1 Defining Integer Programming Problems

- We will deal with linear constraints.
- The abbreviation MIP stands for "mixed integer program", IP for "integer programming" and BIP for "binary integer programming". More definitions and notes can be found in Chapter 11 of our course textbook, Introduction to Operations Research, 8th Ed., by Hillier and Lieberman. Page numbers given below refer to this text.
- Consider the same problem formulation we've seen before:

$$\begin{align*}
\text{max } f(x) \\
\text{Subject to} \\
Ax &\leq b \\
x_i &\geq 0, \forall i \\
x_i &\in I^+ 
\end{align*}$$

That last condition, that every $x_i$ be an integer, is a headache that makes things much more difficult. One intuitive way of solving integer programming is to use simplex method and then round the solution to the nearest integer. But this may not work as sometimes the rounded solution may not be feasible.

Often we are interested in a special case, such as $x_i \in \{0, 1\}$, where the decision variable corresponds to a yes/no decision. (Text, p. 479: this is a binary integer programming problem, BIP.) Should I turn on the power? Should I build this house? Should I ship this product?. Integers arise where divisibility is not possible: people, airplanes, stores.

- **Either-or constraints:** consider the case where a choice can be made between two constraints, so that only one must hold; the other can hold as well but is not required to do so. For example, there may be a choice as to which of two resources to use for a certain purpose, so that it is necessary for only one of the two resource constraints to hold mathematically. (See also the text, p. 487.) Then this a logical OR of the two constraints.
Either \( 3x_1 + 2x_2 \leq 18 \) \hspace{1cm} (1.1)
...or \( x_1 + 4x_2 \leq 16 \) \hspace{1cm} (1.2)

This can be handled by adding a penalty \( M \), where \( M \) is a large positive number. Adding \( M \) to a constraint has the effect of eliminating it, because it would be satisfied automatically by any solutions that satisfy the other constraint. The example above then becomes:

Either \( 3x_1 + 2x_2 \leq 18 \) \hspace{1cm} (1.3)
\[ x_1 + 4x_2 \leq 16 + M \] \hspace{1cm} (1.4)
...or \( 3x_1 + 2x_2 \leq 18 + M \) \hspace{1cm} (1.5)
\[ x_1 + 4x_2 \leq 16 \] \hspace{1cm} (1.6)

This can also be written, for some \( y \in 0, 1 \):

Either \( 3x_1 + 2x_2 \leq 18 + My \) \hspace{1cm} (1.7)
\[ x_1 + 4x_2 \leq 16 + M(1 - y) \] \hspace{1cm} (1.8)

- A similar method can be used in the case where \textbf{K out of N constraints must hold}, but any K out of N can be chosen. Here the problem includes choosing the optimal K constraints. In the notation let \( N = n \), and:

\[
\begin{align*}
    f_1(x_1...x_n) & \leq d_1 + My_1 \\
    f_2(x_1...x_n) & \leq d_2 + My_2 \\
    & \quad \text{......} \\
    f_N(x_1...x_n) & \leq d_N + My_N \\
\end{align*}
\]

New constraint: \( \sum_{i=1}^{N} y_i = N - k \) \\
\( y_i \in \{0, 1\} \)

- Functions with \textbf{N possible values, or multiple option outcomes}, can be handled in the following way. The function \( f \) is required to take on one of \( N \) possible values.

2
Then:

\[
\begin{align*}
  f(x) &\in \{d_1, d_2, ..., d_N\} \\
  f(x) &= \sum_{i=1}^{N} d_i y_i \\
  \sum_{i=1}^{N} y_i &= 1, \text{ } y_i \text{ is binary.}
\end{align*}
\]

- **Fixed charge problems** occur when there is some fixed setup charge for an activity, such as constructing a factory. After setup the variable cost of continuing operations is related to the level of activity there. The total cost of each activity \( j \) can be represented by a function of the form:

\[
f_j(x_j) = \begin{cases} 
  k_j + c_j x_j & \text{if } x_j > 0 \\
  0 & \text{if } x_j = 0
\end{cases}
\]

