Financial Optimization
ISE 347/447

Lecture 14

Dr. Ted Ralphs

Reading for This Lecture

- C&T Chapter 11

Solving Integer Programs

- Unlike other kinds of optimization problems, there are no effective direct methods for solving integer programs.
- Most methods involve some sort of *implicit enumeration*.
- The most important component of most solution methods for integer programs is the ability to generate bounds on the value of an optimal solution.
- This can be done using either *duality* or *relaxation*.

Duality in Integer Programming

- Let's consider an integer linear program

$$\begin{aligned} min \quad & c^\top x \\ s.t. \quad & Ax = b \\ & x \geq 0 \\ & x \in \mathbb{Z}^n \end{aligned}$$

- As with other mathematical optimization problems, there is a duality theory for discrete optimization.
- We can "dualize" some of the constraints by allowing them to be violated and then penalizing their violation in the objective function.
- We relax some of the constraints by defining, for given Lagrange multipliers p, the Lagrangian relaxation

$$Z(p) = \min_{x \in X} \{c^\top x + p^\top (A'x - b)\}$$

where $X = \{x \in \mathbb{Z}^n | A''x = b, x \geq 0\}$ and $A^\top = [(A')^\top, (A'')^\top]$.

Duality for Integer Optimization Problems

- Since $Z(p)$ is a lower bound on the optimal solution to the original ILP, we consider the *Lagrangian dual* $\max Z(p)$.
- As long as we can optimize over the set X, we can solve the Lagrangian dual efficiently.
- As before, the optimal solution to the Lagrangian dual yields a lower bound on the optimal value of the original ILP (weak duality).
- However, for integer programming, strong duality does not hold in general.
- The difference between the optimal solution to the ILP and the optimal solution to a given dual is called the *duality gap*.
- The size of the duality gap is an indication of the difficulty of a given integer program.

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

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

- If $\bigcup_{1 \le i \le k} X_i \supset \mathcal{S}$, then the disjunction $\{X_i\}_{i=1}^k$ is said to be *valid* for an MILP with feasible set $\mathcal{S}$.
- 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 $\mathrm{conv}(\bigcup_{1 \le i \le k} X_i)$.
- 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.

Solving Integer Optimization Problems

- *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 a MILP and we wish to find $\min_{x \in F} c^\top x$.
- Consider a partition of F into subsets $F_1, \ldots, F_k$. Then

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

- In other words, we can optimize over each subset independently.
- Idea: If we can't solve the original problem directly, we might be able to divide it into smaller *subproblems* by imposing a disjunction.
- Dividing the original problem into subproblems is called *branching*.
- Taken to the extreme, this scheme is equivalent to complete enumeration.

Branch and Bound

- For the rest of the lecture, assume all variables have finite upper and lower bounds.
- Any feasible solution to the problem provides an upper bound $u(F)$ on the optimal solution value.
- We can use heuristic methods to obtain an upper bound.
- Idea: After branching, try to obtain a *lower bound* $b(F_i)$ on the optimal solution value for each of the subproblems.
- If $b(F_i) \geq u(F)$, then we don't need to consider subproblem i.
- One easy way to obtain a lower bound is by solving the *LP relaxation* obtained by dropping the integrality constraints.

The Importance of Bounding

- 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 $l(\mathcal{S})$ on the optimal solution value.
- We can use heuristic methods to obtain a lower bound.
- Idea: After branching, try to obtain an *upper bound* $b(\mathcal{S}_i)$ on the optimal solution value for each of the subproblems.
- If $b(\mathcal{S}_i) \leq l(\mathcal{S})$, 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

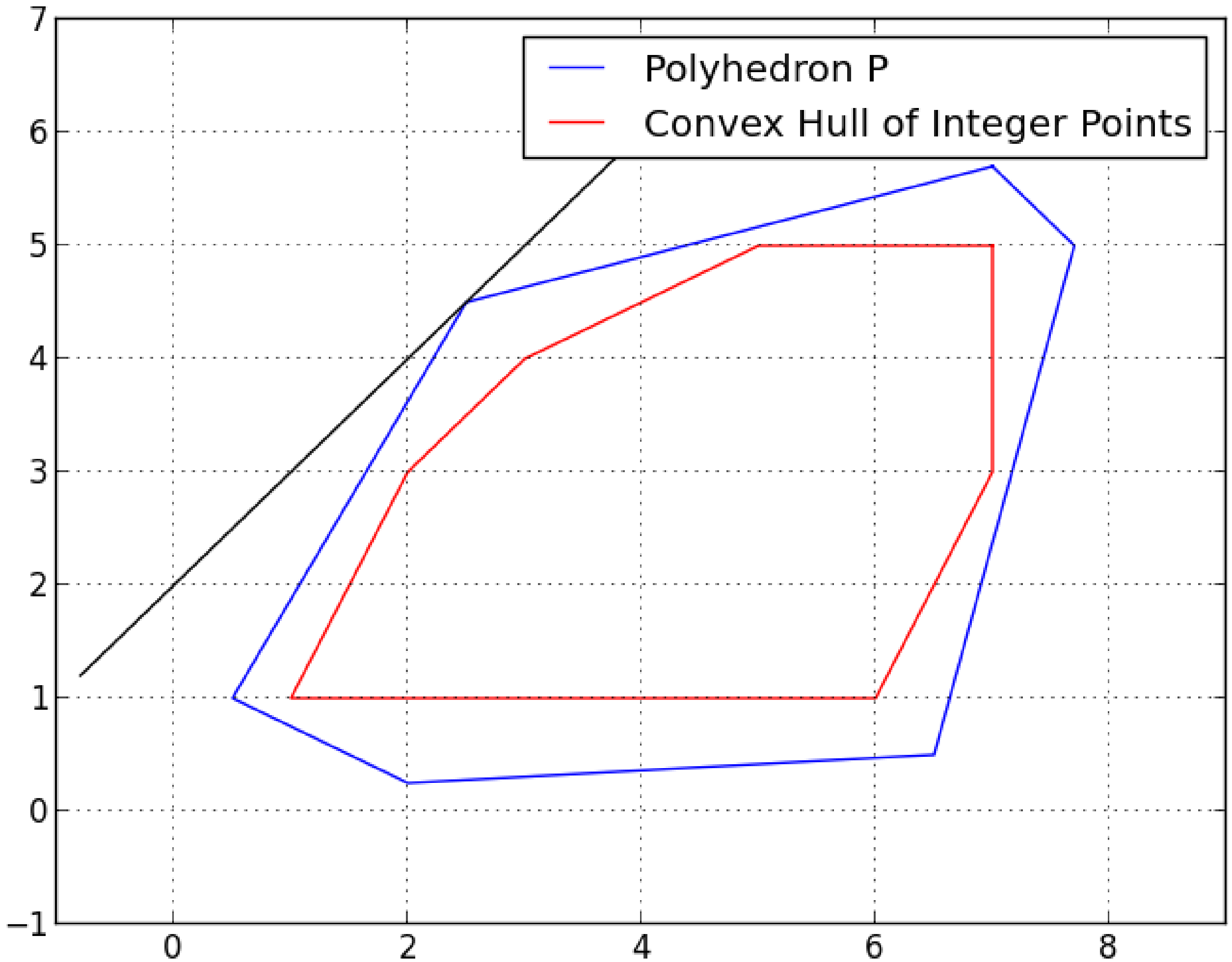


Figure 1: The original feasible region

The Geometry of Branching (cont'd)

