CHAPTER 9 - Integer Programming

Section 9.1: Intro to Integer Programming.

- In some of our projects, we've already encountered a pure integer or mixed integer programming problem. In either case, we call the problem an integer programming problem or IP for short, as opposed to an LP as we're used to.
- An IP for which all decision variables are integers is called a pure IP.
- An IP for which only some of the decision variables are required to be integers is a mixed IP.
- An IP where all decision variables are 0 or 1 is called a 0-1 IP and we'll see (and have seen) that many applications require this constraint. These are a subset of pure IP's.
- Here are a few examples:

Pure IP:

$$\text{Max } Z = 3x_1 + 2x_2$$
$$\text{Subject to}$$
$$x_1 + x_2 \leq 6$$
$$x_1, x_2 \geq 0, \; x_1, x_2 \text{ integer}$$

Mixed IP:

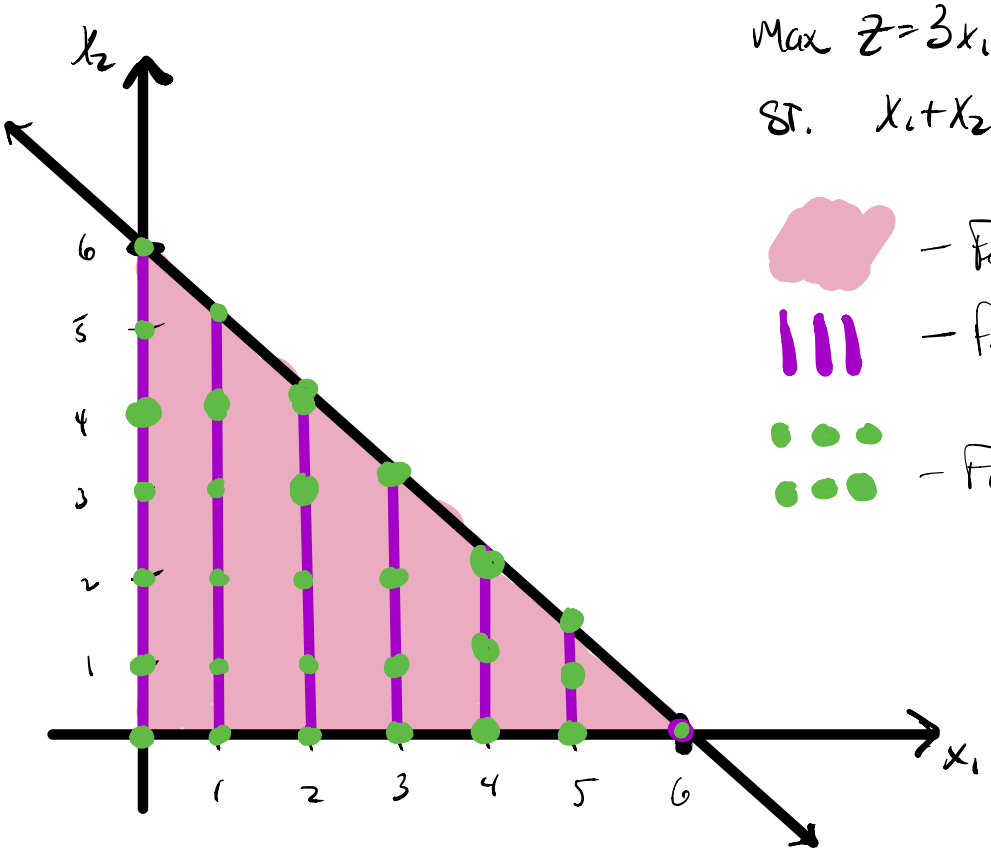
$$\text{Max } Z = 3x_1 + 2x_2$$
$$\text{Subject to}$$
$$x_1 + x_2 \leq 6$$
$$x_1, x_2 \geq 0, \; x_1 \text{ integer}$$

LP

$$\text{Max } Z = 3x_1 + 2x_2$$
$$\text{Subject to}$$
$$x_1 + x_2 \leq 6$$
$$x_1, x_2 \geq 0$$

- For any IP, changing the decision variables to real numbers is called the LP relaxation of the IP. Until now, we've essentially been solving the LP relaxation for any IP and rounding our answers to obtain a feasible solutions. We've seen that this doesn't always produce the best answers.

