

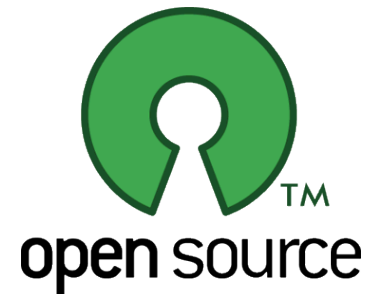
# Computational Integer Programming

## Lecture 8: Branch and Bound

Dr. Ted Ralphs



LEHIGH  
UNIVERSITY



***COR@L***  
COMPUTATIONAL OPTIMIZATION  
RESEARCH AT LEHIGH 

## 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

## Computational Integer Optimization

- We now turn to the details of how integer optimization problems are solved in practice.
- Computationally, the most important aspects of solving integer optimization problems are
  - A method for obtaining good *bounds* on the value of the optimal solution (usually by solving a *relaxation* or *dual*; and
  - A method for generating *valid disjunctions* violated by a given (infeasible) 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.

## Integer Optimization and Disjunction

- As we know, the difficulty in solving an integer optimization problem arises from the requirement that certain variables take on integer values.
- Such requirements can be described in terms of logical *disjunctions*, constraints of the form

$$x \in \bigcup_{1 \leq i \leq k} X_i$$

for  $X_i \subseteq \mathbb{R}^n, i \in 1, \dots, k$ .

- The integer variables in a given formulation may represent logical conditions that were originally expressed in terms of disjunction.
- In fact, the MILP Representability Theorem tells us that any MILP can be re-formulated as an optimization problem whose feasible region is

$$\mathcal{F} = \bigcup_{i=1}^k \mathcal{P}_i + \text{intcone}\{r^1, \dots, r^t\}$$

is the *disjunctive set*  $\mathcal{F}$  defined above, for some appropriately chosen polytopes  $\mathcal{P}_1, \dots, \mathcal{P}_k$  and vectors  $r^1, \dots, r^t \in \mathbb{Z}^n$ .

## Two Conceptual Reformulations

- From what we have seen so far, we have two conceptual reformulations of a given integer optimization problem.
- The first is in terms of *disjunction*:

$$\max \left\{ c^\top x \mid x \in \left( \bigcup_{i=1}^k \mathcal{P}_i + \text{intcone}\{r^1, \dots, r^t\} \right) \right\} \quad (\text{DIS})$$

- The second is in terms of *valid inequalities*:

$$\max \{ c^\top x \mid x \in \text{conv}(\mathcal{S}) \} \quad (\text{CP})$$

where  $\mathcal{S}$  is the feasible region.

- In principle, if we had a method for generating either of these reformulations, this would lead to a practical method of solution.
- Unfortunately, these reformulations are necessarily of exponential size in general, so there can be no way of generating them efficiently.

## Valid Disjunctions

- In practice, we dynamically generate parts of the reformulations (CP) and (DIS) in order to obtain a proof of optimality for a particular instance.
- The concept of *valid disjunction*, arises from a desire to approximate the feasible region of (DIS).

**Definition 1.** Let  $\{X_i\}_{i=1}^k$  be a collection of subset of  $\mathbb{R}^n$ . Then if  $\bigcup_{1 \leq i \leq k} X_i \supseteq \mathcal{S}$ , the disjunction associated with  $\{X_i\}_{i=1}^k$  is said to be *valid* for an MILP with feasible set  $\mathcal{S}$ .

**Definition 2.** If  $\{X_i\}_{i=1}^k$  is a disjunction valid for  $\mathcal{S}$  and  $X_i$  is polyhedral for all  $i \in \{1, \dots, k\}$ , then we say the disjunction is *linear*.

**Definition 3.** If  $\{X_i\}_{i=1}^k$  is a disjunction valid for  $\mathcal{S}$  and  $X_i \cap X_j = \emptyset$  for all  $i, j \in \{1, \dots, k\}$ , we say the disjunction is *partitive*.

**Definition 4.** If  $\{X_i\}_{i=1}^k$  is a disjunction valid for  $\mathcal{S}$  that is both linear and partitive, we call it *admissible*.

## Valid Inequalities

- Likewise, we can think of the concept of a *valid inequality* as arising from our desire to approximate  $\text{conv}(\mathcal{S})$  (the feasible region of (CP)).
- The inequality denoted by  $(\pi, \pi_0)$  is called a *valid inequality* for  $\mathcal{S}$  if  $\pi^\top x \leq \pi_0 \forall x \in \mathcal{S}$ .
- Note  $(\pi, \pi_0)$  is a valid inequality if and only if  $\mathcal{S} \subseteq \{x \in \mathbb{R}^n \mid \pi^\top x \leq \pi_0\}$ .