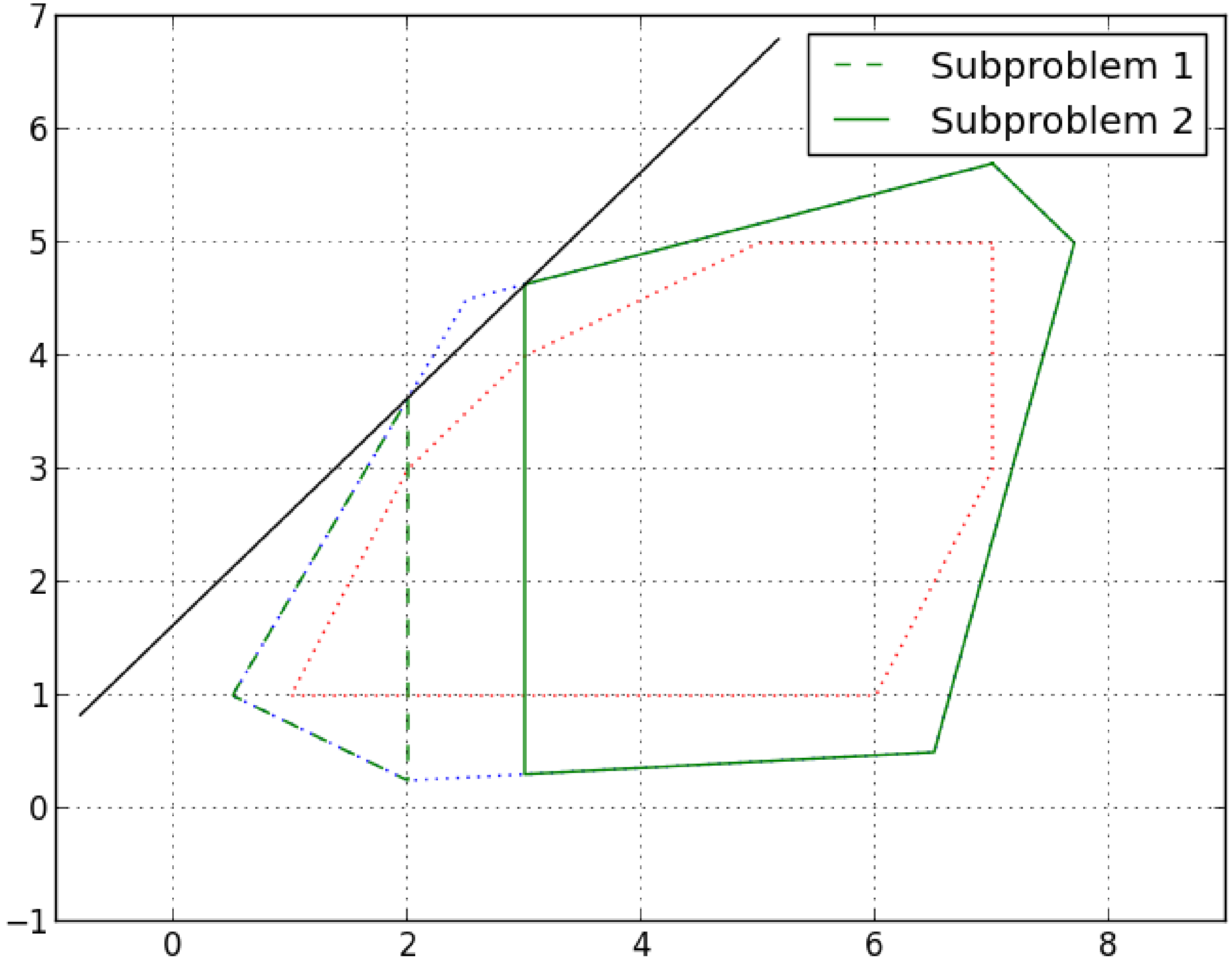


Figure 2: Branching on disjunction $x \leq 2$ OR $x \geq 3$

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)

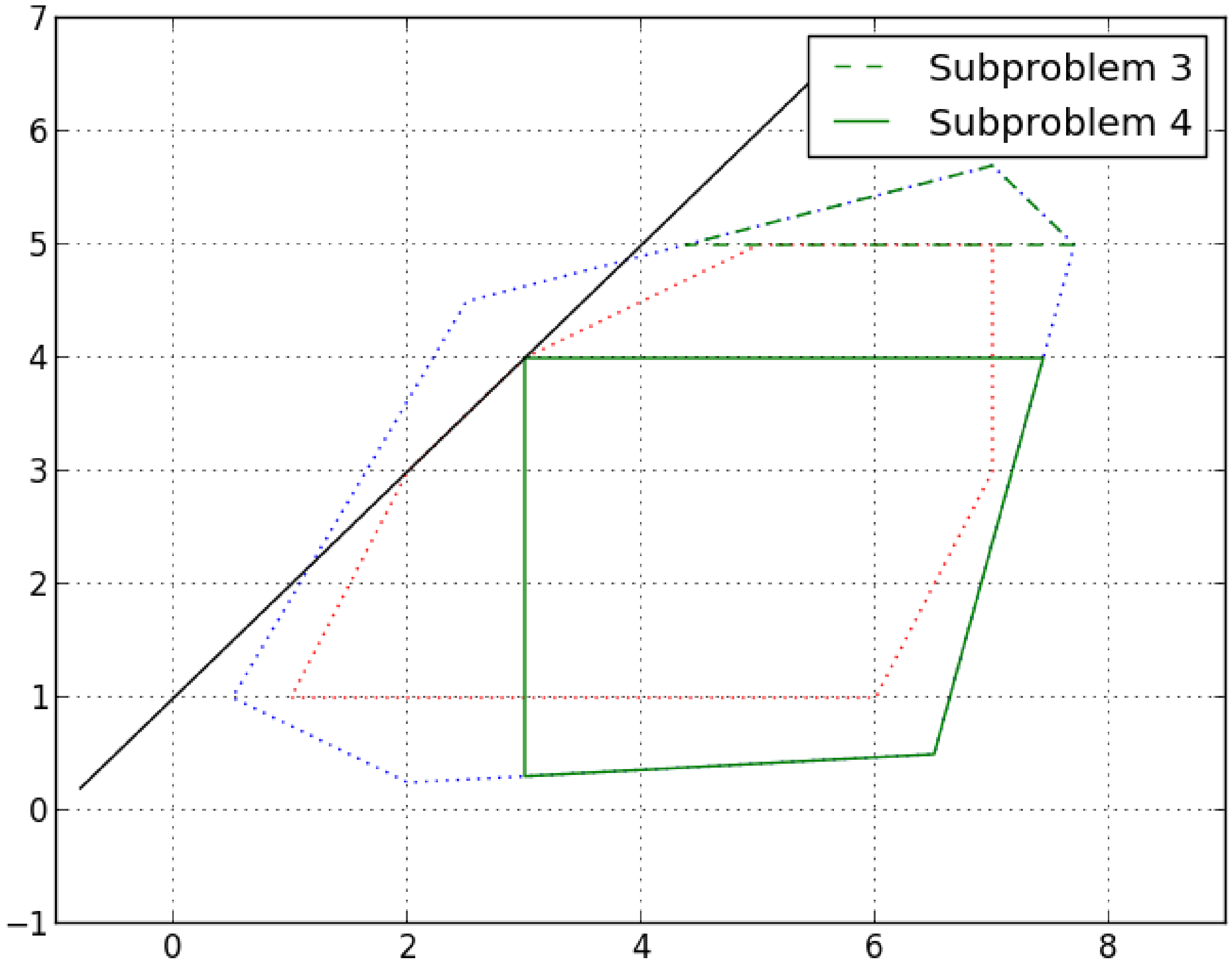
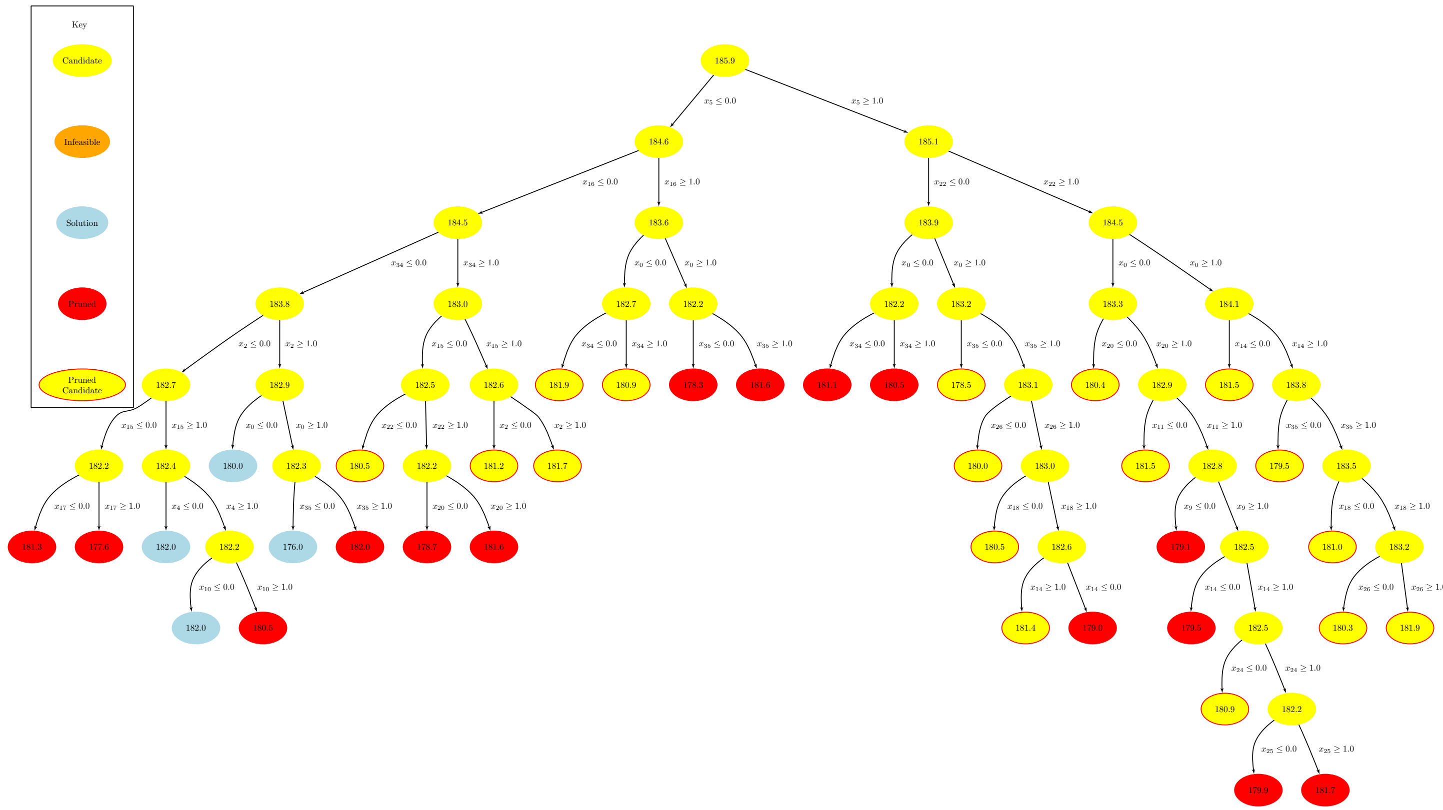