- Graphical representation of an IP:



Max $Z = 3x_1 + 2x_2$

S.T. $x_1 + x_2 \leq 6$

- Feasible space for LP
- Feasible space for mixed IP
- Feasible space for pure IP.

- In most LP problems considered up to this point contain a set of constraints that all must be satisfied. This implies that the set of constraints live in an intersection or an "and" scenario. Auxillary variables and/or binary variables can be introduced to handle "or" constraints or "if-then" constraints we demonstrate with the following set of problems.

Section 9.2: Formulating Integer Programming Problems

- There are a variety of problems that fall into the class of IP problems. We discuss a few here and demonstrate how to formulate them.

1.) Set Covering Problems: a typical application of this problem is to minimize the number of overlapping services provided to a set of consumers. In this sense, we want to "cover" each consumer by providing "just enough" so that everyone is satisfied.

WS #1 Fire station placement.

2.) Fixed-Charge Problems: problems typically involve some type of economic activity with a fixed and variable cost:

$$C(x) = \begin{cases} F + cx & \text{if } x > 0 \\ 0 & \text{otherwise} \end{cases}$$

F = fixed cost, c = variable cost.

Introducing a boolean variable can convert this piecewise linear (i.e. nonlinear) cost function into one that is purely linear in several other variables.

WS #2 formulating a fixed-charge problem

3.) Capital Budgeting Problems: involve several investment projects for which we select specific investments that will maximize Net Present Value (NPV). Here, we cannot be only partially involved in an investment.

$$x_i = \begin{cases} 1 & \text{select investment } i \\ 0 & \text{do not select investment } i \end{cases}$$

WS #1 work with a budgeting problem.

- In some cases, we may be able to introduce auxillary variables or boolean variables to deal with either-or constraints and if-then constraints. Up until now, we've really only considered and constraints — all constraints must be satisfied.

1.) Either-Or: Suppose we wish to satisfy

$$f(x_1, x_2, \ldots, x_n) \leq 0$$

OR $$g(x_1, x_2, \ldots, x_n) \leq 0$$

Introduce boolean variable $y \in \{0, 1\}$ such that

$$f(x_1, x_2, \ldots, x_n) \leq My$$
$$g(x_1, x_2, \ldots, x_n) \leq M(1-y)$$

where M is sufficiently large. Two cases:

$y=0$: $f(x_1, \dots, x_n) \le 0$

$g(x_1, \dots, x_n) \le M$

↳ $g(x_1, \dots, x_n) \ge 0$ may or may not be satisfied.

$y=1$: $f(x_1, \dots, x_n) \le M$

$g(x_1, \dots, x_n) \le 0$

↳ $f(x_1, \dots, x_n) \le 0$ may or may not be satisfied.

IWS #2 work with an either-or problem

2) If-Then: Suppose we wish to satisfy

If $f(x_1, \dots, x_n) > 0$, then $g(x_1, \dots, x_n) \ge 0$.

To ensure this occurs, we can introduce $y \in \{0,1\}$ such that

$$-g(x_1, \dots, x_n) \le My$$

$$f(x_1, \dots, x_n) \le M(1-y)$$

where M is sufficiently large. Two cases:

$y=0$: $-g(x_1, \dots, x_n) \le 0 \rightarrow g(x_1, \dots, x_n) \ge 0$ ensures

$f(x_1, \dots, x_n) \le M$

↳ Allows $f(x_1, \dots, x_n) > 0$ and if it is

$y=1$: $-g(x_1, \ldots, x_n) \leq M$ ← true no matter what $g \leq 0$ or $g > 0$.

$f(x_1, \ldots, x_n) \leq 0$

↳ Ensures $f(x_1, \ldots, x_n) \neq 0$

WS #3 work with an If-Then Constraint.

Section 9.3: Branch & Bound Method for Pure IPs

- IP algorithms are based on solving a series of continuous LPs using the methods already discussed.
- General steps to take:

1.) Relax the solution space of the IP by deleting the integer restriction (i.e. consider the LP Relaxation).

2.) Solve the LP, and identify the continuous optimum.

Stop → if the solution is integer valued.

