Lecture: Introduction to Integer Programming

https://bicmr.pku.edu.cn/~wenzw/bigdata2022.html

Acknowledgement: this slides is based on Prof. Ted Ralphs's lecture notes

Outline

Formal Setting

- We consider linear optimization problems with additionally constraints
$$X = \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}.$$

- The general form of such a mathematical optimization problem is
$$z_{IP} = \max\{c^\top x \mid Ax \leq b, x \in \mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}\},$$
where for $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$.

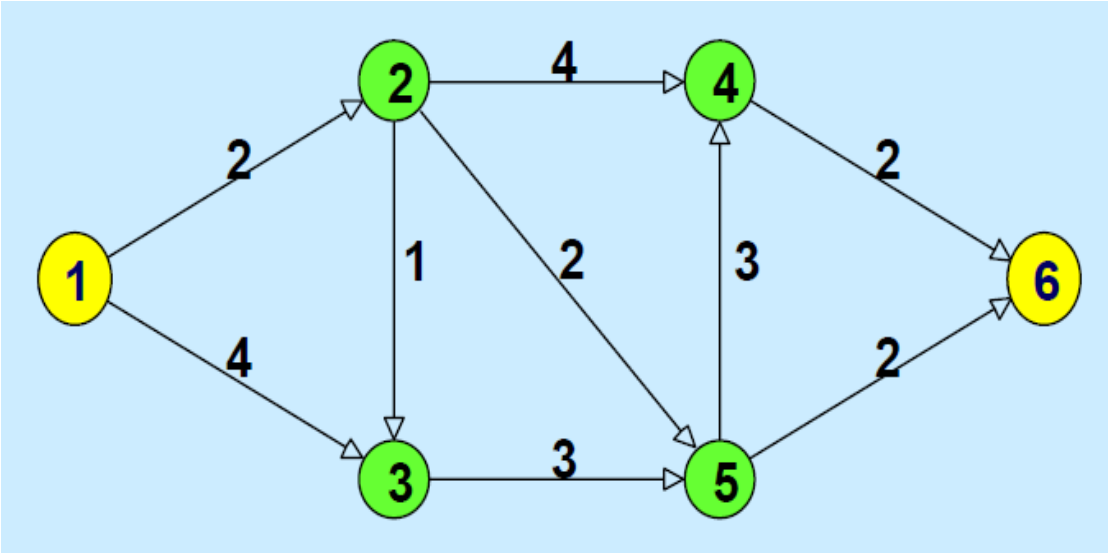
- This type of optimization problem is called a mixed integer linear optimization problem.

- If $p = n$, then we have a pure integer linear optimization problem.

Special Case: Binary Integer Optimization

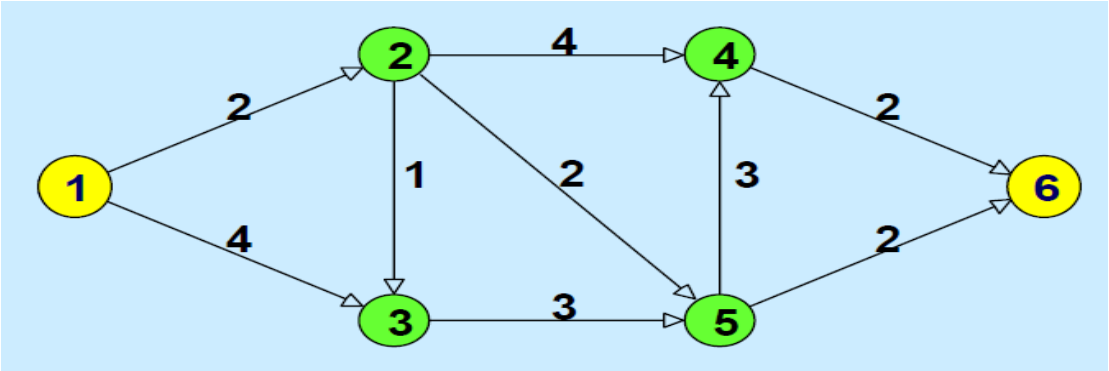
- In many cases, the variables of an IP represent yes/no decisions or logical relationships.
- These variables naturally take on values of 0 or 1.
- Such variables are called binary.
- IPs involving only binary variables are called binary optimization problems.

The shortest path problem



- Consider a network G = (N, A) with cost c_{ij} on each edge $(i,j) \in A$. There is an origin node s and a destination node t.
- Standard notation: n = |N|, m = |A|
- cost of of a path: $c(P) = \sum_{(i,j)\in P} c_{ij}$
- What is the shortest path from s to t?

The shortest path problem



$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij}$$

s.t. $\sum_j x_{sj} = 1$

$$\sum_j x_{ij} - \sum_j x_{ji} = 0, \text{ for each i } \neq s \text{ or } t$$

$$-\sum_i x_{it} = -1$$

$$x_{ij} \in \{0,1\} \text{ for all } (i,j)$$

Combinatorial Optimization

- A combinatorial optimization problem $CP = (N, \mathcal{F})$ consists of
  1. A finite ground set N,
  2. A set $\mathcal{F} \subseteq 2^N$ of feasible solutions, and
  3. A cost function $c \in Z^n$
- The cost of $F \in \mathcal{F}$ is $c(F) = \sum_{j \in F} c_j$.
- The combinatorial optimization problem is then

$$\max\{c(F) \mid F \in \mathcal{F}\}$$

- There is a natural association with a 0-1 math program.
- Many COPs can be written as BIPs or MIPs.