## Optimality Conditions

- Let us now consider an MILP  $(A, b, c, p)$  with feasible set  $\mathcal{S} = \mathcal{P} \cap (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p})$ , where  $\mathcal{P}$  is the given formulation.
- Further, let  $\{X_i\}_{i=1}^k$  be a linear disjunction valid for this MILP so that  $X_i \cap \mathcal{P} \subseteq \mathbb{R}^n$  is polyhedral.
- Then  $\max_{X_i \cap \mathcal{S}} c^\top x$  is an MILP for all  $i \in 1, \dots, k$ .
- For each  $i = 1, \dots, k$ , let  $\mathcal{P}_i$  be a polyhedron such that  $X_i \cap \mathcal{S} \subseteq \mathcal{P}_i \subseteq \mathcal{P} \cap X_i$ .
- In other words,  $\mathcal{P}_i$  is a valid formulation for subproblem  $i$ , possibly strengthened by additional valid inequalities.
- Note that  $\{\mathcal{P}_i\}_{i=1}^k$  is itself a valid linear disjunction.
- We will see why there is a distinction between  $X_i$  and  $\mathcal{P}_i$  later on.
- Conceptually, we are combining and relaxing the formulations (CP) and (DIS).



## Optimality Conditions (cont'd)

- From the disjunction on the previous slide, we obtain a relaxation of a general MILP.
- This relaxation yields a practical set of optimality conditions.
- In particular,

$$\max_{i \in 1, \dots, k} \max_{x \in \mathcal{P}_i \cap \mathbb{R}_+^n} c^\top x \geq z_{IP}, \quad (1)$$

which implies that if we have  $x^* \in \mathcal{S}$  such that

$$\max_{i \in 1, \dots, k} \max_{x \in \mathcal{P}_i \cap \mathbb{R}_+^n} c^\top x = c^\top x^*, \quad (\text{OPT})$$

then  $x^*$  must be optimal.

## More on Optimality Conditions

- Although it is not obvious, these optimality conditions can be seen as a generalization of those from LP.
- They are also the optimality conditions implicitly underlying many advanced algorithms.
- There is an associated duality theory that we will see later.
- By parameterizing (1), we obtain a “dual function” that is the solution to a dual that generalizes the LP dual.

## Branch and Bound

- *Branch and bound* is the most commonly-used algorithm for solving MILPs.
- It is a *recursive, divide-and-conquer* approach.
- Suppose  $\mathcal{S}$  is the feasible set for an MILP and we wish to compute  $\max_{x \in \mathcal{S}} c^\top x$ .
- Consider a *partition* of  $\mathcal{S}$  into subsets  $\mathcal{S}_1, \dots, \mathcal{S}_k$ . Then

$$\max_{x \in \mathcal{S}} c^\top x = \max_{\{1 \leq i \leq k\}} \left\{ \max_{x \in \mathcal{S}_i} c^\top x \right\}$$

- 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.

## Branching in Branch and Bound

- Branching is achieved by selecting an admissible disjunction  $\{X_i\}_{i=1}^k$  and using it to partition  $\mathcal{S}$ , e.g.,  $\mathcal{S}_i = \mathcal{S} \cap X_i$ .
- We only consider linear disjunctions so that the subproblem remain MILPs after branching.
- The reason for choosing partitive disjunctions is self-evident.
- The way this disjunction is selected is called the *branching method* and is a topic we will examine in some depth.
- Generally speaking, we want  $x^* \notin \bigcup_{1 \leq i \leq k} X_i$ , where  $x^*$  is the (infeasible) solution produced by solving the *bounding problem* associated with a given subproblem.
- A typical disjunction is

$$X_1 = \{x_j \leq \lfloor x_j^* \rfloor\}, \quad (2)$$

$$X_2 = \{x_j \geq \lceil x_j^* \rceil\}, \quad (3)$$

where  $x^* \in \operatorname{argmax}_{x \in \mathcal{P}} c^\top x$ .

## Bounding in Branch and Bound

- The *bounding problem* is a problem solved to obtain a bound on the optimal solution value of a subproblem  $\max_{\mathcal{S}_i} c^\top x$ .
- Typically, the bounding problem is either a relaxation or a dual of the subproblem (these concepts will be defined formally in Lecture 7).
- Solving the bounding problem serves two purposes.
  - In some cases, the solution  $x^*$  to the relaxation may actually be a feasible solution ( $x^* \in \mathcal{S}$ , in which case  $c^\top x^*$  is a **global lower bound**  $l(\mathcal{S})$ ).
  - *Bounding* enables us to inexpensively a bound  $b(\mathcal{S}_i)$  on the optimal solution value of subproblem  $i$ .
- If  $b(\mathcal{S}_i) \leq l(\mathcal{S})$ , then  $\mathcal{S}_i$  can't contain a solution strictly better than the best one found so far.
- Thus, we may discard or *prune* subproblem  $i$ .

## Constructing a Bounding Problem

- There are many ways to construct a bounding problem and this will be the topic of later lectures.
- The easiest of these is to form the *LP relaxation*  $\max_{\mathcal{P} \cap \mathbb{R}_+^n} c^T x$ , 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  $\Rightarrow$  **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

- In LP-based branch and bound, 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 problem?