Figure 3: Branching on disjunction $y \leq 4$ OR $y \geq 5$ in Subproblem 2

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 in nonempty, go to Step 2. Otherwise, the algorithm is completed.

Branch and Bound Tree



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

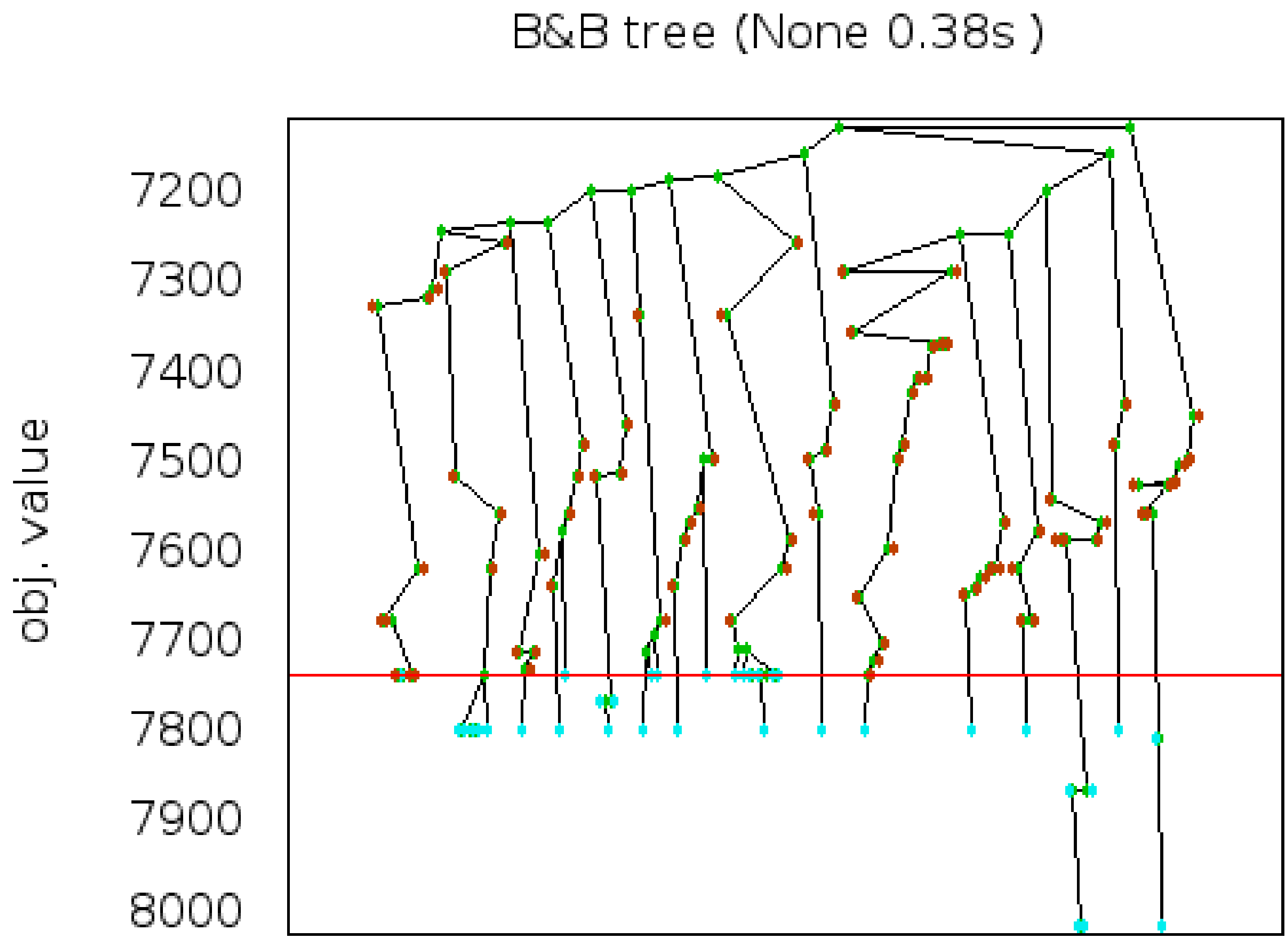


Figure 4: Tree after 400 nodes

A Thousand Words

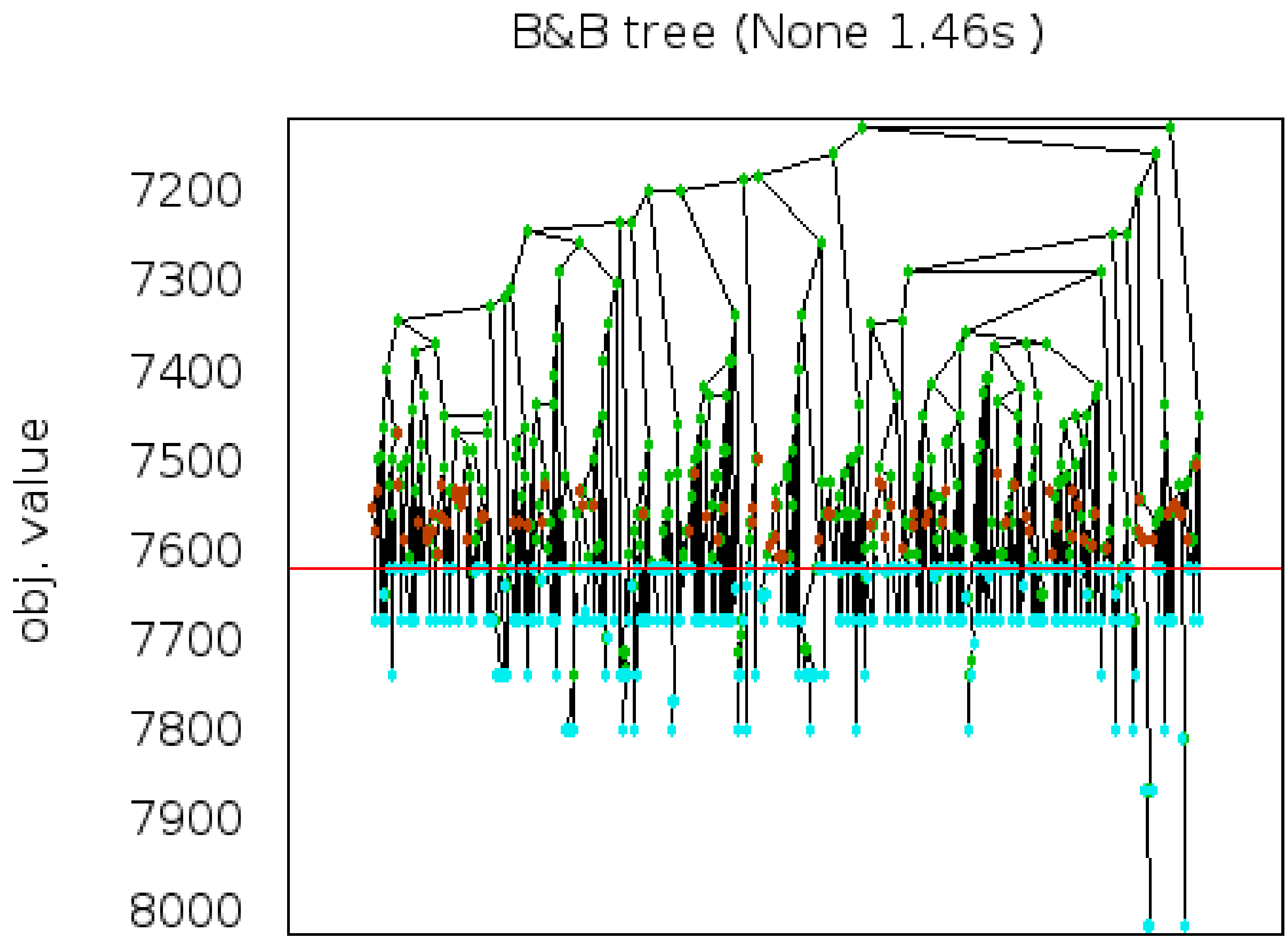


Figure 5: Tree after 1200 nodes

A Thousand Words

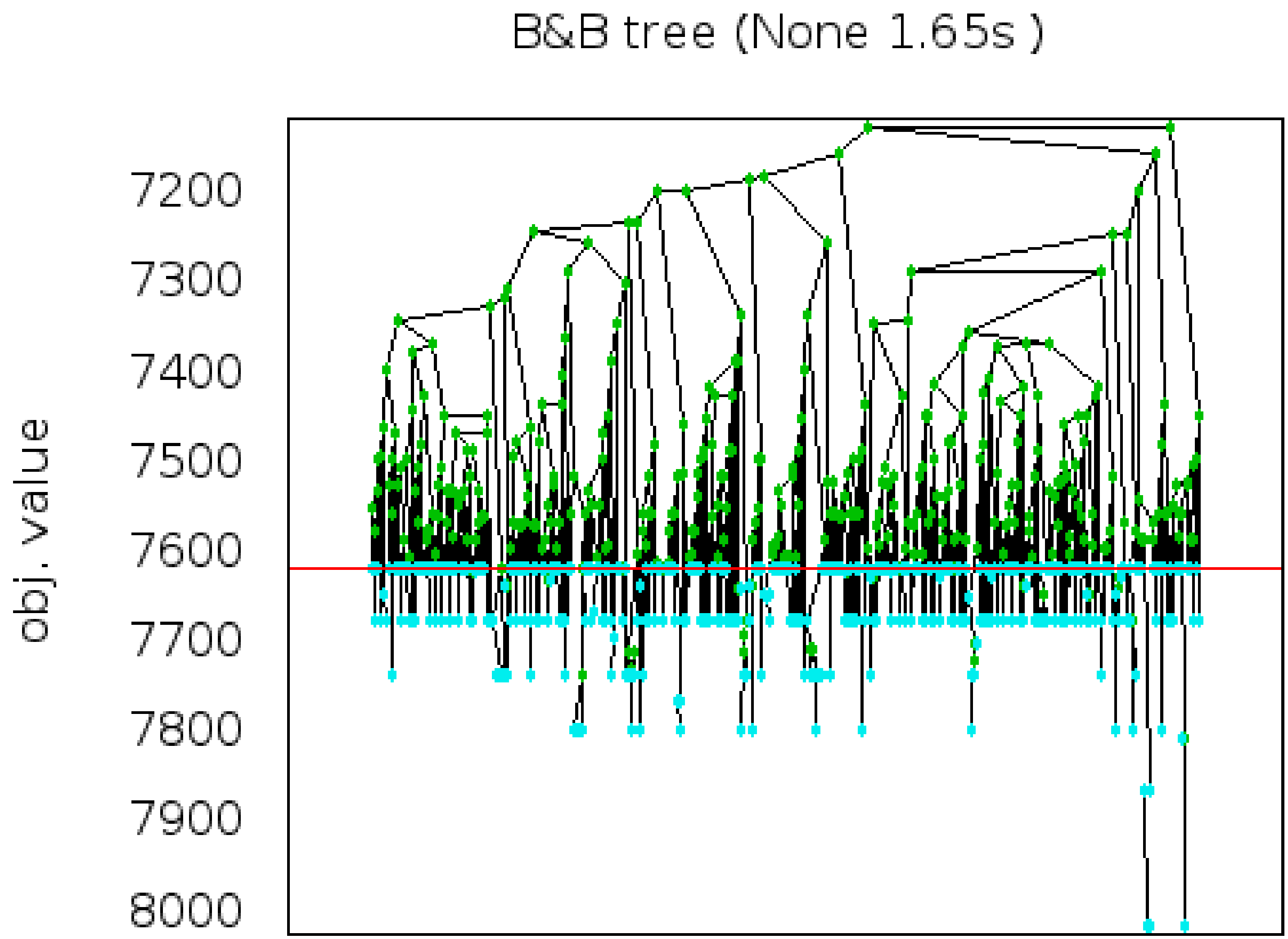


Figure 6: Final tree

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.

