloor b \rfloor = b' - \sum a'_i x_i$$

to get the cut

$$b' - \sum a'_i x_i \leq 0.$$

This cut can then be added to the linear program and the problem resolved. The problem is guaranteed not to get the same solution.

This method can be shown to guarantee finding the optimal integer solution. There are a couple of disadvantages:

1. Round-off error can cause great difficulties: Is that 3.000000001 really a 3, or should I generate a cut? If I make the wrong decision I could either cut off a feasible solution (if it is really a 3 but I generate a cut) or I could end up with an infeasible solution (if it is not a 3 but I treat it as one).

2. The number of constraints that are generated can be enormous. Just like branch and bound can generate a huge number of subproblems, this technique can generate a huge number of constraints.

The combination of these makes this cutting plane technique impractical by itself. Recently however, more powerful techniques have been discovered for special problem structure. This is the subject of the next section.

3.3.2 Cuts for Special Structure

Gomory cuts have the property that they can be generated for any integer program. Their weakness is their downfall: they do not seem to cut off much more than the linear programming solution in practice. An alternative approach is to generate cuts that are specially designed for the particular application. We saw that in a simple form in the lockbox problem, where we used the constraints $x_{ij} \leq y_j$ because they were stronger than $\sum_i x_{ij} \leq 100y_j$. In this section, we examine the symmetric traveling salesperson problem.

Recall that there is no good, compact formulation for the TSP. Earlier, we generated a formulation as follows:

$$\begin{aligned}
\text{Minimize} \quad & \sum_{ij} c_{ij}x_{ij} \\
\text{subject to} \quad & \sum_{j \neq i} x_{ij} = 2 \text{ for all } i \\
& \sum_{i \in S} \sum_{j \notin S} x_{ij} \geq 2 \text{ for all } S \subset N \\
& x_{ij} \in \{0,1\} \text{ for all } i, j.
\end{aligned}$$

Recall that in the second set of constraints (called *subtour elimination* constraints) there are many, many constraints. One approach is to initially ignore these constraints and simply solve the problem over the first set of constraints. Suppose we have a six node problem as shown in Figure 6.

This problem can be formulated as follows (ignoring subtour constraints):

$$\begin{aligned}
\text{Minimize} \quad & 4x_{12} + 4x_{13} + 3x_{14} + 4x_{23} + 2x_{25} \\
& \quad + 3x_{36} + 4x_{45} + 4x_{46} + 4x_{56} \\
\text{subject to} \quad & x_{12} + x_{13} + x_{14} = 2 \\
& x_{12} + x_{23} + x_{25} = 2 \\
& x_{13} + x_{23} + x_{36} = 2 \\
& x_{14} + x_{45} + x_{46} = 2 \\
& x_{25} + x_{45} + x_{56} = 2 \\
& x_{36} + x_{46} + x_{56} = 2 \\
& x_{ij} \in \{0,1\} \text{ for all } i, j
\end{aligned}$$

Solving the LP relaxation in AMPL gives:

```
set NODES;
set ARCS within {NODES, NODES};
```

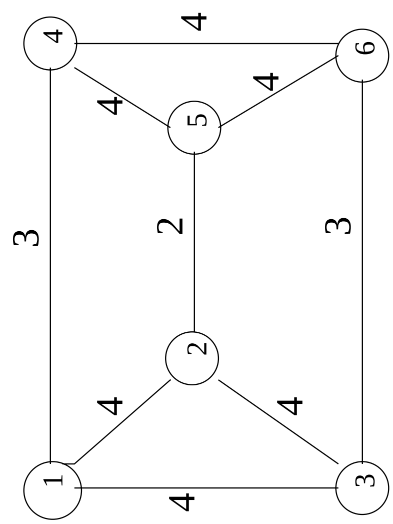

Figure 6: Six node traveling salesperson problem

```
param cost{ARCS};

var x{ARCS} >= 0, <= 1;

minimize z: sum{(i,j) in ARCS} cost[i,j] * x[i,j];
subject to incidence{i in NODES}:
        sum{(i,j) in ARCS} x[i,j] + sum{(j,i) in ARCS} x[j,i] = 2;

--------------------------

 set NODES := 1 2 3 4 5 6;
param: ARCS: cost := 1 2 4, 1 3 4, 1 4 3, 2 3 4, 2 5 2, 3 6 3,
                     4 5 4, 4 6 4, 5 6 4;

--------------------------

ampl: solve;
[...CPLEX messages deleted...]
ampl: display z, x;
z = 20

x :=
```

```
1 2   0.5
1 3   0.5
1 4   1
2 3   0.5
2 5   1
3 6   1
4 5   0.5
4 6   0.5
5 6   0.5
;
```

This solution, while obviously not a tour, actually satisfies all of the subtour elimination constraints. At this point we have three choices:

1. We can do branch and bound on our current linear program, or
2. We can apply Gomory cuts to the resulting tableau, or
3. We can try to find other classes of cuts to use.

