Computational Integer Programming
Universidad de los Andes

Lecture 2

Dr. Ted Ralphs

References 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
- "Noncommercial Software for Mixed-Integer Linear Programming," Linderoth and Ralphs.
- "Computational Issues for Branch-and-Cut Algorithms," by Martin.

Computational Integer Programming

- Computationally, perhaps the most important aspect of solving integer programs is obtaining good *bounds* on the value of the optimal solution.
- To begin our examination of computational methods, we will motivate this fact by introducing the branch and bound algorithm.
- Later, we'll look at various methods of obtaining bounds.
- Finally, we'll talk about specific variants of 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 F is the feasible region for some MILP and we wish to solve $\max_{x \in F} c^\top x$.
- Consider a partition of F into subsets $F_1, \ldots F_k$. Then

$$\max_{x \in F} c^\top x = \max_{\{1 \le i \le k\}} \{\max_{x \in F_i} c^\top x\}$$

.

- 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.

The Importance of Bounding

- For the rest of the lecture, assume all variables have finite upper and lower bounds.
- Any feasible solution to a given integer programming problem provides an lower bound $l(F)$ on the optimal solution value.
- We can use heuristic methods to obtain a lower bound.
- Idea: After branching, try to obtain an *upper bound* $b(F_i)$ on the optimal solution value for each of the subproblems.
- If $b(F_i) \leq l(F)$, 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.

LP-based Branch and Bound

- 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 $\Rightarrow$ MILP is infeasible.
  2. We obtain a feasible solution for the MILP $\Rightarrow$ optimal solution.
  3. We obtain an optimal solution to the LP that is not feasible for the MILP $\Rightarrow$ 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

- The most common way to branch is 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 $x_i \geq \lceil \hat{x}_i \rceil$.
- Such a method of branching is called a *branching rule*.
- Why is this a valid branching rule?
- What does it mean in a 0-1 integer program?

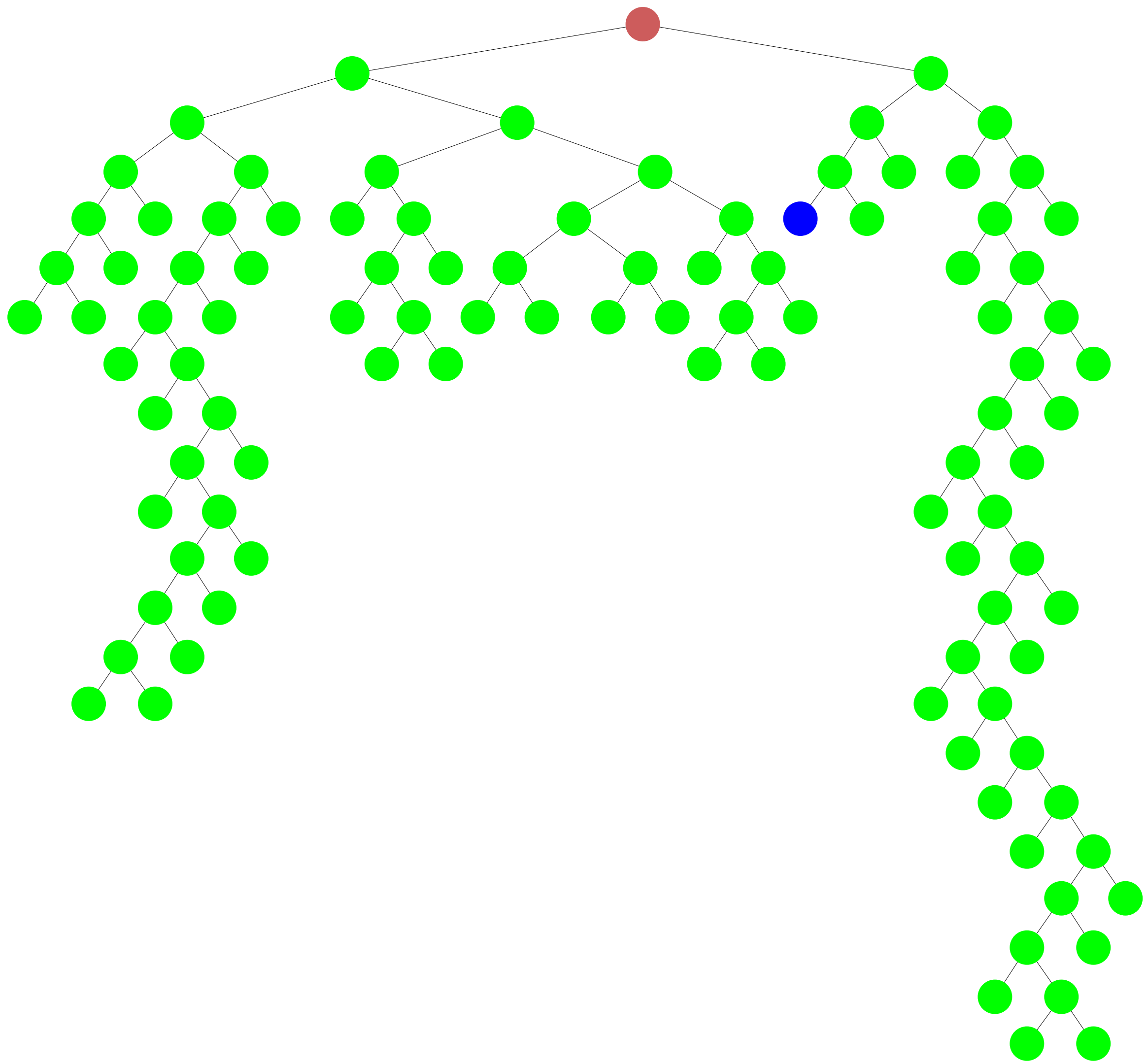
Continuing the Algorithm After Branching

- 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*.