Specialized Branching Rules

- As we discussed in the last lecture, certain disjunctive constraint can be enforced by branching rules.
- For example, consider the constraint imposing minimum transaction levels discussed in the last lecture.
- Rather than including binary variables, we can branch on the disjunction by creating two subproblems.
- Suppose that in the current portfolio $\hat{x}$, we have $0 < \hat{x}_i < l_i$.
  - We create one subproblem by imposing the constraint $x_i = 0$.
  - We create another subproblem by imposing the constraint $x_i \geq l_i$.
- By branching exhaustively in this way, we avoid the inclusion of the extra binary variables.

Back to Formulation

- The most vital aspect of branch and bound is obtaining "good" lower bounds.
- In this respect, not all formulations are created equal.
- Choosing the right one is critical.
- A typical MILP can have many alternative formulations.
- Each formulation corresponds to a different polyhedron enclosing the integer points that are feasible for the problem.
- The more closely the polyhedron approximates the convex hull of the integer solutions, the better the bound will be.

Example: The Lockbox Problem

- Consider a mortgage or credit card company that receives checks from all over the United States.
- In general, they would like the checks to get them as quickly as possible.
- It is therefore common practice to open a number of *lockboxes* around the U.S. to receive checks from different regions.
- In this way, the checks are received and cleared earlier resulting in increased revenues.

Example: The Lockbox Problem

- We are given n potential locbox locations and m regions.
- There is a fixed cost c_j of operating a lockbox in location j.
- There is a cost d_{ij} resulting from lost interest of having customers in region i send checks to location j.
- We have two sets of binary variables.
  - y_j is 1 if a lockbox is opened in location j, 0 otherwise.
  - x_{ij} is 1 if region i is served by facility j, 0 otherwise.
- Here is one formulation:

$$
\begin{aligned}
min \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
s.t. \ \sum_{j=1}^{n} x_{ij} = 1 & \qquad \forall i \\
\sum_{i=1}^{m} x_{ij} \leq m y_j & \qquad \forall j \\
x_{ij}, y_j \in \{0, 1\} & \qquad \forall i, j
\end{aligned}
$$

Example: The Lockbox Problem

- Here is another formulation for the same problem:

$$
\begin{aligned}
min \sum_{j=1}^{n} c_j y_j + \sum_{i=1}^{m} \sum_{j=1}^{n} d_{ij} x_{ij} & \\
s.t. \sum_{j=1}^{n} x_{ij} = 1 & \quad \forall i \\
x_{ij} \leq y_j & \quad \forall i, j \\
x_{ij}, y_j \in \{0, 1\} & \quad \forall i, j
\end{aligned}
$$

- Why might we prefer the second formulation to the first one?

Example: The Lockbox Problem

- Notice that the set of integer solutions contained in each of the polyhedra is the same (why?).
- However, the second polyhedra strictly includes the first one.
- Therefore, the second polyhedra will yield better lower bounds and be better for branch and bound.
- Notice that the second formulation includes more constraints, but will likely solve more quickly.

Formulation Strength and Ideal Formulations

- Consider two formulations A and B for the same ILP.
- Denote the corresponding feasible regions for their LP relaxations as P_A and P_B.
- Formulation A is said to be *at least as strong as* formulation B if $P_A \subseteq P_B$.
- If the inclusion is strict, then A is *stronger than* B.
- If F is the set of all feasible integer solutions for the ILP, then we must have $conv(F) \subseteq P_A$ (why?).
- A is *ideal* if $conv(F) = P_A$

Example: Lot-sizing Problem

- Example: A Lot-sizing Problem
  - We want to minimize the costs of production, storage, and set-up.
  - Data for period $t = 1, \ldots, T$:
    - d_t: total demand,
    - c_t: production set-up cost,
    - p_t: unit production cost,
    - h_t: unit storage cost.
  - Variables for period $t = 1, \ldots, T$:
    -
    -
    -

Lot-sizing: The "natural" formulation

- Here is the formulation based on the "natural" set of variables:

$$\begin{aligned} \min \sum_{t=1}^{T} & (p_t y_t + h_t s_t + c_t x_t) \\ \text{s.t. } & y_1 = d_1 + s_1, \\ & s_{t-1} + y_t = d_t + s_t, \qquad \text{for } t = 2, \ldots, T, \\ & y_t \leq \omega x_t, \qquad \text{for } t = 1, \ldots, T, \\ & s_T = 0, \\ & s, y \in \mathbb{R}_+^T, \\ & x \in \{0,1\}^T. \end{aligned}$$

- Here, $\omega = \sum_{t=1}^{T} d_t$, an upper bound on y_t.

Lot-sizing: The "extended" formulation

- Suppose we split the production lot in period t into smaller pieces.
- Define the variables q_{it} to be the production in period i designated to satisfy demand in period $t \geq i$.
- Now, $y_i = \sum_{t=i}^{T} q_{it}$.
- With the new set of variables, we can impose the tighter constraint

$$q_{it} \leq d_t x_i \text{ for } i = 1, \dots, T \text{ and } t = 1, \dots, T.$$

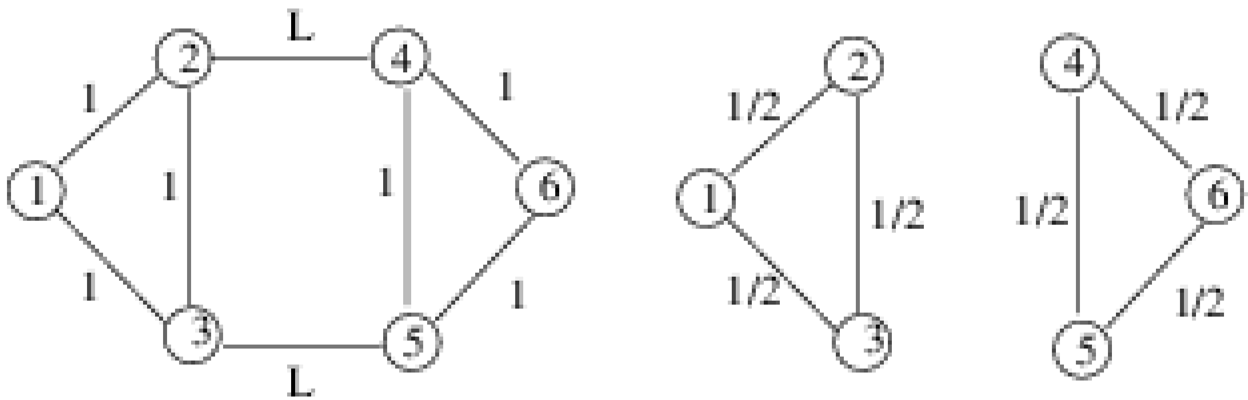
- The additional variables make the formulation ideal.
- If we *project* into the original space, we will get the convex hull of solutions to the first formulation.
- Again, this in contrary to conventional wisdom for formulating linear programs.