3.) Starting from the cont. optimum point, we add special constraints that iteratively modify the LP solution space in a way that eventually renders an optimal integer solution.

WS #1 and #2 work with algorithm.

- Appropriate heuristic (i.e. a practical technique that is not guaranteed to be optimal) to select the next sub problem and its branching variable.

↳ Computational experience shows that these techniques are data dependent.

- Summary of B-and-B method: Assume this is a Max IP.

Step 0: Set a LB of $z = -\infty$. Set $i = 0$.

Step 1: Select LPi, the next subproblem. Solve LPi and attempt to FATHOM it using one of three conditions: (Fathoming / Bounding)

1.) The optimal z-value of LPi cannot yield a better solution than the current LB.

2.) LPi yields a better feasible integer solution than the current LB.

3.) LPi is infeasible.

Two cases will arise:

a.) If LPi is FATHOMED and a better solution is found, update the LB. If all subproblems have been FATHOMED, stop; the LB is optimal. Else, set $i = i+1$ and repeat step 1.

b.) If LPi is not FATHOMED, go to step 2.

Step 2: (Branching) Select one of the integer variables X_j, whose optimum value X_j^* in the LPi solution that is not integer. Create two LP problems that correspond to

$$X_j \leq \lfloor X_j^* \rfloor \quad \text{and} \quad X_j \geq \lfloor X_j^* \rfloor + 1$$

Set $i = i+1$, and go to step 1.

WS 9.3, Part 2 work with another IP problem.

Section 9.4: Branch-and-Bound for Mixed IPs

- The Branch-and-Bound method for Pure IPs can be generally adapted to mixed IPs — only branch on the integer variables.

WS 9.4 Work with a mixed IP problem.

Section 9.5: Solving Knapsack Problems using the B-and-B Method.

- A Knapsack Problem is an IP with a single constraint:

$$\text{Max} \quad z = c_1x_1 + c_2x_2 + \cdots + c_nx_n$$

$$\text{Subject to} \quad a_1x_1 + a_2x_2 + \cdots + a_nx_n \leq b$$

$$x_i \geq 0, \; x_i = \text{integer}$$

Usually $x_i \in \{0,1\}$.

* NPV problems are knapsack problems.

- Here, we can interpret each piece of the problem in the following way:

c_i = benefit obtained from item i

b = the amount of available resource (total)

a_i = the amount of available resource used by item i.

- In any Knapsack IP, we will be faced with solving a series of LP relaxations. Many of these types of LP problems can be solved by inspection of the ratio c_i/a_i

for each decision variable. We rank the variables using this measure:

Rank by: $\frac{c_i}{a_i}$ > 1 means that item is more valuable

< 1 means that item is less valuable.

Then, place the items into the knapsack according to their rank until the resource is used up.

WS 9.5, #1 Walk on solving an LP relaxation problem by rank.

- Now, we can use the rank method for each iteration of the Branch-and-Bound method for solving the original IP.
- When branching in a knapsack problem, the added constraints become very simplified i.e. $x_i = 0$, $x_i = 1$.

WS 9.5, #2. Walk with a small knapsack problem.

Section 9.6: Combinatorial Optimization Problems by the Branch-and-Bound Method.

- We first consider the Traveling Salesperson Problem (TSP). This problem can be formulated as a specific type of Assignment Problem:

Goal: Find the shortest (closed) tour (i.e. Hamiltonian path) in a network of n nodes.

Formulation: Let there be n nodes and let d_{ij} be the distance between nodes i and j. ($d_{ij} = \infty$ if cities i and j are not linked or if $i = j$)

Define the decision variables as

$$x_{ij} = \begin{cases} 1 & \text{if city } j \text{ is reached by city } i \\ 0 & \text{otherwise.} \end{cases}$$

The TSP model is:

$$\text{Minimize} \quad Z = \sum_{i=1}^{n} \sum_{j=1}^{n} d_{ij} x_{ij}$$

Subject to

Define a regular assignment model:

- $\sum_{j=1}^{n} x_{ij} = 1$ for all $i = 1, 2, \ldots, n$
- $\sum_{i=1}^{n} x_{ij} = 1$ for all $j = 1, 2, \ldots, n$
- $x_{ij} \in \{0, 1\}$

New part →

- Solution forms a round-trip.