# The Geometry of Branching

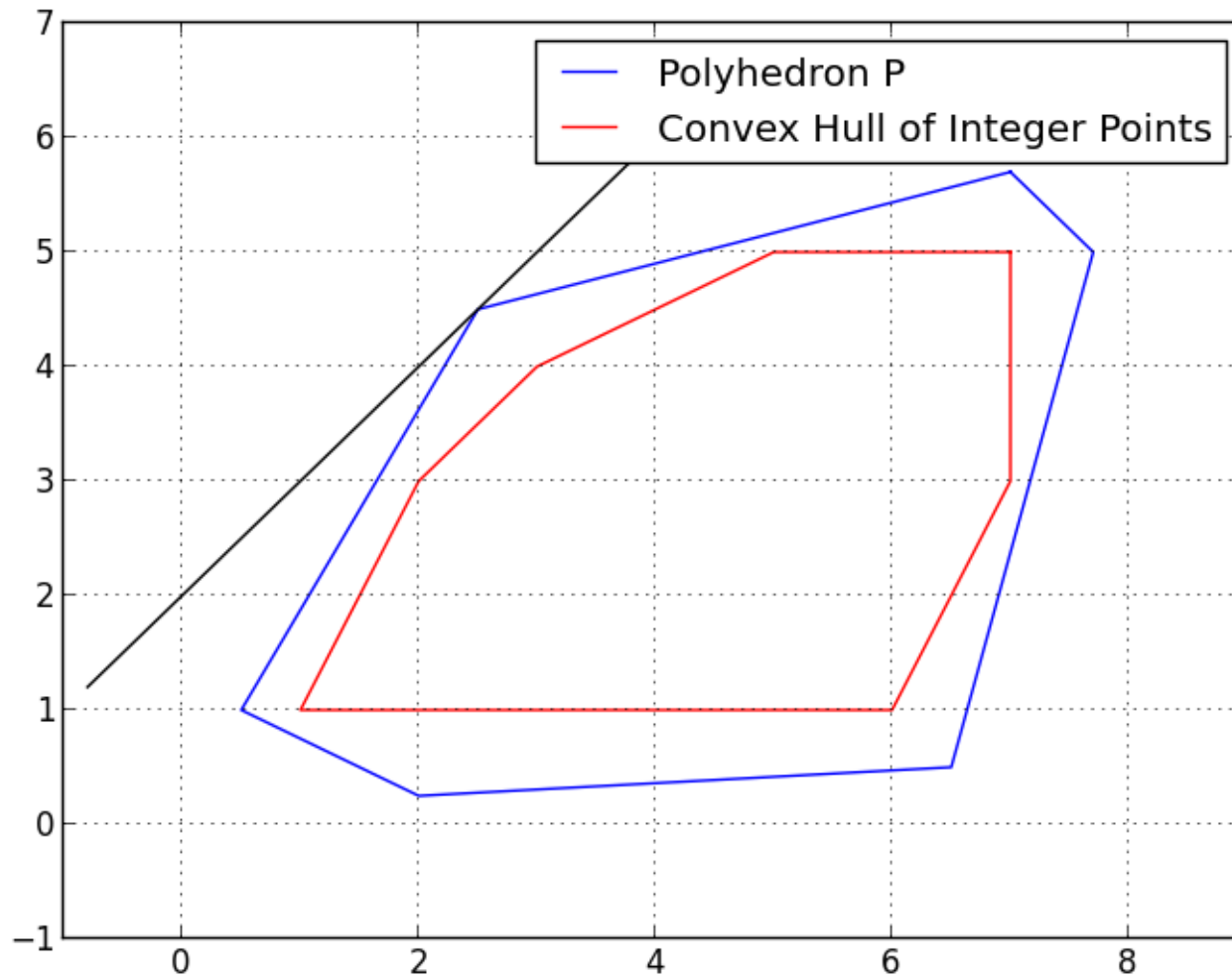


Figure 1: The original feasible region

## The Geometry of Branching (cont'd)

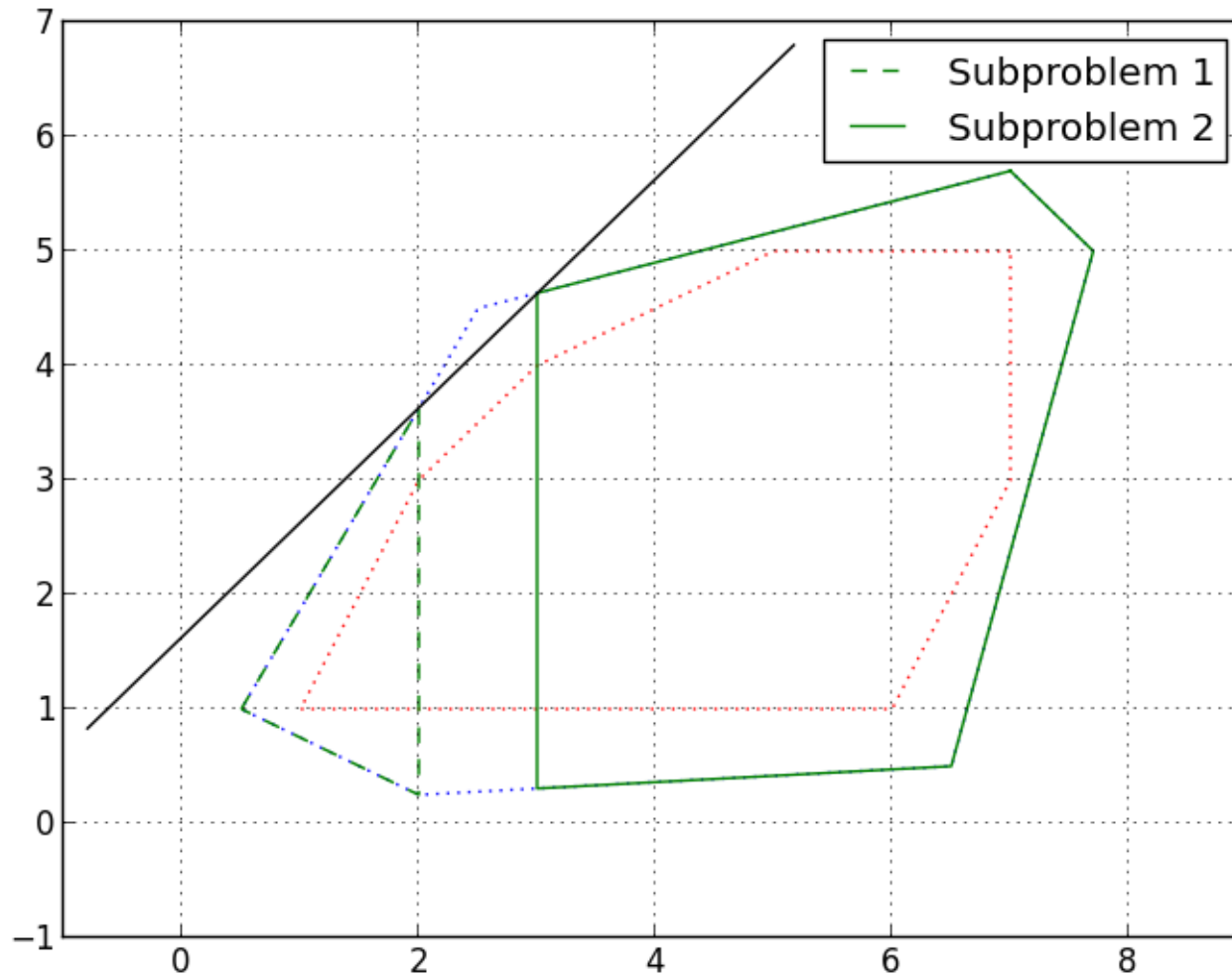


Figure 2: Branching on disjunction  $x_1 \leq 2$  OR  $x_1 \geq 3$

## Continuing the Algorithm After Branching

- After branching, we solve each of the subproblems *recursively*.
- Now we have an additional factor to consider.
- As mentioned earlier, 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