How Hard is Integer Programming?

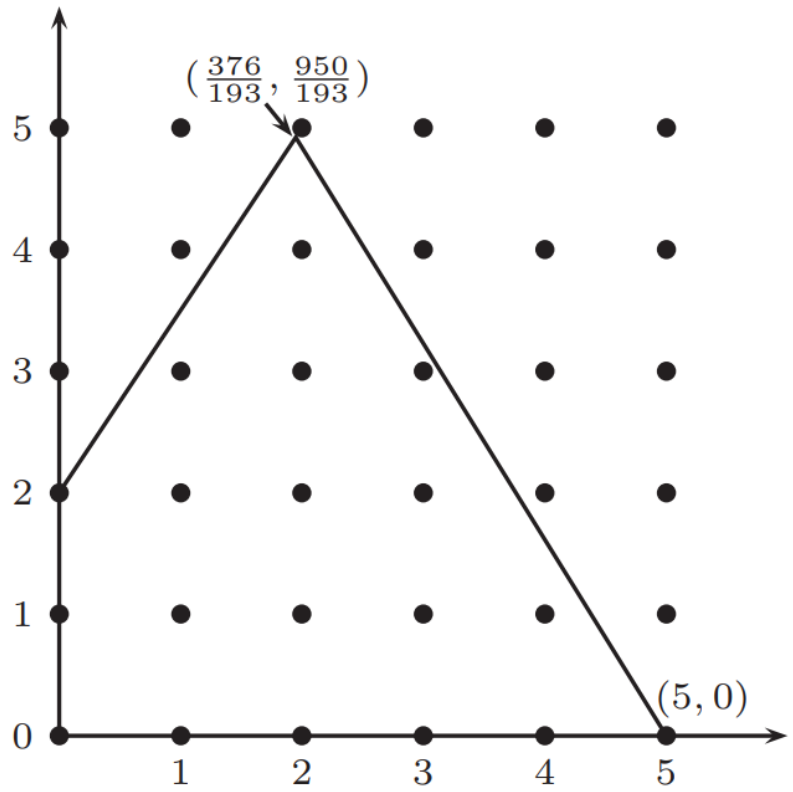
- Solving general integer programs can be much more difficult than solving linear programs.
- There in no known polynomial-time algorithm for solving general MIPs.
- Solving the associated linear programming relaxation results in an upper bound on the optimal solution to the MIP.
- In general, solving the LP relaxation, an LP obtained by dropping the integerality restrictions, does not tell us much.
  - Rounding to a feasible integer solution may be difficult.
  - The optimal solution to the LP relaxation can be arbitrarily far away from the optimal solution to the MIP.
  - Rounding may result in a solution far from optimal.
  - We can bound the difference between the optimal solution to the LP and the optimal solution to the MIP (how?).

How Hard is Integer Programming?

Consider the integer program

$$\begin{aligned} \max \quad & 50x_1 + 32x_2, \\ \text{s.t.} \quad & 50x_1 + 31x_2 \leq 250, \\ & 3x_1 - 2x_2 \geq -4, \\ & x_1, x_2 \geq 0 \text{ and integer.} \end{aligned}$$

The linear programming solution $(376\backslash 193, 950\backslash 193)$ is a long way from the optimal integer solution $(5, 0)$.



Integer Programming and Convexity

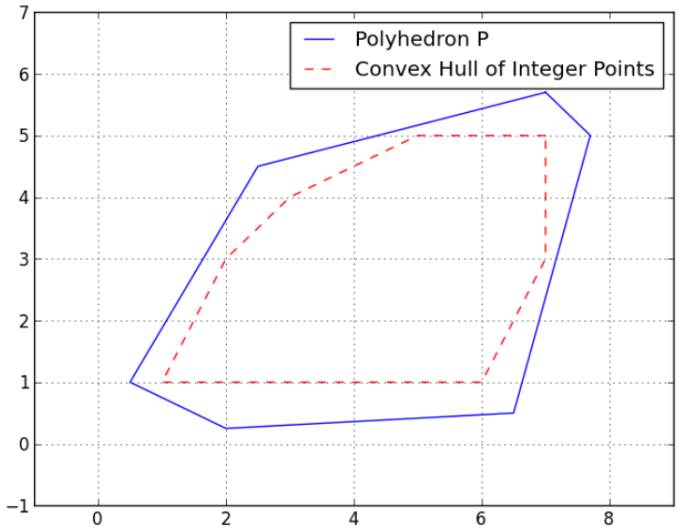
- The feasible region of an integer program is nonconvex.
- The nonconvexity is of a rather special form, though.
- Although the feasible set is nonconvex, there is a convex set over which we can optimize in order to get a solution (why?).
- The challenge is that we do not know how to describe that set.
- Even if we knew the description, it would in general be too large to write down explicitly.
- Integer variables can be used to model other forms of nonconvexity, as we will see later on.