- The distances, d_{ij}, are typically arranged into a distance matrix, denoted by $[d_{ij}]$, with "∞" on the diagonal. Further, this matrix is symmetric ($d_{ij} = d_{ji}$) if we're talking about distance. In other applications (i.e. costs, c_{ij}), it need not be symmetric.

- There are exactly $(n-1)!$ tours in an n-node TSP (half that if $[d_{ij}]$ is symmetric). If n is small, an exhaustive search may be the best way to solve it. Otherwise, we can use B-and-B with the Hungarian method at each iteration.

- B-and-B Method for TSP:
  - Iterative method with branching and bounding.
  - If the solution to the assignment happens to be a tour, then that solution is optimal. Otherwise, restrictions are placed on variables to force a tour in the next iteration (i.e. branching criteria). Fathoming occurs as normal at each iteration.
  - Quick Breakdown:
    - Solve initial Assignment IP via Hungarian Method.
    - If the solution is a tour, then its optimal. Otherwise the solution contains subtours.
    - Branch on subtour variables, placing M or ∞ in that variables spot. Solve subproblems.
    - Continue to branch and bound until optimal tour is found.

WS 9.6 #1, #2. Work with a TSP.

- LP formulation: We can formulate the last constraint of the TSP with the following set of equations

$$u_i - u_j + n x_{ij} \leq n-1 \quad \text{for } (i \neq j,\ i = 2, \ldots, n,\ j = 2, \ldots, n)$$

where $u_i \geq 0$ are continuous variables. These constraints allow for tours starting at node 1 but eliminates any solution starting at node 1 that includes sub tours. However, this formulation becomes "slow" and unwieldy for large n.

- The Branch-and-Bound method for solving TSP (or any IP) is an exact algorithm, in that it will find the global max/min of the IP but the computational time could be expensive. In fact, in any numerical analysis course, you'll typically study that rate of convergence to this solution as n (the # of variables) approaches ∞.

- For large n, an exact algorithm may not find the solution in a reasonable amount of time prompting the use of heuristics. Heuristic approaches find "good" solutions quickly.

- Two general flavors:

  - Local Search Heuristics: also known as "greedy" algorithms, the search criteria is characterized by improving the z-value at each iteration. If the z-value cannot be improved, stop.

    ↳ Can typically land in a local extrema, rather than a global

    ↳ there is typically no formulation to escape a local min/max.

    ↳ The knapsack algorithm is a greedy algorithm.

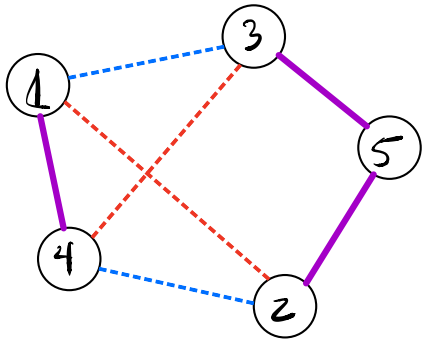
    ↳ Typically involves searching the neighborhood of any decision variable; for n large, use a "random walk" to select neighborhoods

- Three local-search heuristics for TSP include Nearest-Neighbor, Reversal, and Cheapest Insertion:

  - Nearest-Neighbor (NNH): Start with any city, connect it to the closest unlinked city. The recently linked city is connected to the next closest unlinked city. Continue.

  WS 9.6.P2 # 1a. Work with NNH.

  - Reversal (RH): Attempts to improve a selected tour by reversing the order of any sub-path within the tour. For example,



Tour: 1-3-5-2-4-1

Altered Tour: 1-2-5-3-4-1

Is the altered tour better?

WS 9.6.P2 # 1b work with Reversal Heuristic.

- Cheapest Insertion (CIH): Start with any city, join it to its closest neighbor, creating a subtour. Replace an edge of this subtour (i.e. edge (i,j)) with two edges $((i,k)$ and $(k,j))$ where k is not in the current subtour and increases the length of the new subtour by the smallest amount.

WS 9.6.P2 work with CIH problem.

- Meta-Heuristics: motivated to improve local search heuristics by allowing the search criteria to exit local optima by permitting inferior moves.
  - ↳ Needs different benchmarks to terminate. (e.g. # of iterations since last best solution.)
  - ↳