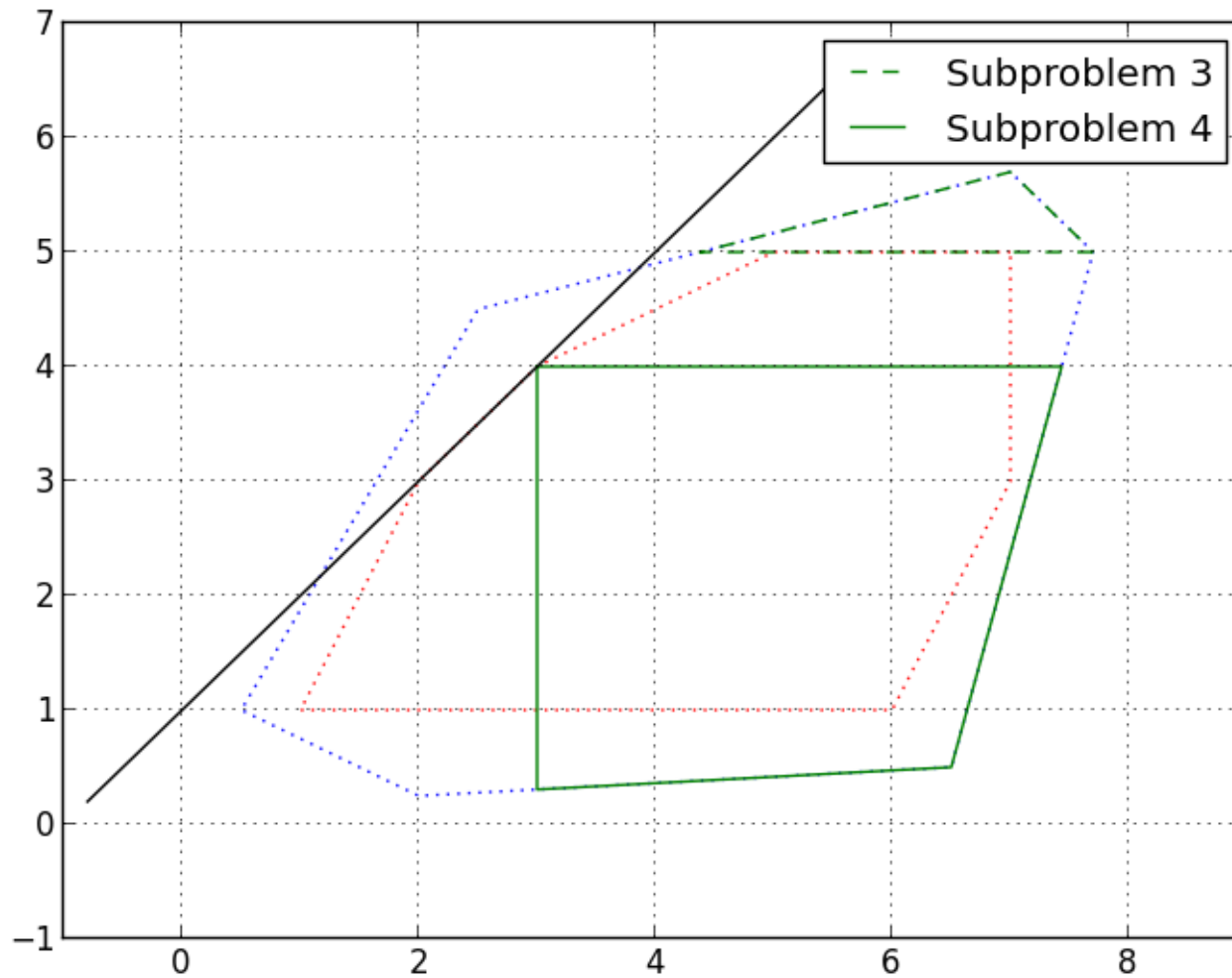
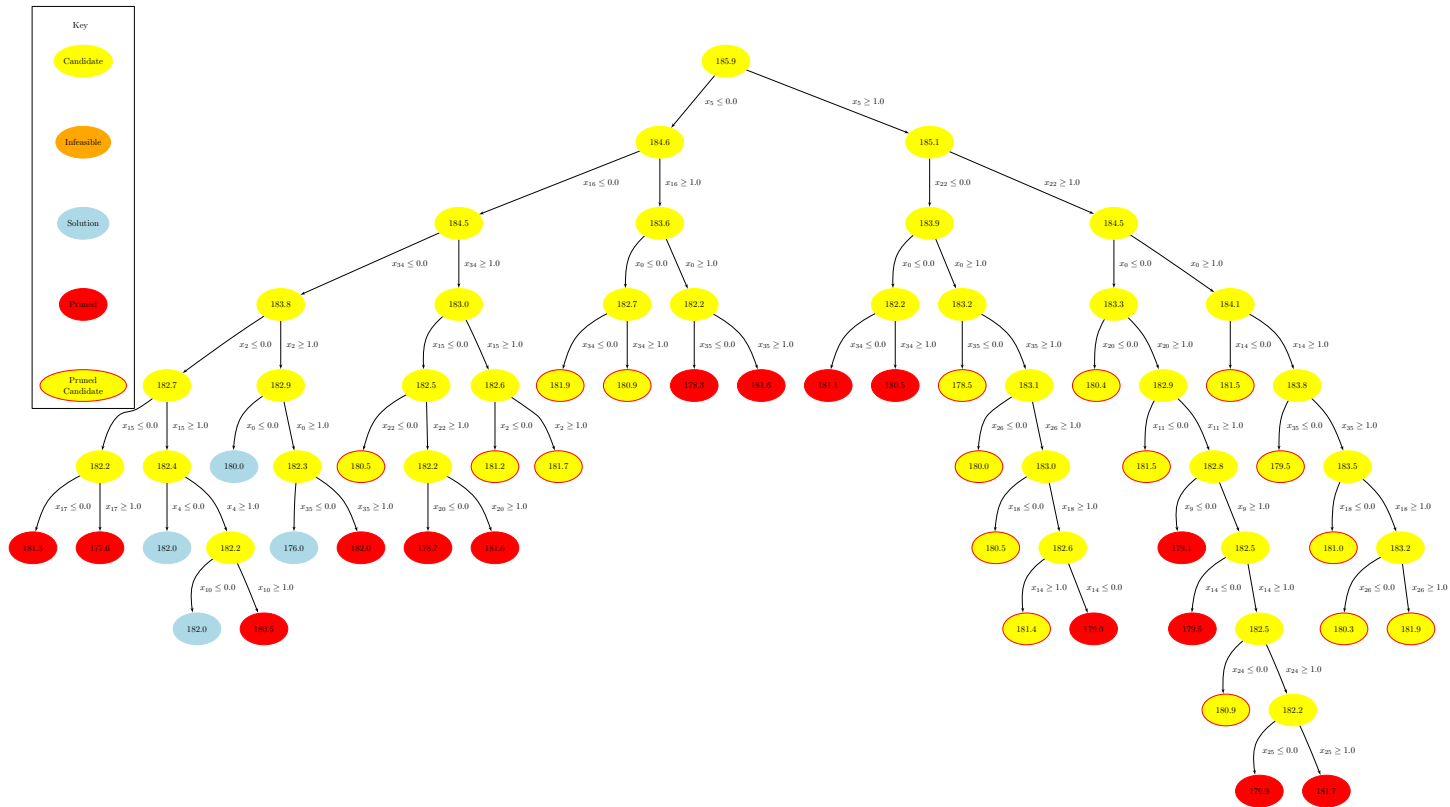