The Geometry of Integer Programming

- Let's consider again an integer linear program

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & Ax \leq b \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

- The feasible region is the integer points inside a polyhedron.



- Why does solving the LP relaxation not necessarily yield a good solution?

Conjunction versus Disjunction

- A more general mathematical view that ties integer programming to logic is to think of integer variables as expressing **disjunction**.
- The constraints of a standard mathematical program are **conjunctive**.
  - All constraints must be satisfied.

$$g_1(x) \leq b_1 \text{ AND } g_2(x) \leq b_1 \text{ AND } \cdots \text{ AND } g_m(x) \leq b_m$$

  - This corresponds to intersection of the regions associated with each constraint.
- Integer variables introduce the possibility to model disjunction.
  - At least one constraint must be satisfied.

$$g_1(x) \leq b_1 \text{ OR } g_2(x) \leq b_1 \text{ OR } \cdots \text{ OR } g_m(x) \leq b_m$$

  - This corresponds to union of the regions associated with each constraint.

Representability Theorem

The connection between integer programming and disjunction is captured most elegantly by the following theorem.

Theorem

A set $\mathcal{F} \subseteq \mathbb{R}^n$ is MIP representable if and only if there exist rational polytopes $\mathcal{P}_1, \cdots, \mathcal{P}_k$ and vectors $r^1, \cdots, r^t \in \mathbb{Z}^n$ such that

$$\mathcal{F} = \bigcup_{i=1}^{n} \mathcal{P}_i + \text{intcone}\{r^1, \cdots, r^t\}.$$

where $\text{intcone}\{r^1, \cdots, r^t\} = \left\{\sum_{i=1}^{t} \lambda_i r_i \mid \lambda \in \mathbb{Z}^t_+\right\}$

Roughly speaking, we are optimizing over a union of polyhedra, which can be obtained simply by introducing a disjunctive logical operator to the language of linear programming.

Outline

Modeling with Integer Variables

- From a practical standpoint, why do we need integer variables?
- We have seen in the last lecture that integer variable essentially allow us to introduce disjunctive logic.
- If the variable is associated with a physical entity that is indivisible, then the value must be integer.
- At its heart, integrality is a kind of disjunctive constraint.
- 0-1 (binary) variables are often used to model more abstract kinds of disjunctions (non-numerical).
  - Modeling yes/no decisions.
  - Enforcing logical conditions.
  - Modeling fixed costs.
  - Modeling piecewise linear functions.

Modeling Binary Choice

- We use binary variables to model yes/no decisions.
- Example: Integer knapsack problem
  - We are given a set of items with associated values and weights.
  - We wish to select a subset of maximum value such that the total weight is less than a constant K.
  - We associate a 0-1 variable with each item indicating whether it is selected or not.

$$\begin{aligned} \max \quad & \sum_{j=1}^{m} c_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{m} w_j x_j \leq K \\ & x \in \{0,1\}^n \end{aligned}$$

Modeling Dependent Decisions

- We can also use binary variables to enforce the condition that a certain action can only be taken if some other action is also taken.
- Suppose x and y are binary variables representing whether or not to take certain actions.
- The constraint $x \leq y$ says "only take action x if action y is also taken"

Example: Facility Location Problem

- We are given n potential facility locations and m customers.
- There is a fixed cost c_j of opening facility j.
- There is a cost d_{ij} associated with serving customer i from facility j.
- We have two sets of binary variables.
  - y_j is 1 if facility j is opened, 0 otherwise.
  - x_{ij} is 1 if customer i is served by facility j, 0 otherwise.
- Here is one formulation:

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

Selecting from a Set

- We can use constraints of the form $\sum_{j \in T} x_j \geq 1$ to represent that at least one item should be chosen from a set T.
- Similarly, we can also model that at most one or exactly one item should be chosen.
- Example: Set covering problem
  - A set covering problem is any problem of the form.

$$\min \quad \{c^\top x \mid Ax \geq 1, x_j \in \{0,1\}\}$$

    where A is a 0-1 matrix.
  - Each row of A represents an item from a set S.
  - Each column A_j represents a subset S_j of the items.
  - Each variable x_j represents selecting subset S_j.
  - In other words, each item must appear in at least one selected subset.

Modeling Disjunctive Constraints

- We are given two constraints $a^\top x \geq b$ and $c^\top x \geq d$ with nonnegative coefficients.
- Instead of insisting both constraints be satisfied, we want at least one of the two constraints to be satisfied.
- To model this, we define a binary variable y and impose

$$\begin{aligned} a^\top x &\geq yb, \\ c^\top x &\geq (1-y)d, \\ y &\in \{0,1\}. \end{aligned}$$

- More generally, we can impose that at least k out of m constraints be satisfied with

$$\begin{aligned} a_i^\top x &\geq y_i b_i, \\ \sum_{i=1}^{m} y_i &\geq k, \\ y_i &\in \{0,1\}. \end{aligned}$$

Modeling Disjunctive Constraints (cont'd)

- Consider the disjunctive constraints $a^\top x \geq b$ and $c^\top x \geq d$ where the coefficients are allowed to be negative.
- To model this, we use the Big-M Reformulation. we define a binary variable y and impose

$$\begin{aligned} a^\top x &\geq b - My, \\ c^\top x &\geq d - M(1-y), \\ y &\in \{0,1\}. \end{aligned}$$

where M is a sufficiently large positive number.