Strengthening Formulations

- Often, a given formulation can be strengthened with additional inequalities satisfied by all feasible integer solutions.
- Example: The Perfect Matching Problem
  - We are given a set of n people that need to paired in teams of two.
  - Let c_{ij} represent the "cost" of the team formed by person i and person j.
  - We wish to minimize cost over all teams.
  - We have $x_{ij} = 1$ if i and j are matched, $x_{ij} = 0$ otherwise.

$$
\begin{aligned}
min \sum_{i,j} c_{ij} x_{ij} \\
s.t. \ \sum_{j} x_{ij} = 1, \ \forall i \in N \\
x_{ij} \in \{0,1\}, \ \forall i,j.
\end{aligned}
$$

Valid Inequalities for Matching



- Consider the graph on the left above.
- The optimal perfect matching has value $L + 2$.
- The optimal solution to the LP relaxation has value 3.
- This formulation can be extremely weak.
- Add the valid inequality $x_{24} + x_{35} \geq 1$.
- Every perfect matching satisfies this inequality.

The Odd Set Inequalities

- We can generalize the inequality from the last slide.
- Consider the cut S corresponding to any odd set of nodes.
- The *cutset* corresponding to S is

$$\delta(S) = \{\{i,j\} \in E | i \in s, j \notin S\}.$$

- An *odd cutset* is any $\delta(S)$ for which $|S|$ is odd.
- Note that every perfect matching contains at least one edge from every odd cutset.
- Hence, each odd cutset of induces a possible valid inequality.

$$\sum_{e \in \delta(S)} x_e \geq 1, S \subset N, |S| odd.$$

Using the New Formulation

- If we add all of the odd set inequalities, the new formulation is ideal.
- However, the number of inequalities is exponential in size.
- Only a small number of these inequalities will be active at the optimal solution.
- We can generate these inequalities on the fly.
- This can be done efficiently.

Branch and Cut Algorithms

- If we combine constraint generation with branch and bound, we get *branch and cut*.
- The relaxation at each node is strengthened using valid inequalities.
- This increases the lower bound and improves efficiency.
- Branch and cut is the current state of the art for solving ILPs.

Gomory Inequalities

- The *Gomory procedure* is a generic procedure for generating valid inequalities for mixed integer linear programs.
- It assumes no special problem structure.
- Consider a pure integer program with feasible region $\mathcal{P}$ represented in standard form.
- For a given $u \in \mathbb{R}^m$, we have that $uAx = ub$ for all $x \in \mathcal{P} \cap \mathbb{Z}^n$.
- Because $x \geq 0$ for all $x \in \mathcal{P} \cap \mathbb{Z}^n$, it follows that

$$\lfloor uA \rfloor x \leq ub \, \forall x \in \mathcal{P} \cap \mathbb{Z}^n.$$

- Since $\lfloor uA \rfloor \in \mathbb{Z}^n$, it finally follows that

$$\lfloor uA \rfloor x \leq \lfloor ub \rfloor \, \forall x \in \mathcal{P} \cap \mathbb{Z}^n.$$

- This last inequality is called a *Gomory inequality*.

Generating Gomory Inequalities

- Gomory inequalities are easy to generate in LP-based branch and bound.
- If the solution to the current LP relaxation is not feasible, then we must have $(B^{-1}b)_i \notin \mathbb{Z}$ for some i between 1 and m.
- Taking u to be the i^{th} row of B^{-1}, we see that

$$x_l + \sum_{j \in NB} \lfloor ua_j \rfloor x_j \leq \lfloor ub \rfloor, \ \forall x \in \mathcal{P} \cap \mathbb{Z}^n,$$

  where

  - l is the index of the i^{th} basic variable,
  - NB is the set of indices of the nonbasic variables, and
  - a_j is the j^{th} column of A.

- Eliminating x_l from the above inequality using the equation $uAx = ub$ for all $x \in \mathcal{P} \cap \mathbb{Z}^n$, we obtain

$$\sum_{j \in NB} (ua_j - \lfloor ua_j \rfloor) x_j \geq ub - \lfloor ub \rfloor,$$

Example: Gomory Cuts

Consider the polyhedron $\mathcal{P}$ described by the constraints

$$4x_1 + x_2 \leq 28 \quad (1)$$

$$x_1 + 4x_2 \leq 27 \quad (2)$$

$$x_1 - x_2 \leq 1 \quad (3)$$

$$x_1, x_2 \geq 0 \quad (4)$$

Graphically, it can be easily determined that the facet-inducing valid inequalities describing $\text{conv}(\mathcal{S} = \mathcal{P} \cap \mathbb{Z}^2)$ are

$$x_1 + 2x_2 \leq 15 \quad (5)$$

$$x_1 - x_2 \leq 1 \quad (6)$$

$$x_1 \leq 5 \quad (7)$$

$$x_2 \leq 6 \quad (8)$$

$$x_1 \geq 0 \quad (9)$$

$$x_2 \geq 0 \quad (10)$$

Example: Gomory Cuts (cont.)