The \( k_j \)s are the setup costs, \( c_j \)s are the incremental costs for each additional unit of activity/production, and \( x_j \)s denote the level of activity \( j \) \( (x_j \geq 0) \). To convert this fixed charge problem to an MIP format, we pose \( n \) questions that must be answered yes or no: should activity \( j \) be undertaken? Each of these decisions is represented by auxiliary binary variable \( y_j \), giving the objective \( c_j x_j + k_j y_j \) for each activity, and:

\[
y_j = \begin{cases} 
  1 & \text{if } x_j > 0 \\
  0 & \text{if } x_j = 0
\end{cases}
\]

To ensure \( y \) is 1 when \( x \) is nonzero, also add the constraint \( x_j \leq M y_j \), where \( M \) is a large number that exceeds the maximum feasible value of any \( x_j \). Then the MIP formulation of the fixed charge problem is:
\[ \text{Min } Z = \sum_{i=1}^{N} (c_j x_j + k_j y_j) \]

S.T. \( x_j \geq 0 \)

\( x_j - M y_j \leq 0 \)

...and original constraints on \( x_j \)

\[ y_j = \begin{cases} 
1 & \text{if } x_j > 0 \\
0 & \text{if } x_j = 0 
\end{cases} \]

- **Binary representation** can be used to convert multi-valued general integer variables to sets of binary integer variables. For example,

\[
1 \leq x_i \leq 10 \\
x_i \in \mathbb{Z} \\
\text{Let } x_i = Z_1 + 2Z_2 + 4Z_3 + 8Z_4 \\
Z_j \in [0, 1] \\
\text{Subject to} \\
Z_1 + Z_2 + Z_3 + Z_4 \geq 1 \\
Z_1 + 2Z_2 + 4Z_3 + 8Z_4 \leq 10
\]

## 2 Solution Techniques

- In some cases there are just a few solutions, so you can try them all.
- You can try taking away the integer constraints: this is called an **LP relaxation**. The idea is to switch to real solutions, until you stumble upon an integer solution. For some types of LP problems, the optimal solution is fortuitously an integer by nature. Such problems are a good fit for the LP relaxation technique. An example is the minimum cost flow problem with integer parameters, and its special cases: the transportation problem, the assignment problem, the shortest path problem, and the maximum flow problem. These problems have a special structure that ensures every basic feasible solution is an integer.
- Rounding an LP problem to the nearest integer solution is sometimes acceptable. But rounding is not reliable in general for IP problems. Counterexamples include (i) when the nearest integer happens to lie outside of the feasible region; and, (ii) when the best noninteger solution lies closer to a nonoptimal integer solution than to the optimal integer
solution. These cases are illustrated in the text, pp. 503-504.

2.1 Branch and Bound Method

Special IP solution techniques exist for cases where those simpler methods do not apply. The most popular IP algorithms are called **branch and bound** methods, analogous to "divide and conquer." A basic branch and bound approach has the following steps for a BIP maximization problem:

**Step 0.** Initialize the problem. "Divide": initialize a binary search tree, also called an enumeration tree. (See pp. 515 to 521 for illustrations of these trees.) "Conquer": \( Z^* = -\infty \).

**Step 1.** Branch: select a subproblem, and turn it into two new subproblems by letting \( x_i = 0, x_i = 1 \).

**Step 2.** Bound: solve the two problem relaxations, or obtain a bound on how good its best feasible solution can be. This typically involves an LP relaxation to compute the result quickly.

**Step 3.** Fathom: here "to fathom" means "to dismiss a branch from further consideration." There are three ways a branch can be fathomed. (1) The branch’s LP relaxation may have no feasible solutions. (2) If \( Z \leq Z^* \), where \( Z \) is its LP relaxation result, it is not optimal can can be dismissed. (3) If the LP result for the subproblem has an integer solution, then no more work needs to be done on this branch. If this integer solution is better than the largest solution found on other branches, then it becomes the new candidate solution to the entire problem.