Modeling a Restricted Set of Values

- We may want variable x to only take on values in the set $\{a_1, \cdots, a_m\}$.
- We introduce m binary variables y_j, $j = 1, \cdots, m$ and the constraints

$$x = \sum_{j=1}^{m} a_j y_j,$$

$$\sum_{j=1}^{m} y_j = 1,$$

$$y_j \in \{0, 1\}.$$

Piecewise Linear Cost Functions

- We can use binary variables to model arbitrary piecewise linear cost functions.
- The function is specified by ordered pairs $(a_i, f(a_i))$ and we wish to evaluate it at a point x.
- We have a binary variable y_i, which indicates whether $a_i \leq x \leq a_i + 1$.
- To evaluate the function, we will take linear combinations $\sum_{i=1}^{k} \lambda_i f(a_i)$ of the given functions values.
- This only works if the only two nonzero λ_i's are the ones corresponding to the endpoints of the interval in which x lies.

Minimizing Piecewise Linear Cost Functions

- The following formulation minimizes the function.

$$
\begin{aligned}
\min \quad & \sum_{i=1}^{k} \lambda_i f(a_i) \\
\text{s.t.} \quad & \sum_{i=1}^{k} \lambda_i = 1, \quad \sum_{i=1}^{k-1} y_i = 1, \\
& \lambda_1 \leq y_1, \\
& \lambda_i \leq y_{i-1} + y_i, \\
& \lambda_k \leq y_{k-1}, \\
& \lambda_i \geq 0, \quad y_i \in \{0, 1\}.
\end{aligned}
$$

- The key is that if $y_j = 1$, then $\lambda_i = 0, \forall i \neq j, j+1$.

Modeling General Nonconvex Functions

- One way of dealing with general nonconvexity is by dividing the domain of a nonconvex function into regions over which it is convex (or concave).
- We can do this using integer variables to choose the region.
- This is precisely what is done in the case of the piecewise linear cost function above.
- Most methods of general global optimization use some form of this approach.

Fixed-charge Problems

- In many instances, there is a fixed cost and a variable cost associated with a particular decision.
- Example: Fixed-charge Network Flow Problem
  - We are given a directed graph $G = (N, A)$.
  - There is a fixed cost c_{ij} associated with "opening" arc (i, j) (think of this as the cost to "build" the link).
  - There is also a variable cost d_{ij} associated with each unit of flow along arc (i, j).
  - Consider an instance with a single supply node.
    - Minimizing the fixed cost by itself is a minimum spanning tree problem (easy).
    - Minimizing the variable cost by itself is a minimum cost network flow problem (easy).
    - We want to minimize the sum of these two costs (difficult).

Modeling the Fixed-charge Network Flow Problem

- To model the FCNFP, we associate two variables with each arc.
  - x_{ij} (fixed-charge variable) indicates whether arc (i,j) is open.
  - f_{ij} (flow variable) represents the flow on arc (i,j).
  - Note that we have to ensure that $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$\min \quad \sum_{(i,j)\in A} c_{ij}x_{ij} + d_{ij}f_{ij}$$

$$\text{s.t.} \quad \sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} f_{ji} = b_i, \qquad \forall i \in N$$

$$f_{ij} \leq Cx_{ij}, \qquad \forall (i,j) \in A$$

$$f_{ij} \geq 0, \qquad \forall (i,j) \in A$$

$$x_{ij} \in \{0,1\}, \qquad \forall (i,j) \in A$$

Alternative Formulations

- A key concept in the rest of the course will be that every mathematical model has many alternative formulations.
- Many of the key methodologies in integer programming are essentially automatic methods of reformulating a given model.
- The goal of the reformulation is to make the model easier to solve.

Simple Example: Knapsack Problem

- We are given a set $N = \{1, \cdots, n\}$ of items and a capacity W.
- There is a profit p_i and a size w_i associated with each item $i \in N$.
- We want to choose the set of items that maximizes profit subject to the constraint that their total size does not exceed the capacity.
- The most straightforward formulation is to introduce a binary variable x_i associated with each item.
- x_i takes value 1 if item i is chosen and 0 otherwise.
- Then the formulation is

$$\begin{aligned} \min \quad & \sum_{j=1}^{n} p_j x_j \\ \text{s.t.} \quad & \sum_{j=1}^{n} w_j x_j \leq W, \\ & x_i \in \{0,1\}, \quad \forall i \end{aligned}$$

An Alternative Formulation

- Let us call a set $C \subseteq N$ a cover is $\sum_{i \in C} w_i > W$.
- Further, a cover C is minimal if $\sum_{i \in C \setminus \{j\}} w_i \leq W$ for all $j \in C$.
- Then we claim that the following is also a valid formulation of the original problem.

$$\begin{aligned} \max \quad & \sum_{j=1}^{n} p_j x_j, \\ \text{s.t.} \quad & \sum_{j \in C} x_j \leq |C| - 1, \quad \text{for all minimal covers } C \\ & x_i \in \{0,1\}, \quad i \in N \end{aligned}$$

- Which formulation is "better"?

