Reading for This Lecture

- Nemhauser and Wolsey Sections II.3.1, II.3.6, II.4.1, II.4.2, II.5.4
- Wolsey Chapter 7
Before delving any deeper into the theory of integer programming, we now turn to the basics of how integer programs are solved in practice.

Computationally, the most important aspect of solving integer programs is obtaining good *bounds* on the value of the optimal solution.

In this lecture, we will motivate this fact by introducing the branch and bound algorithm.

We will then look at various methods of obtaining bounds.

Later, we will examine branch and bound in more detail.
Branch and Bound

- **Branch and bound** is the most commonly-used algorithm for solving MILPs.
- It is a divide and conquer approach.
- Suppose \( S \) is the feasible set for an MILP and we wish to solve \( \max_{x \in S} c^\top x \).
- Consider a partition of \( S \) into subsets \( S_1, \ldots, S_k \). Then
  \[
  \max_{x \in S} c^\top x = \max_{1 \leq i \leq k} \{ \max_{x \in S_i} c^\top x \}
  \]

  - In other words, we can optimize over each subset separately.
- **Idea:** If we can’t solve the original problem directly, we might be able to solve the smaller subproblems recursively.
- Dividing the original problem into subproblems is called **branching**.
- Taken to the extreme, this scheme is equivalent to complete enumeration.
Integer Programs and Disjunction

• The difficulty in solving an integer program arises from the requirement that certain variables take on integer values.

• Such requirements can be described in terms of logical disjunctions, i.e., constraints of the form

\[ x \in \bigcup_{1 \leq i \leq k} X_i \]

• If \( \bigcup_{1 \leq i \leq k} X_i \supset S \), then the disjunction \( \{X_i\}_{i=1}^k \) is said to be valid for an MILP with feasible set \( S \).

• Any MILP can be described by the combination of a finite set of linear inequalities and a finite set of (linear and binary) disjunctions.
Handling Disjunction

- Although we cannot directly solve models including disjunctive constraints, we can handle small numbers of disjunctions in one of two ways.
  - **Enumeration**: Create a separate subproblem for each term of the disjunction and simply solve them each recursively.
  - **Convexification**: (Implicitly) reformulate the problem as $\text{conv}(\bigcup_{1 \leq i \leq k} X_i)$.

- Methods based on the former principle are more straightforward than those based on the latter.

- State-of-the-art algorithms exploit both of these methods.

- For now, we will focus on techniques based on enumeration.
The Importance ofBounding

• The biggest problem with using disjunctions to describe MILPs is that the resulting methods generally require an exponential number of steps.

• *Bounding* enables us to avoid enumerating all possible disjunctions.

• Any feasible solution to a given integer programming problem provides a lower bound \( l(S) \) on the optimal solution value.

• We can use heuristic methods to obtain a lower bound.

• **Idea**: After branching, try to obtain an upper bound \( b(S_i) \) on the optimal solution value for each of the subproblems.

• If \( b(S_i) \leq l(S) \), then we don’t need to consider subproblem \( i \).

• One easy way to obtain an upper bound is by solving the *LP relaxation* obtained by dropping the integrality constraints.

• For the rest of the lecture, assume all variables have finite upper and lower bounds.
LP-based Branch and Bound: Initial Subproblem

• In LP-based branch and bound, we first solve the LP relaxation of the original problem. The result is one of the following:

1. The LP is infeasible ⇒ MILP is infeasible.
2. We obtain a feasible solution for the MILP ⇒ optimal solution.
3. We obtain an optimal solution to the LP that is not feasible for the MILP ⇒ upper bound.

• In the first two cases, we are finished.

• In the third case, we must branch and recursively solve the resulting subproblems.
Branching in LP-based Branch and Bound

• To branch, we identify a valid disjunction that is violated by the solution \( \hat{x} \) to the LP relaxation.

• A systematic method of choosing such a disjunction for branching is called a branching rule.

• Typically, we use binary disjunctions involving linear constraints for this purpose.

• The most commonly used disjunctions are the variable disjunctions, imposed as follows:
  
  – Select a variable \( i \) whose value \( \hat{x}_i \) is fractional in the LP solution.
  – Create two subproblems.
    * In one subproblem, impose the constraint \( x_i \leq \lfloor \hat{x}_i \rfloor \).
    * In the other subproblem, impose the constraint \( x_i \geq \lceil \hat{x}_i \rceil \).