In fact, for the traveling salesperson problem, there are a number of other classes of cuts to use. These cuts look at different sets of arcs and try to say something about how a tour can use them. For instance, look at the set of arcs in Figure 7.

It is fairly easy to convince yourself that no tour can use more than 4 of these arcs. This is an example of a broad class of inequalities called *comb* inequalities. This means that the inequality stating that the sum of the x values on these arcs is less than or equal to 4 is a valid inequality (it does not remove any feasible solution to the integer program). Our solution, however, has 4.5 units on those arcs. Therefore, we can add a constraint to get the following formulation and result:

```
set NODES;
set ARCS within {NODES, NODES};
param cost{ARCS};

var x{ARCS} >= 0, <= 1;

minimize z: sum{(i,j) in ARCS} cost[i,j] * x[i,j];
subject to incidence{i in NODES}:
```

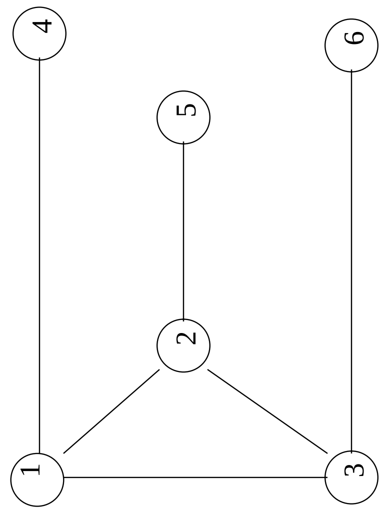

Figure 7: Arc Set for Comb Inequality

```
        sum{(i,j) in ARCS} x[i,j] + sum{(j,i) in ARCS} x[j,i] = 2;
subject to comb:
        x[1,2] + x[1,3] + x[1,4] + x[2,3] + x[2,5] + x[3,6] <= 4;

--------------------

ampl: solve;
[...CPLEX messages deleted...]
ampl: display z, x;
z = 21

x :=
1 2   1
1 3   1
1 4   0
2 3   0
2 5   1
3 6   1
4 5   1
4 6   1
5 6   0
;
```

So, we have the optimal solution.

This method, perhaps combined with branch and bound if the solution is still fractional after all known inequalities are examined, has proven to be a very practical and robust method for solving medium-sized TSPs. Futhermore, many classes of inequalities are known for other combinatorial optimization problems. This approach, seriously studied only for the last ten years or so, has greatly increased the size and type of instances that can be effectively solved to optimality.

Exercise 8 (Optional) *For the knapsack problem*

$$\begin{array}{ll} \textit{Maximize} & 8x_1 + 11x_2 + 6x_3 + 4x_4 \\ \textit{subject to} & 5x_1 + 7x_2 + 4x_3 + 3x_4 \leq 14 \\ & x_j \in \{0,1\} \;\; j = 1, \ldots, 4 \end{array}$$

show that the following inequalities do not remove any feasible solutions:

$$x_1 + x_2 + x_3 \leq 2,$$
$$x_1 + x_2 + x_4 \leq 2.$$

Use these inequalities to solve the LP relaxation of the problem by AMPL (i.e., with the 0-1 restrictions on x_j *replaced by* $0 \leq x_j \leq 1$*).*

4 Solutions to Some Optional Problems

Exercise 1: The formulation is:

$$
\begin{array}{lll}
\text{Maximize} & \sum_{j=1}^{10} p_j x_j & \\
\text{subject to} & \sum_{j=1}^{10} x_j & = \; 5 \\
& x_2 - x_3 & \le \; 0 \\
& x_1 + x_7 + x_8 & \le \; 2 \\
& x_3 + x_5 & \le \; 1 \\
& x_4 + x_5 & \le \; 1 \\
& x_j & \in \; \{0,1\} \;\; j = 1, \ldots, 10.
\end{array}
$$

Exercise 2: Let x_j be a $0,1$ variable which equals 1 if route j is used, 0 otherwise. Then the vehicle routing problem can be formulated as a set covering problem:

$$
\begin{array}{lll}
\text{Minimize} & 10x_1 + 12x_2 + 12x_3 + 13x_4 + 11x_5 & \\
& \quad + 9x_6 + 7x_7 + 8x_8 + 8x_9 & \\
\text{subject to} & x_1 + x_2 + x_3 + x_7 & \ge \; 1 \\
& x_3 + x_4 + x_5 + x_8 & \ge \; 1 \\
& x_1 + x_4 + x_6 + x_8 & \ge \; 1 \\
& x_1 + x_2 + x_5 + x_6 + x_9 & \ge \; 1 \\
& x_2 + x_3 + x_4 + x_5 + x_6 + x_9 & \ge \; 1 \\
& x_j & = \; 0 \text{ or } 1 \text{ for all } j.
\end{array}
$$