Back to the Facility Location Problem

- Recall our earlier formulation of this problem.
- Here is another formulation for the same problem:

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

- Notice that the set of integer solutions contained in each of the polyhedra is the same (why?).
- However, the second polyhedron is strictly included in the first one (how do we prove this?).
- Therefore, the second polyhedron will yield a better lower bound.
- The second polyhedron is a better approximation to the convex hull of integer solutions.

Formulation Strength and Ideal Formulations

- Consider two formulations A and B for the same ILP.
- Denote the feasible regions corresponding to their LP relaxations as $\mathcal{P}_A$ and $\mathcal{P}_B$.
- Formulation A is said to be at least as strong as formulation B if $\mathcal{P}_A \subseteq \mathcal{P}_B$
- If the inclusion is strict, then A is stronger than B.
- If $\mathcal{S}$ is the set of all feasible integer solutions for the ILP, then we must have $\text{conv}(\mathcal{S}) \subseteq \mathcal{P}_A$ (why?).
- A is ideal if $\text{conv}(\mathcal{S}) = \mathcal{P}_A$.
- If we know an ideal formulation, we can solve the IP (why?).
- How do our formulations of the knapsack problem compare by this measure?

Strengthening Formulations

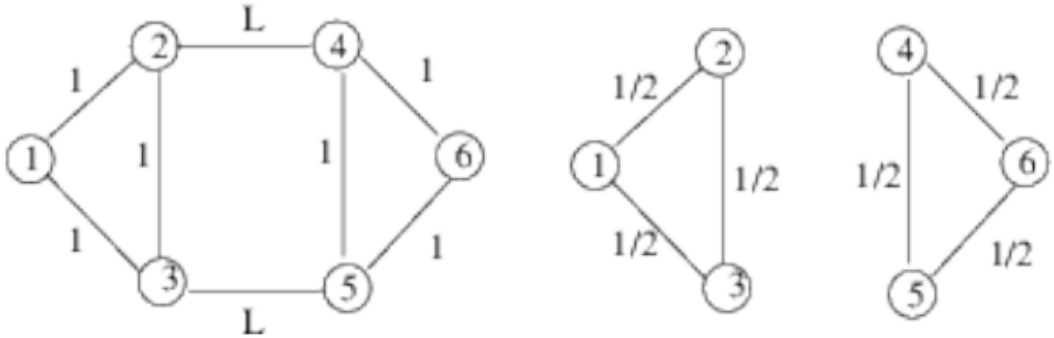
- Often, a given formulation can be strengthened with additional inequalities satisfied by all feasible integer solutions.
- Example: The Perfect Matching Problem
  - We are given a set of n people that need to paired in teams of two.
  - Let c_{ij} represent the "cost" of the team formed by person i and person j.
  - We wish to maximize efficiency over all teams.
  - We can represent this problem on an undirected graph $G = (N, E)$.
  - The nodes represent the people and the edges represent pairings.
  - We have $x_e = 1$ if the endpoints of e are matched, $x_e = 0$ otherwise.

$$\max \quad \sum_{e=\{i,j\}\in E} c_e x_e$$

$$\text{s.t.} \quad \sum_{\{j|\{i,j\}\in E\}} x_{ij} = 1, \qquad \forall i \in N$$

$$x_e \in \{0,1\}, \qquad \forall e = \{i,j\} \in E$$

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 the $|S|$ is odd.
- Note that every perfect matching contains at least one edge from every odd cutset.
- Hence, each odd cutset induces a possible valid inequality.

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

Using the New Formulation

- If we add all of the odd set inequalities, the new formulation is ideal.
- Hence, we can solve this LP and get a solution to the IP.
- However, the number of inequalities is exponential in size, so this is not really practical.
- Recall that only a small number of these inequalities will be active at the optimal solution.
- Later, we will see how we can efficiently generate these inequalities on the fly to solve the IP

Extended Formulations

- We have so far focused on strengthening formulations using additional constraints.
- However, changing the set of variables can also have a dramatic effect.
- Example: A Lot-sizing Problem
  - We want to minimize the costs of production, storage, and set-up.
  - Data for period $t = 1, \cdots, T$:
    - d_t: total demand,
    - c_t: production set-up cost,
    - p_t: unit production cost,
    - h_t: unit storage cost.
  - Variables for period $t = 1, \cdots, T$:
    - s_t: inventory level corresponding at the end of period t
    - y_t: production quantity in period t
    - x_t: binary setup variables. $x_t = 1$ if the resource is setup in period t, and 0 otherwise.

Lot-sizing: The "natural" formulation

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

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

- Here, $\omega_t = \sum_{i=t}^{T} d_i$ 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_{ti} to be the production in period t designated to satisfy demand in period $i \geq t$.
- Now, $y_t = \sum_{i=t}^{T} q_{ti}$.
- With the new set of variables, we can impose the tighter constraint

$$q_{ti} \leq d_i x_t \text{ for } i = 1, \cdots, T \text{ and } t = 1, \cdots, T.$$

- The additional variables strengthen the formulation.
- Again, this in contrary to conventional wisdom for formulating linear programs.