Example:
Max \( Z = 9x_1 + 5x_2 + 6x_3 + 4x_4 \)
S.T. (1) \( 6x_1 + 3x_2 + 5x_3 + 2x_4 \leq 10 \)
(2) \( x_3 + x_4 \leq 1 \)
(3) \( -x_1 + x_3 \leq 0 \)
(4) \( -x_2 + x_4 \leq 0 \)
(5) \( x_j \) is binary \( \forall j \)

Branch: try two subproblems, \( x_1 = 0, x_1 = 1 \).
(A) For the \( x_1 = 0 \) branch, Max \( Z = 9 \cdot 0 + 5x_2 + 6x_3 + 4x_4 \)
S.T. (1) \( 0 + 3x_2 + 5x_3 + 2x_4 \leq 10 \)
...and so on for the other constraints.
(B) The \( x_1 = 1 \) branch is similar.
We solve each LP relaxation, and find
(A) gives (0,1,0,1), Z = 9.
(B) gives (1, 4/5, 0, 4/5), Z = 16\frac{1}{2}.
Now we know the problem is bounded by 16\frac{1}{2}, and the best candidate integer solution at this step is Z = 9. So we have fathomed branch (A), but we are not done with branch (B).
At this point, create two subproblems at branch (B) with \( x_2 = 0, \) \( x_2 = 1 \) and continue.

- Stop when all branches are fathomed. Branches are or are not good candidates. i.e. we either have no solution or an integer solution or a sub-optimal solution. Note: adding an integer constraint can never make the solution better.

- What about the case of integers, not binary variables? Two options are, (1) to recode it in binary, as we showed at the end of Section 1, or (2) judge how to split at the branching step by looking at the LP solution and the feasible region. For \( 0 \leq x_i \leq b_j \), split \( x_j \leq \lceil x^*_j \rceil \) and \( x_j \geq \lfloor x^*_j \rfloor \). See the text, p. 516 for more details.

Example:
Max \( Z = 220x_1 + 80x_2 \)
S.T. \( 5x_1 + 2x_2 \leq 16 \)
\( 2x_1 - x_2 \leq 4 \)
\( -x_1 + 2x_2 \leq 4 \)
\( x_1, x_2 \geq 0 \) are integers

Optimal noninteger solution for this problem: \( (2.6, 1.3) \), at which \( Z = 693.3 \).
Iteration 1: \( \lceil x^*_1 \rceil = 2 \). Solve with constraint \( x_1 \leq 2 \) in subproblem 1, and \( x_1 \geq 3 \) in subproblem 2. Solving problem 1 gives an integer solution.

2.2 Branch and Cut
The branch and bound method becomes slow when problems reach 100 variables or so. As an alternative, the branch and cut approach was proposed. Branch and cut methods in general combine automatic problem preprocessing, the generation of cutting planes, and smart branch and bound techniques (p. 522). Here we discuss BIP preprocessing, which itself has three categories:

1. **Fixing variables.** Identify variables that can be fixed at one of their possible values, 0 or 1, because the other cannot possibly be part of a solution which is both feasible and optimal. Example: given \( 3x_1 \leq 2 \), then since all variables are binary we conclude that \( x_1 = 0 \).

2. **Eliminate redundant constraints.** Example: we saw above that \( x_1 = 0 \). Then if we are given the additional constraint \( 3x_1 + x_2 \leq 2 \), we know it is superfluous and can remove it.
3. **Tighten constraints.** Tighten some constraints in a way that reduces the feasible region for the LP relaxation without eliminating any feasible solutions for the BIP problem. Example:

Max \( Z = 3x_1 + 2x_2 \)
S.T. \( 2x_1 + 3x_2 \leq 4 \)

Since the variables are binary, this constraint is really just saying \( x_1 + x_2 \leq 1 \). So replace the original with this second way of writing it, and the result will be a nice, clean integer solution. It is worth having a tightening routine at the beginning of your program to clean up constraints like this. Cutting plane methods are based on this idea. Theory involves identifying extra constraints that bound relaxation without removing true solutions.

**Constraint Programming.**

This is one other technique for integer programming. This is good for
- Writing large number of constraints in compact way.
- Finding feasible solutions efficiently.
- Searching for good flexible solution.

Say we have a constraint "All \( x_i \) values must be different."
We can do this by setting up

\[
x_{ij} = 1 \text{ if } x_j = x_i
\]

forcing \( \sum_{j=1}^{n} x_{ij} = 1 \) and \( \sum_{i=1}^{n} x_{ij} = 1 \)

There are \( 2n \) constraints, \( n^2 \) variables

\[
x_{ij} = \begin{cases} 
1 & \text{if } x_j = x_i \\
0 & \text{otherwise}
\end{cases}
\]

Constraint program keeps simple statement and writes "\( x_i \) values all different" as a single constraint. This is in compact form.

*End of Oct 26 lecture.*