Exercise 3: Let x_j = starting time of operation j.

$$
y_{ij} = \begin{cases} 0 \text{ if operation } j \text{ is performed before } i \\ 1 \text{ if operation } i \text{ is performed before } j. \end{cases}
$$

Then the formulation is:

$$
\begin{array}{rll}
\text{Minimize} & x_8 & \\
\text{subject to} & x_i + p_i \le x_j & \text{for } (i,j) = (1,4),(2,4),(3,6), \\
& & (4,7),(5,7),(5,8),(6,8) \\
& x_7 + p_7 \le d_1 & \\
& \left.\begin{array}{r} My_{ij} + x_i - x_j \ge p_j \\ M(1-y_{ij}) + x_j - x_i \ge p_i \end{array}\right\} & \text{for } (i,j) = (1,2),(1,3),(1,5), \\
& & (1,6),(1,8),(2,3),(2,5),(2,6), \\
& & (2,8),(3,4),(3,5),(3,7),(4,5), \\
& & (4,6),(4,8),(5,6),(6,7),(7,8) \\
& x_j = 0 \text{ or } 1 & \text{for } j = 1,\ldots,8.
\end{array}
$$

The first set of constraints are the precedence constraints and the last two sets are the noninterference constraints.

Exercise 4: First we solve the problem as a linear program with CPLEX. The solution is $x_1 = 3.33333$, $x_2 = x_3 = 0$. Rounding it yields $x_1 = 3$, $x_2 = x_3 = 0$ which fails to satisfy the constraint $6x_1 + 20x_2 + 4x_3 = 20$.

In fact, the only feasible integer solution is

$$x_1 = 2, x_2 = 0, x_3 = 2,$$

and it cannot be obtained by simple rounding.

Exercise 5: The enumeration tree is given in Figure 8.

Three optimum integer solutions were found, namely

$$\begin{cases} x_1 = 3 \\ x_2 = 2 \end{cases} \quad \begin{cases} x_1 = 4 \\ x_2 = 1 \end{cases} \quad \begin{cases} x_1 = 5 \\ x_2 = 0 \end{cases}$$

each with value $z = 5$.

Exercise 6

From the enumeration tree of Exercise 5, we see that only one branching is necessary since both subproblems z_2 and z_3 are inactive by reason of integrality. The better solution is $x_1 = 4, x_2 = 1.2$ with value $z_3 = 5.2$.

Exercise 7

```
CPLEX> display problem
Maximize
 obj: 4 X1 + 7 X2 + 6 X3 + 5 X4 + 4 X5
Subject To
 c1: 5 X1 + 8 X2 + 3 X3 + 2 X4 + 7 X5 <= 112
 c2: X1 + 8 X2 + 6 X3 + 5 X4 + 4 X5 <= 109
Bounds
      X1 >= 0
      X2 >= 0
      X3 >= 0
      X4 >= 0
      X5 >= 0
Integers
 X1  X2  X3  X4  X5
CPLEX> opt
No MIP presolve or aggregator reductions.
Presolve time =    0.00 sec.

        Nodes                                              Cuts/
   Node  Left     Objective  IInf  Best Integer     Best Node    ItCnt

      0     0      153.6087     2                    153.6087        4
*     1     1      151.0000     0      151.0000      153.3333        5
```