Figure 3: Branching on disjunction  $x_1 \leq 4$  OR  $x_1 \geq 5$  in Subproblem 2

## 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



## Termination Conditions

- Note that although we use multiple disjunctions to branch during the algorithm, the tree can still be seen as encoding a single disjunction.
- To see this, consider the set  $\mathcal{T}$  of subproblems associated with the leaf nodes in the tree.
  - Provided that we use admissible disjunctions for branching, the feasible regions of these subproblems are a partition of  $\mathcal{S}$ .
  - Furthermore, we will see that there exists a collection of polyhedra  $\{\mathcal{P}_i\}_{i \in \mathcal{T}}$ , where
    - \*  $\mathcal{P}_i$  is a formulation for subproblem  $i$ ; and
    - \*  $\{\mathcal{P}_i\}_{i=1}^k$  is admissible with respect to  $\mathcal{S}$ .
- When this disjunction, along with the best solution found so far satisfies the optimality conditions (OPT), the algorithm terminates.
- We will revisit this more formally as we further develop the supporting theory.

## Ensuring Finite Convergence

- For LP-based branch and bound, ensuring convergence requires a convergent branching method.
- Roughly speaking, a convergent branching method is one which will
  - produce a violated admissible disjunction whenever the solution to the bounding problem is infeasible; and
  - if applied recursively, guarantee that at some finite depth, any resulting bounding problem will either
    - \* produce a feasible solution (to the original MILP); or
    - \* be proven infeasible; or
    - \* be pruned by bound.
- Typically, we achieve this by ensuring that at some finite depth, the feasible region of the bounding problem contains at most one feasible solution.
- We will also revisit this result more formally as we develop the supporting theory.



## Algorithmic Choices in Branch and Bound

- Although the basic algorithm is straightforward, the efficiency of it in practice depends strongly on making good algorithmic choices.
- These algorithmic choices are made largely by heuristics that guide the algorithm.
- Basic decisions to be made include
  - The bounding method(s).
  - The method of 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 method of branching.
    - \* Branching wisely is extremely important.
    - \* A “poor” branching can slow the algorithm significantly.
- We will cover the last two topics in more detail in later lectures.

## A Thousand Words

B&B tree (None 0.38s )