An Ideal Formulation for Lot-sizing

- We can further strengthen the formulation by adding the constraint

$$\sum_{t=1}^{i} q_{ti} \geq d_i, \text{ for } i = 1, \cdots, T.$$

- In fact, adding these additional constraints makes the formulation ideal.
- If we project into the original space, we will get the convex hull of solutions to the first formulation.

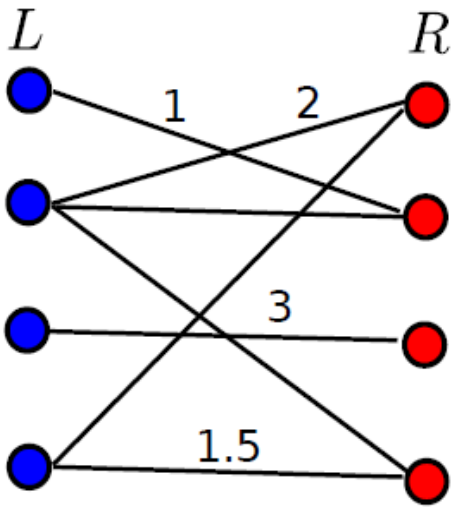
Contrast with Linear Programming

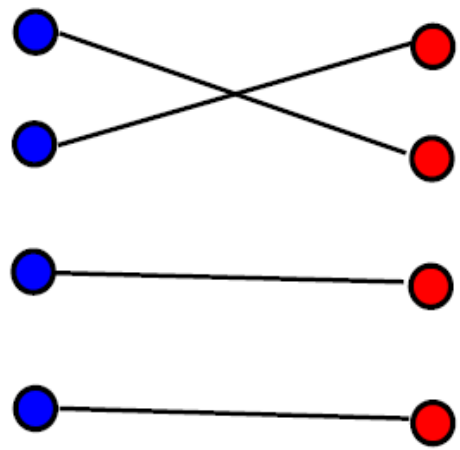
- In linear programming, the same problem can also have multiple formulations.
- In LP, however, conventional wisdom is that bigger formulations take longer to solve.
- In IP, this conventional wisdom does not hold.
- We have already seen two examples where it is not valid.
- Generally speaking, the size of the formulation does not determine how difficult the IP is.

The Max-Weight Bipartite Matching Problem

Given a bipartite graph G = (N, A), with N = L $\cup$ R, and weights w_{ij} on edges (i,j), find a maximum weight matching.

- Matching: a set of edges covering each node at most once
- Let n=|N| and m = |A|.
- Equivalent to maximum weight / minimum cost perfect matching.





The Max-Weight Bipartite Matching

Integer Programming (IP) formulation

$$\begin{aligned} \max \quad & \sum_{ij} w_{ij} x_{ij} \\ \text{s.t.} \quad & \sum_{j} x_{ij} \leq 1, \forall i \in L \\ & \sum_{i} x_{ij} \leq 1, \forall j \in R \\ & x_{ij} \in \{0,1\}, \forall (i,j) \in A \end{aligned}$$

- $x_{ij} = 1$ indicate that we include edge (i, j) in the matching
- IP: non-convex feasible set

The Max-Weight Bipartite Matching

Integer program (IP)

$$\max \quad \sum_{ij} w_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j} x_{ij} \leq 1, \forall i \in L$$

$$\sum_{i} x_{ij} \leq 1, \forall j \in R$$

$$x_{ij} \in \{0,1\}, \forall (i,j) \in A$$

LP relaxation

$$\max \quad \sum_{ij} w_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j} x_{ij} \leq 1, \forall i \in L$$

$$\sum_{i} x_{ij} \leq 1, \forall j \in R$$

$$x_{ij} \geq 0, \forall (i,j) \in A$$

- Theorem. The feasible region of the matching LP is the convex hull of indicator vectors of matchings.
- This is the strongest guarantee you could hope for an LP relaxation of a combinatorial problem
- Solving LP is equivalent to solving the combinatorial problem

Primal-Dual Interpretation

Primal LP relaxation

$$\max \quad \sum_{ij} w_{ij} x_{ij}$$

$$\text{s.t.} \quad \sum_{j} x_{ij} \leq 1, \forall i \in L$$

$$\sum_{i} x_{ij} \leq 1, \forall j \in R$$

$$x_{ij} \geq 0, \forall (i,j) \in A$$

Dual

$$\min \quad \sum_{i} y_i$$

$$\text{s.t.} \quad y_i + y_j \geq w_{ij}, \forall (i,j) \in A$$

$$y \geq 0$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$

Primal-Dual Interpretation

Suppose edge weights $w_{ij} = 1$, then binary solutions to the dual are node covers.

Dual

$$\begin{aligned} \min \quad & \sum_i y_i \\ \text{s.t.} \quad & y_i + y_j \geq 1, \forall (i,j) \in A \\ & y \geq 0 \end{aligned}$$

Dual Integer Program

$$\begin{aligned} \min \quad & \sum_i y_i \\ \text{s.t.} \quad & y_i + y_j \geq 1, \forall (i,j) \in A \\ & y \in \{0,1\} \end{aligned}$$

- Dual problem is solving minimum vertex cover: find smallest set of nodes S such that at least one end of each edge is in S
- From strong duality theorem, we know $P^*_{LP} = D^*_{LP}$
- Consider IP formulation of the dual, then

$$P^*_{IP} \leq P^*_{LP} = D^*_{LP} \leq D^*_{IP}$$

Total Unimodularity

Defintion: A matrix A is Totally Unimodular if every square submatrix has determinant 0, +1 or -1.