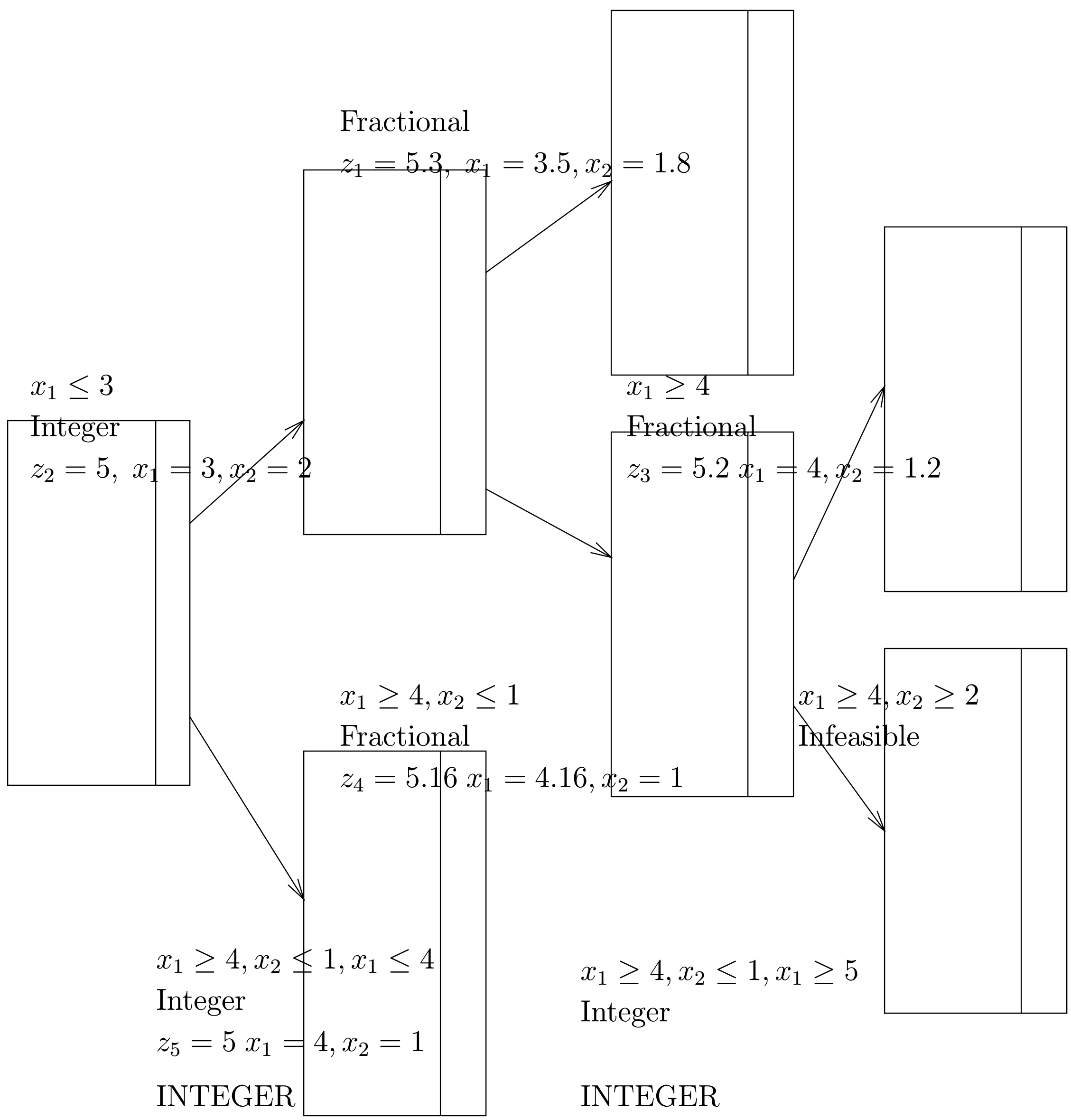

Figure 8: Enumeration tree

```
Integer optimal solution:  Objective =    1.5100000000e+02
Solution time =    0.00 sec.    Iterations = 10   Nodes = 6

CPLEX> display solution -
Variable Name           Solution Value
X1                           14.000000
X4                           19.000000
All other variables in the range 1-5 are zero.
```

Exercise 8

Since $x_1 = x_2 = x_3 = 1$ does not satisfy the knapsack constraint $5x_1 + 7x_2 + 4x_3 + x_4 \leq 14$, it follows that $x_1 + x_2 + x_3 \leq 2$ must hold for all solutions of the knapsack problem.

Similarly for $x_1 + x_2 + x_4 \leq 2$.

```
CPLEX> display problem -
Maximize
 obj: 8 X1 + 11 X2 + 6 X3 + 4 X4
Subject To
 c1: 5 X1 + 7 X2 + 4 X3 + 3 X4 <= 14
 c2: X1 + X2 + X3 <= 2
 c3: X1 + X2 + X4 <= 2
Bounds
 0 <= X1 <= 1
 0 <= X2 <= 1
 0 <= X3 <= 1
 0 <= X4 <= 1
CPLEX> opt
No LP presolve or aggregator reductions.
Presolve time =    0.00 sec.

Iteration Log . . .
Iteration:     1    Objective     =            11.000000

Primal - Optimal:  Objective =    2.1000000000e+01
Solution time =    0.00 sec.    Iterations = 5 (0)

CPLEX> display solution -
Variable Name           Solution Value
X2                            1.000000
X3                            1.000000
X4                            1.000000
All other variables in the range 1-4 are zero.
```