• What does it mean in a 0-1 integer program?
The Geometry of Branching

Figure 1: The original feasible region
Figure 2: Branching on disjunction $x \leq 2$ OR $x \geq 3$
Continuing the Algorithm After Branching

• After branching, we solve each of the subproblems recursively.
• Now we have an additional factor to consider.
• If the optimal solution value to the LP relaxation is smaller than the current lower bound, we need not consider the subproblem further.
• This is the key to the efficiency of the algorithm.
• Terminology
  – If we picture the subproblems graphically, they form a search tree.
  – Each subproblem is linked to its parent and eventually to its children.
  – Eliminating a problem from further consideration is called pruning.
  – The act of bounding and then branching is called processing.
  – A subproblem that has not yet been considered is called a candidate for processing.
  – The set of candidates for processing is called the candidate list.
The Geometry of Branching (cont’d)

Figure 3: Branching on disjunction $y \leq 4$ OR $y \geq 5$ in Subproblem 2
**LP-based Branch and Bound Algorithm**

1. To start, derive a lower bound $L$ using a heuristic method.

2. Put the original problem on the candidate list.

3. Select a problem $S$ from the candidate list and solve the LP relaxation to obtain the bound $b(S)$.
   - If the LP is infeasible $\Rightarrow$ node can be pruned.
   - Otherwise, if $b(S) \leq L$ $\Rightarrow$ node can be pruned.
   - Otherwise, if $b(S) > L$ and the solution is feasible for the MILP $\Rightarrow$ set $L \leftarrow b(S)$.
   - Otherwise, branch and add the new subproblem to the candidate list.

4. If the candidate list is nonempty, go to Step 2. Otherwise, the algorithm is completed.
Branch and Bound Tree

Key

IE418 Lecture 5
Basic Choices in Branch and Bound

• The bounding method(s).

• The rule for selecting the next candidate to process.
  – “Best-first” always chooses the candidate with the highest upper bound.
  – This rule minimizes the size of the tree (*why?*).
  – There may be practical reasons to deviate from this rule.

• The rule for branching.
  – Branching wisely is extremely important.
  – A “*poor*” branching can slow the algorithm significantly.

• We will cover the last two topics in more detail later in the course.
A Thousand Words

B&B tree (None 0.38s )

Figure 4: Tree after 400 nodes
A Thousand Words

B&B tree (None 1.46s)

Figure 5: Tree after 1200 nodes
A Thousand Words

Figure 6: Final tree
Global Bounds

- The pictures show the evolution of the branch and bound process.
- Nodes are pictured at a height equal to that of their lower bound (we are minimizing in this case!!).
  - **Red**: candidates for processing/branching
  - **Green**: branched or infeasible
  - **Turquoise**: pruned by bound (possibly having produced a feasible solution) or infeasible.
- The red line is the level of the current best solution (global upper bound).
- The level of the highest red node is the global lower bound.
- As the procedure evolves, the two bounds grow together.
- The goal is for this to happen as quickly as possible.
Tradeoffs

• We will see that there are many tradeoffs to be managed in branch and bound.

• Note that in the final tree:
  – Nodes below the line were pruned by bound (and may or may not have generated a feasible solution) or were infeasible.
  – Nodes above the line were either branched or were infeasible or generated an optimal solution.

• There is a tradeoff between the goals of moving the upper and lower bounds
  – The nodes below the line serve to move the upper bound.
  – The nodes above the line serve to move the lower bound.

• It is clear that these two goals are somewhat antithetical.

• The search strategy has to achieve a balance between these two antithetical goals.
Tradeoffs in Practice

• In a practical implementation, there are many more choices and tradeoffs than those we have indicated so far.

• The complexity of the problem of optimizing the algorithm itself is immense.

• We have additional auxiliary methods, such as preprocessing and primal heuristics that we can choose to devote more or less effort to.

• We also have the choice of how much effort to devote to choosing a good candidate for branching.

• Finally, we have the choice of how much effort to devote to proving a good bound on the subproblem.

• It is the careful balance of the levels of effort devoted to each of these algorithmic processes the leads to a good algorithmic implementation.