Theorem: If $A \in \mathbb{R}^{m \times n}$ is totally unimodular, and b is an integer vector, then $\{x : Ax \leq b; x \geq 0\}$ has integer vertices.

- Non-zero entries of vertex x are solution of $A'x' = b'$ for some nonsignular square submatrix A' and corresponding sub-vector b'
- Cramer's rule:

$$x_i = \frac{\det(A'_i \mid b')}{\det A'}$$

Claim: The constraint matrix of the bipartite matching LP is totally unimodular.

The Minimum weight vertex cover

- undirected graph G = (N, A) with node weights $w_i \geq 0$
- A vertex cover is a set of nodes S such that each edge has at least one end in S
- The weight of a vertex cover is sum of all weights of nodes in the cover
- Find the vertex cover with minimum weight

Integer Program

$$\min \quad \sum_i w_i y_i$$

$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$

$$y \in \{0,1\}$$

LP Relaxation

$$\min \quad \sum_i w_i y_i$$

$$\text{s.t. } y_i + y_j \geq 1, \forall (i,j) \in A$$

$$y \geq 0$$

LP Relaxation for the Minimum weight vertex cover

- In the LP relaxation, we do not need $y \leq 1$, since the optimal solution y^* of the LP does not change if $y \leq 1$ is added.
  Proof: suppose that there exists an index i such that the optimal solution of the LP y_i^* is strictly larger than one. Then, let y' be a vector which is same as y^* except for $y'_i = 1 < y_i^*$. This y' satisfies all the constraints, and the objective function is smaller.
- The solution of the relaxed LP may not be integer, i.e., $0 < y_i^* < 1$
- rounding technique:

$$y'_i = \begin{cases} 0, & \text{if } y_i^* < 0.5 \\ 1, & \text{if } y_i^* \geq 0.5 \end{cases}$$

- The rounded solution y' is feasible to the original problem

LP Relaxation for the Minimum weight vertex cover

The weight of the vertex cover we get from rounding is at most twice as large as the minimum weight vertex cover.

- Note that $y'_i = \min(\lfloor 2y^*_i \rfloor, 1)$
- Let P^*_{IP} be the optimal solution for IP, and P^*_{LP} be the optimal solution for the LP relaxation
- Since any feasible solution for IP is also feasible in LP, $P^*_{LP} \leq P^*_{IP}$
- The rounded solution y' satisfy

$$\sum_i y'_i w_i = \sum_i \min(\lfloor 2y^*_i \rfloor, 1) w_i \leq \sum_i 2y^*_i w_i = 2P^*_{LP} \leq 2P^*_{IP}$$

Outline

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 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

$$x \in \bigcup_{1 \le i \le k} X_i, \quad X_i \subseteq \mathbb{R}^n.$$

- 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

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

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

Two Conceptual Reformulations

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

$$\max \left\{ c^\top x \mid x \in \bigcup_{1 \le i \le k} \mathcal{P}_i + \text{intcone}\{r^1, \cdots, r^t\} \right\}$$

- The second is in terms of valid inequalities

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

  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 $\mathcal{S} \subseteq \cup_{1 \leq i \leq k} X_i$, the disjunction associated with $\{X_i\}_{i=1}^k$ is said to be valid for an MILP with feasible set $\mathcal{S}$.
  - **Definition 2.** Let $\{X_i\}_{i=1}^k$ is a disjunction valid for $\mathcal{S}$, and X_i is polyhedral for all i, then we say the disjunction is linear.
  - **Definition 3.** Let $\{X_i\}_{i=1}^k$ is a disjunction valid for $\mathcal{S}$, and $X_i \cap X_j = \emptyset$ for all i, j then we say the disjunction is partitive.
  - **Definition 4.** Let $\{X_i\}_{i=1}^k$ 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 (π, π_0) is called a valid inequality for $\mathcal{S}$ if $\pi^\top x \leq \pi_0, \forall x \in \mathcal{S}$.
- Note (π, π_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 a polyhedral.
- Then $\max_{X_i \cap \mathcal{S}} c^\top x$ is an MILP for all $i \in 1, \ldots, k$.
- For each i, 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 $\{\mathcal{P}_i\}$ is itself a valid linear disjunction.