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 $b(S) \le L \Rightarrow$ 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 in nonempty, go to Step 2. Otherwise, the algorithm is completed.

Sample Search Tree



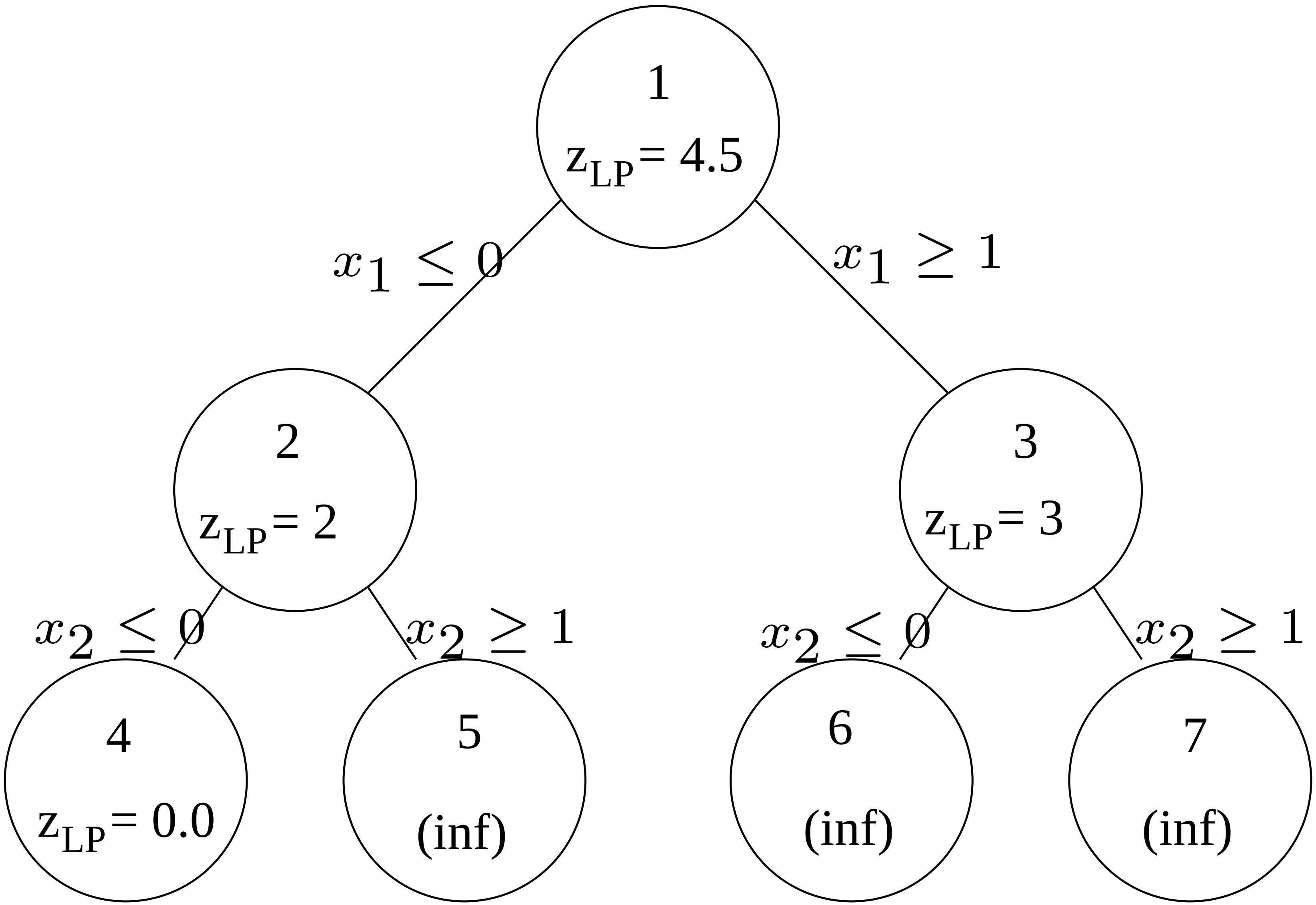
Baby BIP

- Consider a small binary integer program:

$$
\begin{aligned}
\max\, & x_1 + 4x_2 \\
& -2x_1 + 2x_2 \le 1 \\
& x_1 + 2x_2 \le 2.5 \\
& x_1 - 4x_2 \le 0 \\
& x_1, x_2 \in \{0,1\}.
\end{aligned}
\tag{1}
$$

- What is the optimal solution?
- What is the proof?

Branch and Bound Tree



Choices in Branch and Bound

- The bounding method.
- 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.