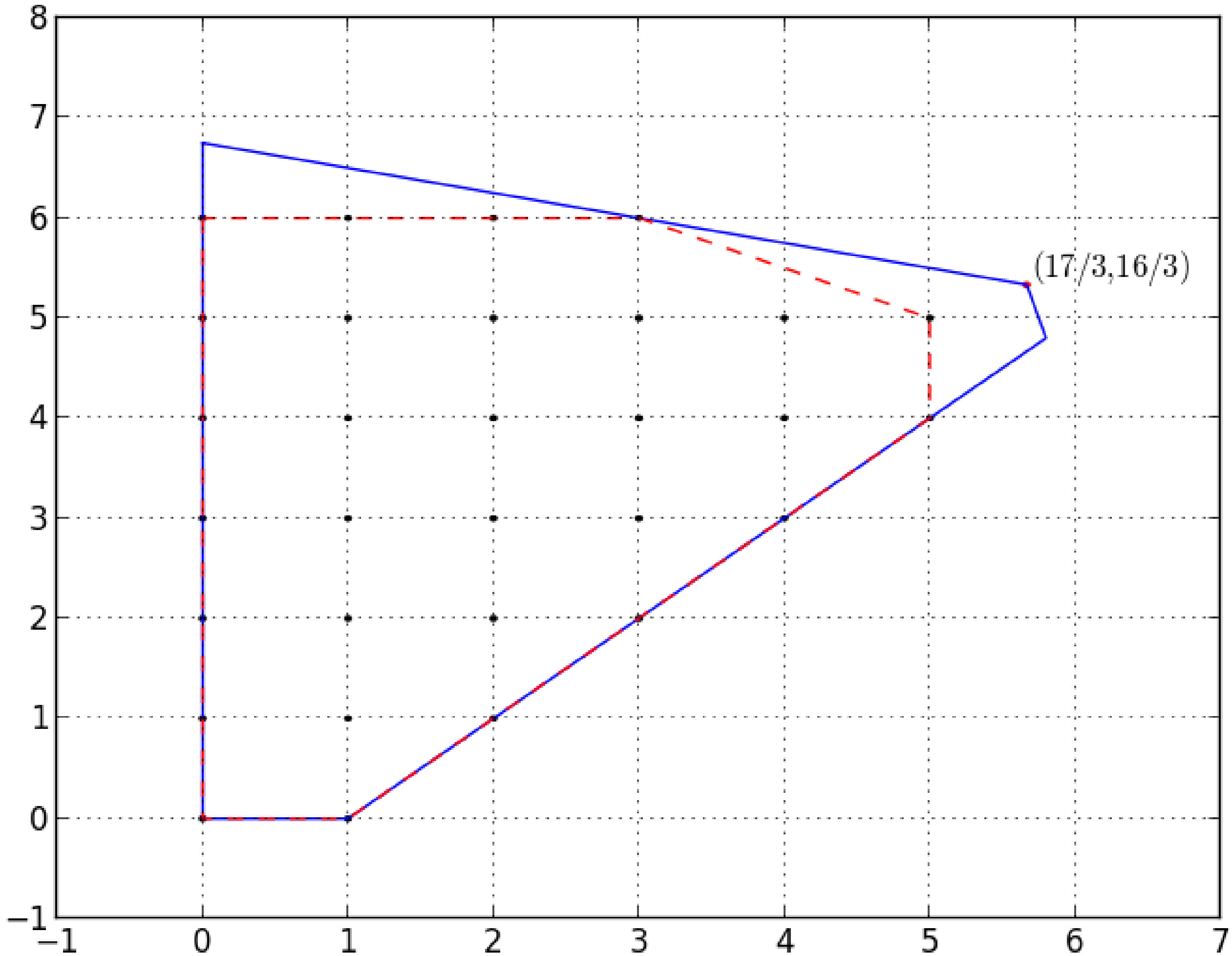


Figure 7: Convex hull of $\mathcal{S}$

Example: Gomory Cuts (cont.)

Consider the optimal tableau of the LP relaxation of the integer program

$$\max\{2x_1 + 5x_2 \mid x \in \mathcal{S}\},$$

shown in Table **??**.

Basic var.	x_1	x_2	s_1	s_2	s_3	RHS
x_2	0	1	-2/30	8/30	0	16/3
s_3	0	0	-1/3	1/3	1	2/3
x_1	1	0	8/30	-2/30	0	17/3

Table 1: Optimal tableau of the LP relaxation

The associated optimal solution to the LP relaxation is also shown in Figure **??**.

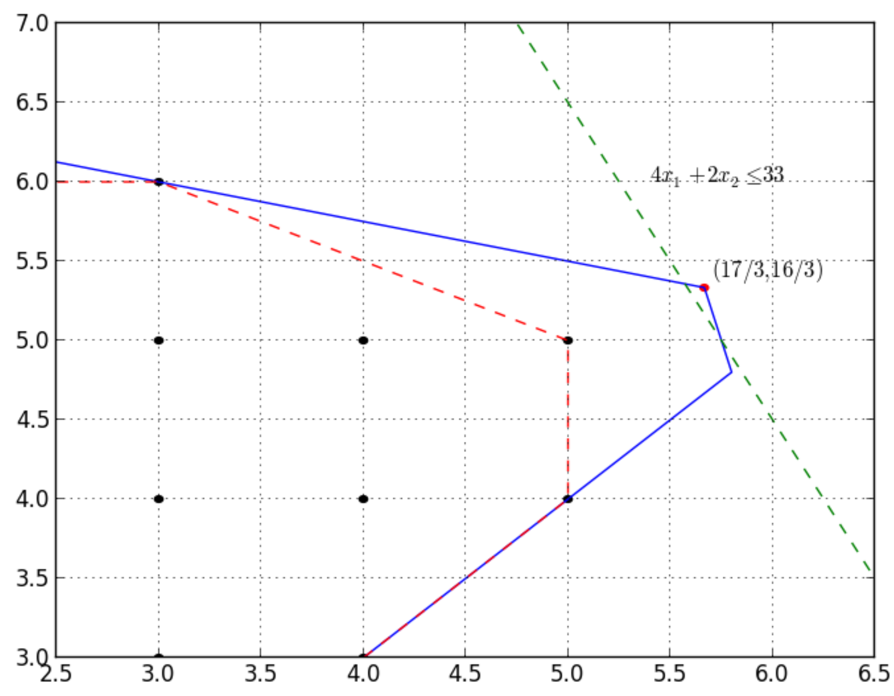
Example: Gomory Cuts (cont.)

The Gomory cut from the first row is

$$\frac{28}{30}s_1 + \frac{8}{30}s_2 \geq \frac{1}{3},$$

In terms of x_1 and x_2, we have

$$4x_1 + 2x_2 \leq 33, \qquad \text{(G-C1)}$$



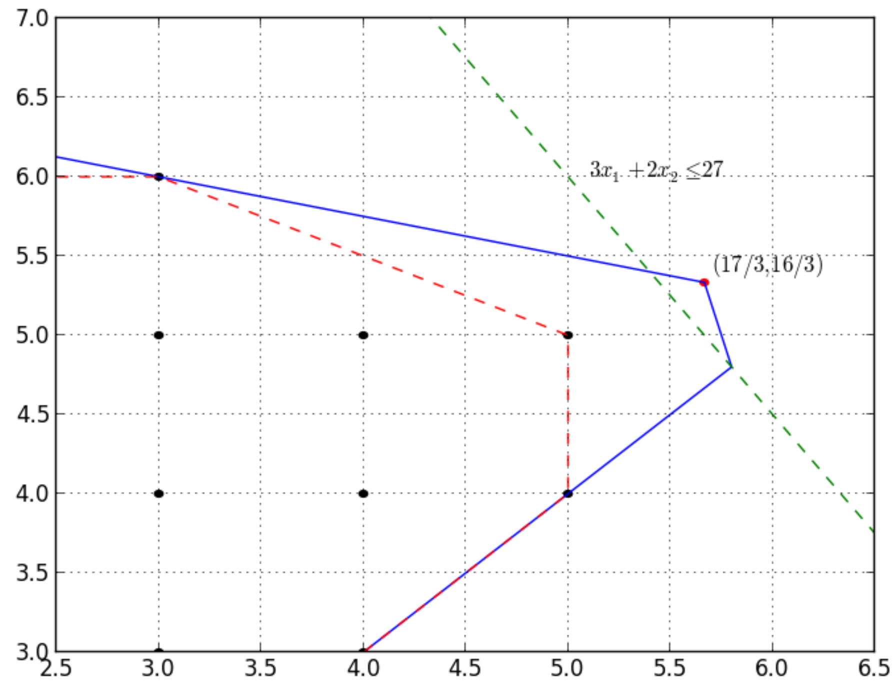
Example: Gomory Cuts (cont.)

The Gomory cut from the second row is

$$\frac{2}{3}s_1 + \frac{1}{3}s_2 \geq \frac{2}{3},$$

In terms of x_1 and x_2, we have

$$3x_1 + 2x_2 \leq 27, \qquad \text{(G-C2)}$$



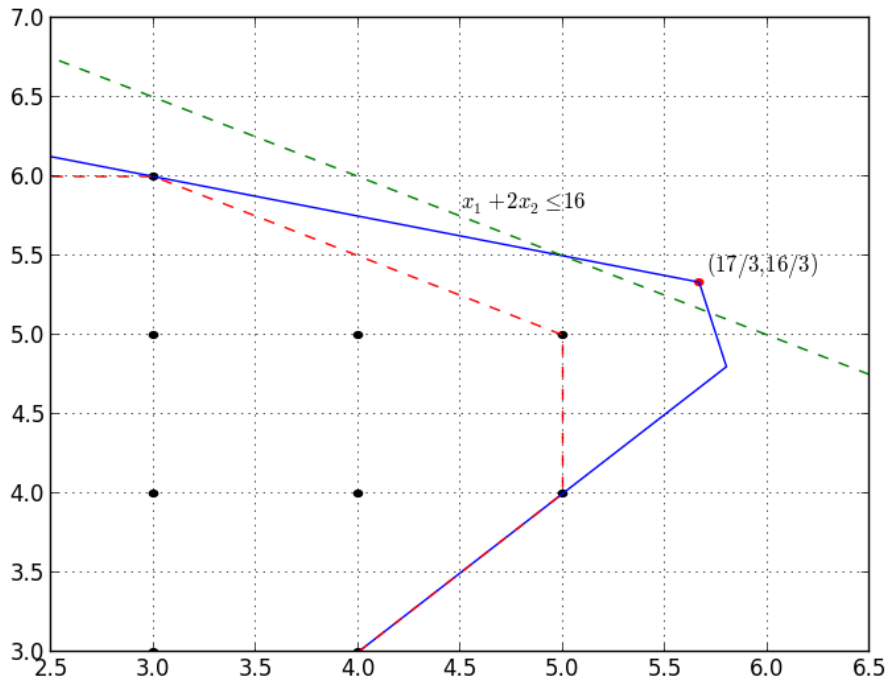
Example: Gomory Cuts (cont.)