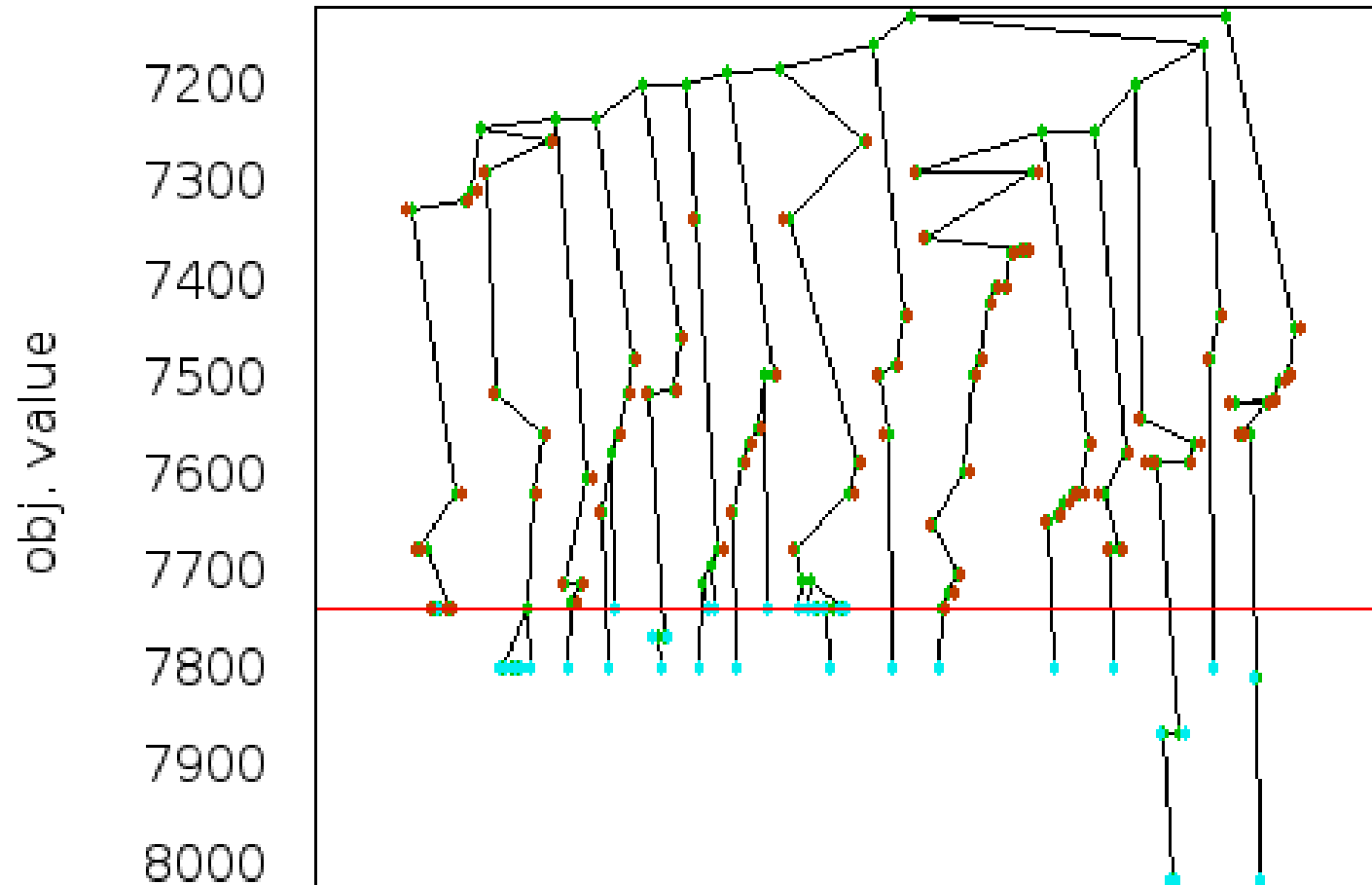


Figure 4: Tree after 400 nodes

Note that we are minimizing here!

## A Thousand Words

B&B tree (None 1.46s )