The Gomory cut from the third row is

$$\frac{8}{30}s_1 + \frac{28}{30}s_2 \geq \frac{2}{3},$$

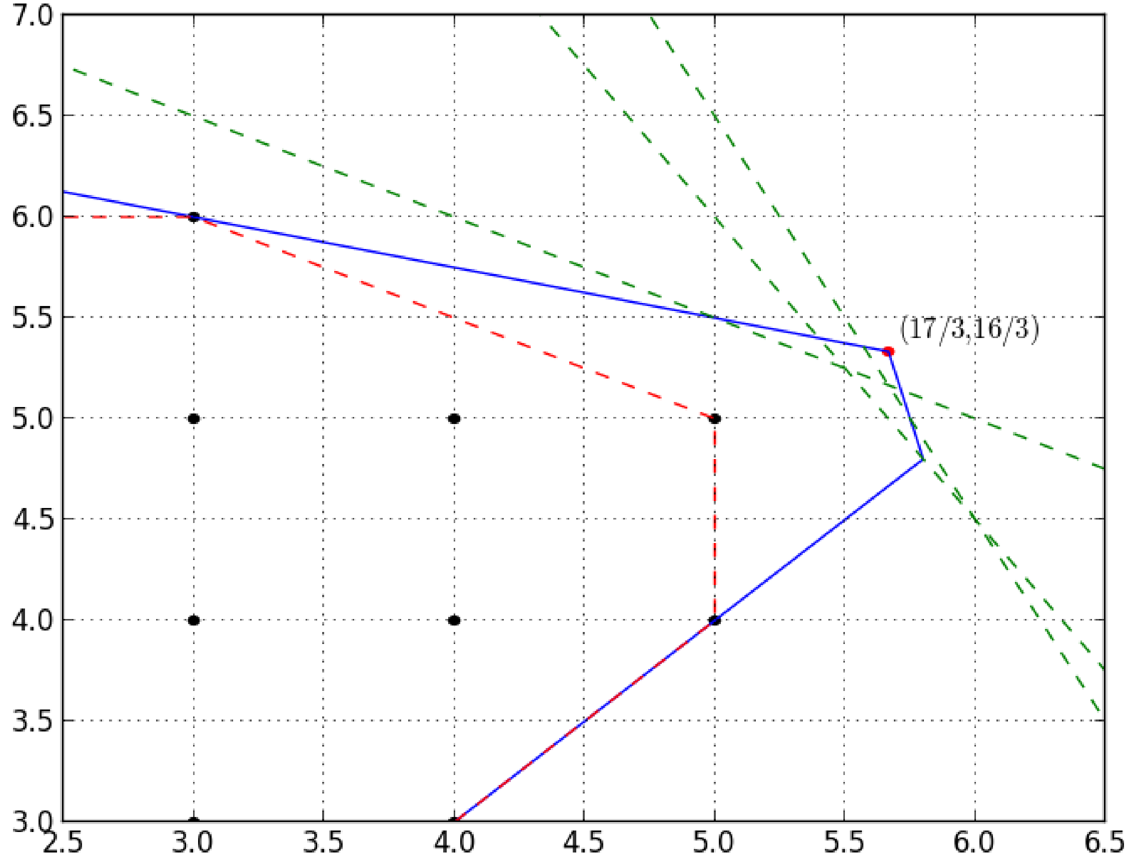
In terms of x_1 and x_2, we have

$$x_1 + 2x_2 \leq 16, \qquad \text{(G-C3)}$$



Example: Gomory Cuts (cont.)

This picture shows the effect of adding all Gomory cuts in the first round.



Valid Inequalities from Disjunctions

Another viewpoint for constructing valid inequalities based on disjunctions comes from the following result:

Proposition 1. *If* $\sum_{j=1}^{n} \pi_j^1 \le \pi_0^1$ *is valid for* $S_1 \subseteq \mathbb{R}_+^n$ *and* $\sum_{j=1}^{n} \pi_j^2 \le \pi_0^2$ *is valid for* $S_2 \subseteq \mathbb{R}_+^n$*, then*

$$\sum_{j=1}^{n} \min(\pi_j^1, \pi_j^2) x \le \max(\pi_0^1, \pi_0^1)$$

for $x \in S_1 \cup S_2$*.*

In fact, all valid inequalities for the union of two polyhedra can be obtained in this way.

Proposition 2. *If* $\mathcal{P}^i = \{x \in \mathbb{R}_+^n \mid A^i x \le b^i\}$ *for* $i = 1, 2$ *are nonempty polyhedra, then* (π, π_0) *is a valid inequality for* $conv(\mathcal{P}^1 \cup \mathcal{P}^2)$ *if and only if there exist* $u^1, u^2 \in \mathbb{R}^m$ *such* $\pi \le u^i A^i$ *and* $\pi_0 \ge u^i b^i$ *for* $i = 1, 2$*.*

Strengthening Gomory Cuts Using Disjunction

- Consider the set of solutions to an IP with one equation.
- We can write the feasible set S as

$$S = \left\{ x \in \mathbb{Z}^n_+ \mid \sum_{j:f_j \le f_0} f_j x_j + \sum_{j:f_j > f_0} (f_j - 1) x_j = f_0 + k \text{ for some integer k} \right\}$$

- Since $k \le -1$ or $k \ge 0$, we have the disjunction

$$\sum_{j:f_j \le f_0} \frac{f_j}{f_0} x_j - \sum_{j:f_j > f_0} \frac{(1 - f_j)}{f_0} x_j \ge 1$$

OR

$$-\sum_{j:f_j \le f_0} \frac{f_j}{(1 - f_0)} x_j + \sum_{j:f_j > f_0} \frac{(1 - f_j)}{(1 - f_0)} x_j \ge 1$$

The Gomory Mixed Integer Cut

- Applying Proposition **??**, we get

$$\sum_{j:f_j \le f_0} \frac{f_j}{f_0} x_j + \sum_{j:f_j > f_0} \frac{(1-f_j)}{(1-f_0)} x_j \ge 1$$

- This is called a *Gomory mixed integer* (GMI) cut.
- GMI cuts dominate the associated Gomory cut in general and can also be obtained easily from the tableau.
- In the case of the mixed integer set

$$S = \left\{ x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p} \mid \sum_{j=1}^{p} a_j x_j + \sum_{j=p+1}^{n} g_j x_j = a_0 \right\},$$

the GMI cut is

$$\sum_{j:f_j \le f_0} \frac{f_j}{f_0} x_j + \sum_{j:f_j > f_0} \frac{(1-f_j)}{(1-f_0)} x_j + \sum_{j:g_j > 0} \frac{g_j}{f_0} x_j - \sum_{j:g_j < 0} \frac{g_j}{(1-f_0)} x_j \ge 1$$