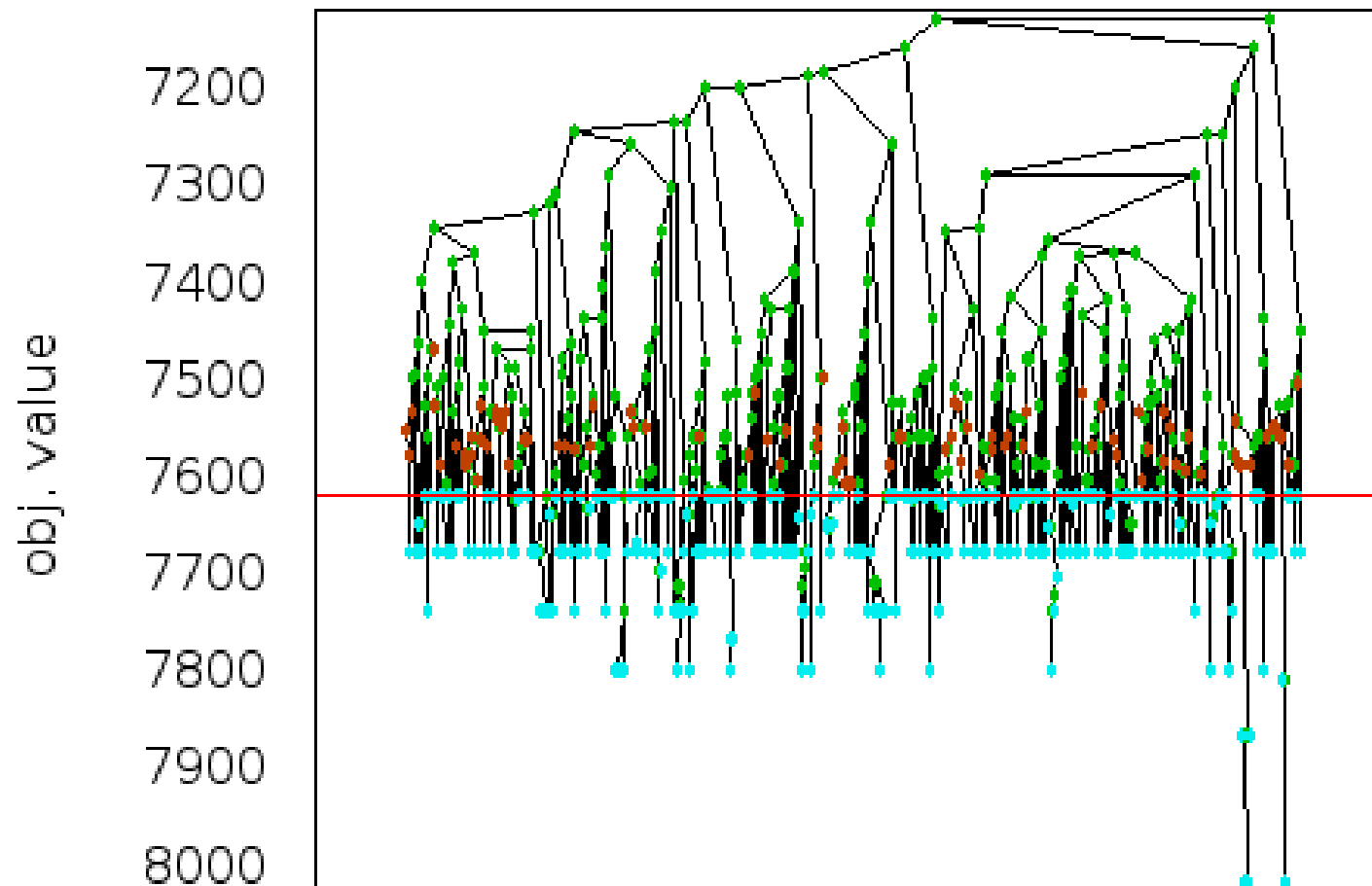


Figure 5: Tree after 1200 nodes

## A Thousand Words

B&B tree (None 1.65s)

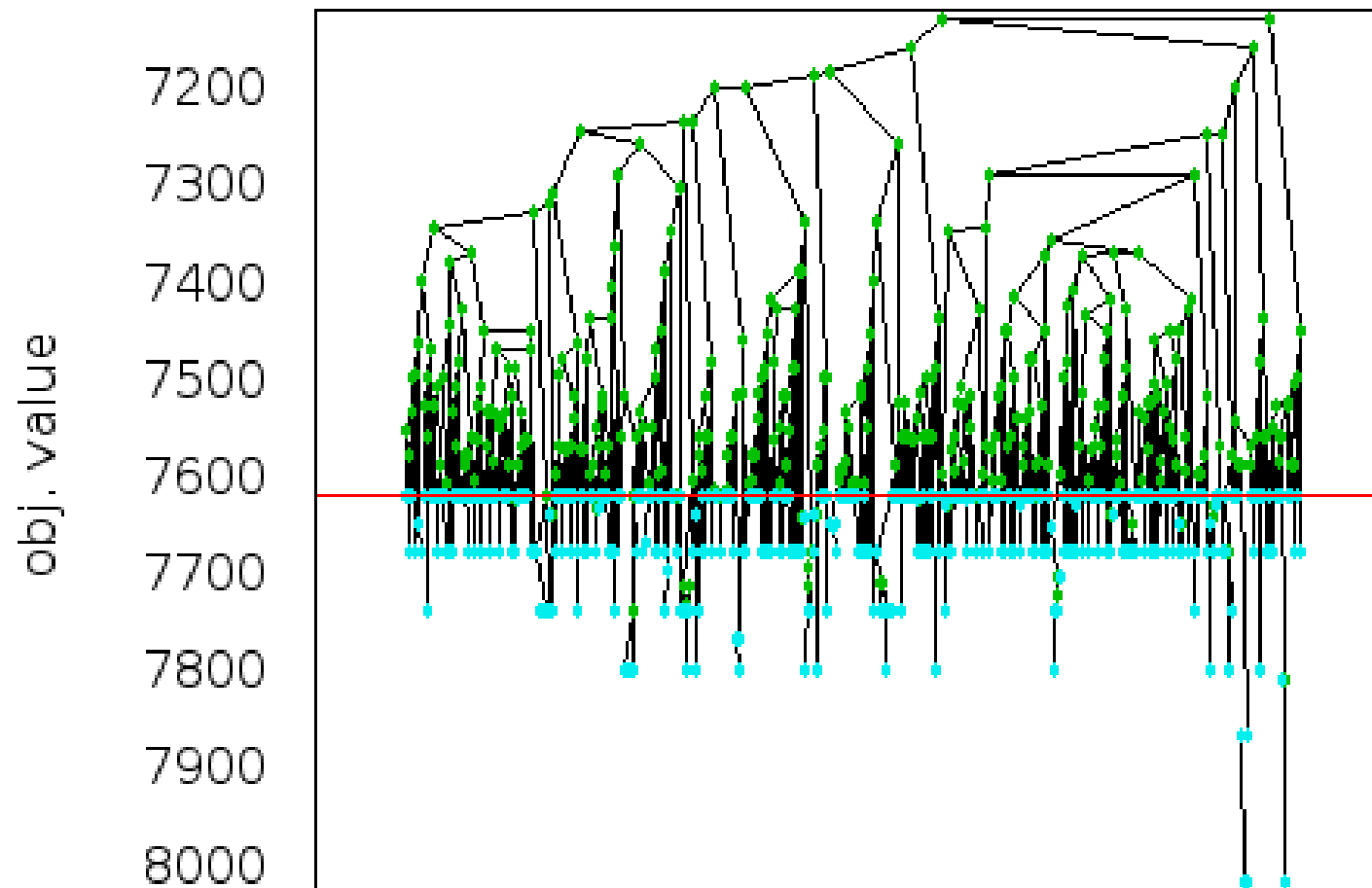


Figure 6: Final tree

## 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 that leads to a good algorithmic implementation.

## Exercise: Install Graphviz, xdot, and GrUMPy

- `pip install coinor.grumpy`
- Graphviz
  - Linux: Install with package manager
  - OS X: `brew install graphviz`
  - Windows: <http://graphviz.org/Download.php>
- xdot: `pip install xdot`
  - Linux: Install with package manager
  - OS X: `brew install pygtk`
  - Windows: <http://pygtk.org/downloads.html>
- `python -m coinor.grumpy.BB`



## Exercise2: Branch and Bound

```
from coinor.grumpy import BBTree, PSEUDOCOST_BRANCHING, MOST_FRACTIONAL
from coinor.grumpy import DEPTH_FIRST, BEST_FIRST, BEST_ESTIMATE

T = BBTree()
T.set_display_mode('xdot')
CONSTRAINTS, VARIABLES, OBJ, MAT, RHS = T.GenerateRandomMIP(rand_seed = 19)
T.BranchAndBound(CONSTRAINTS, VARIABLES, OBJ, MAT, RHS,
                 branch_strategy = PSEUDOCOST_BRANCHING,
                 search_strategy = BEST_FIRST,
                 display_interval = 10000)
```