Optimality Conditions

- 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,

$$\max_{i \in 1,\cdots,k} \max_{x \in \mathcal{P}_i \cap \mathbb{R}^n_+} c^\top x \geq z_{\text{IP}}.$$

- If we have $x^* \in \mathcal{S}$ such that

$$\max_{i \in 1,\cdots,k} \max_{x \in \mathcal{P}_i \cap \mathbb{R}^n_+} c^\top x = c^\top x^*$$

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 the relaxation, 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 $\mathcal{S}_1, \cdots, \mathcal{S}_k$. Then

$$\max_{x \in \mathcal{S}} c^\top x = \max_{1 \le i \le k} \{ \max_{x \in \mathcal{S}_i} c^\top x \}.$$

- 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., $\mathcal{S}_i = S \cap X_i$.
- We only consider linear disjunctions so that the subproblem remain MILPs after branching.
- 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 $x^* \notin \cup_i X_i$, where x^* is the (infeasible) solution produced by solving the bounding problem associated with a given subproblem.
- A typical disjunction is

$$X_1 = \{x_j \geq \lceil x_j^* \rceil\}$$
$$X_2 = \{x_j \leq \lfloor x_j^* \rfloor\}$$

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

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.
- Solving the bounding problem serves two purposes.
  - In some cases, the solution x^* to the relaxation may actually be a feasible solution, in which case $c^\top x^*$ is a global lower bound $l(\mathcal{S})$.
  - Bounding enables us to inexpensively obtain 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 the these is to form the LP relaxation obtained by dropping the integrality constraints.
- For the rest of the lecture, assume all variables have finite upper and lower bounds.

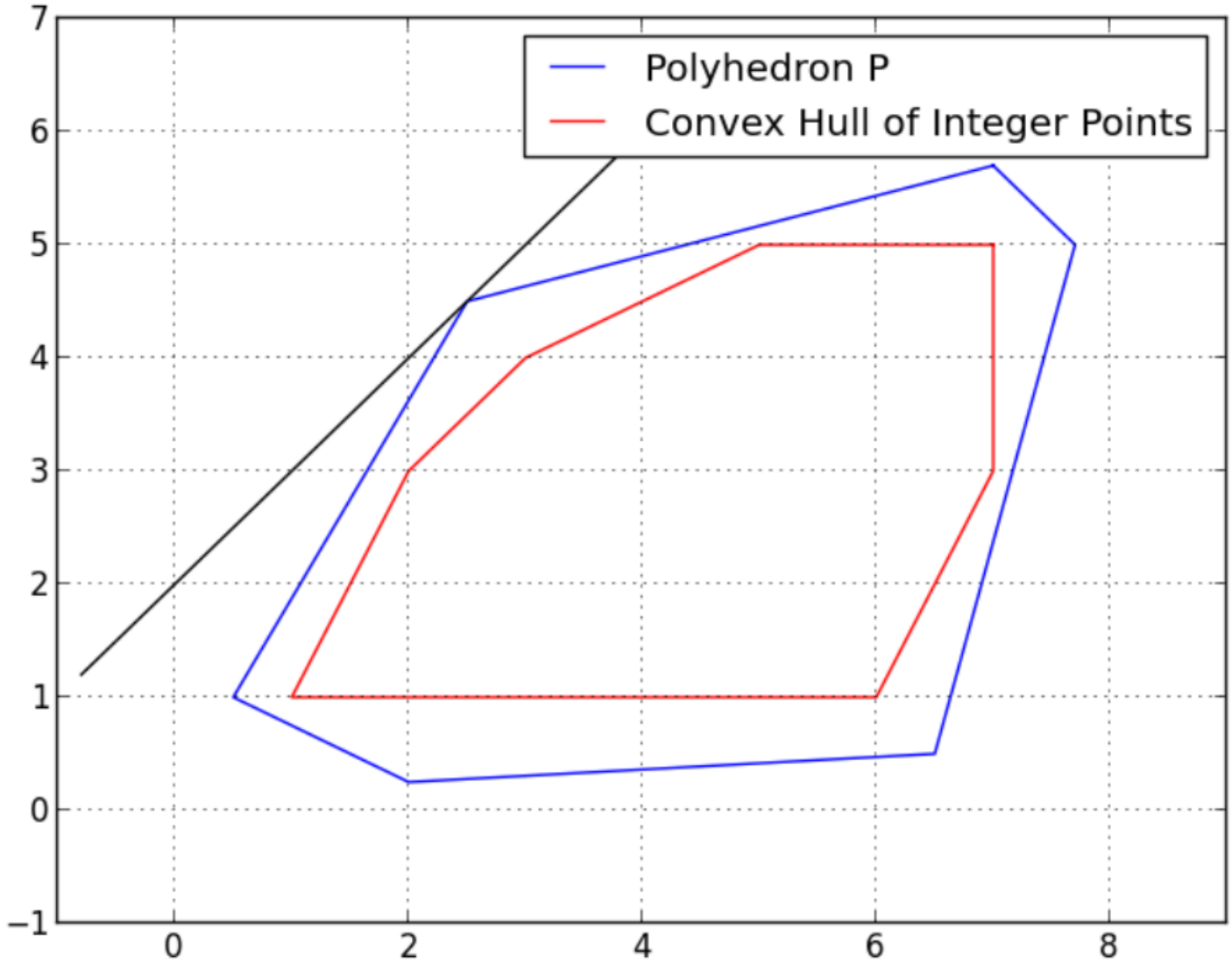
LP-based Branch and Bound: Initial Subproblem

- In LP-based branch and bound, we first solve the LP relaxation of the original problem. The result is one of the following:
  - The LP is infeasible $\Rightarrow$ MILP is infeasible.
  - We obtain a feasible solution for the MILP $\Rightarrow$ optimal solution.
  - 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.

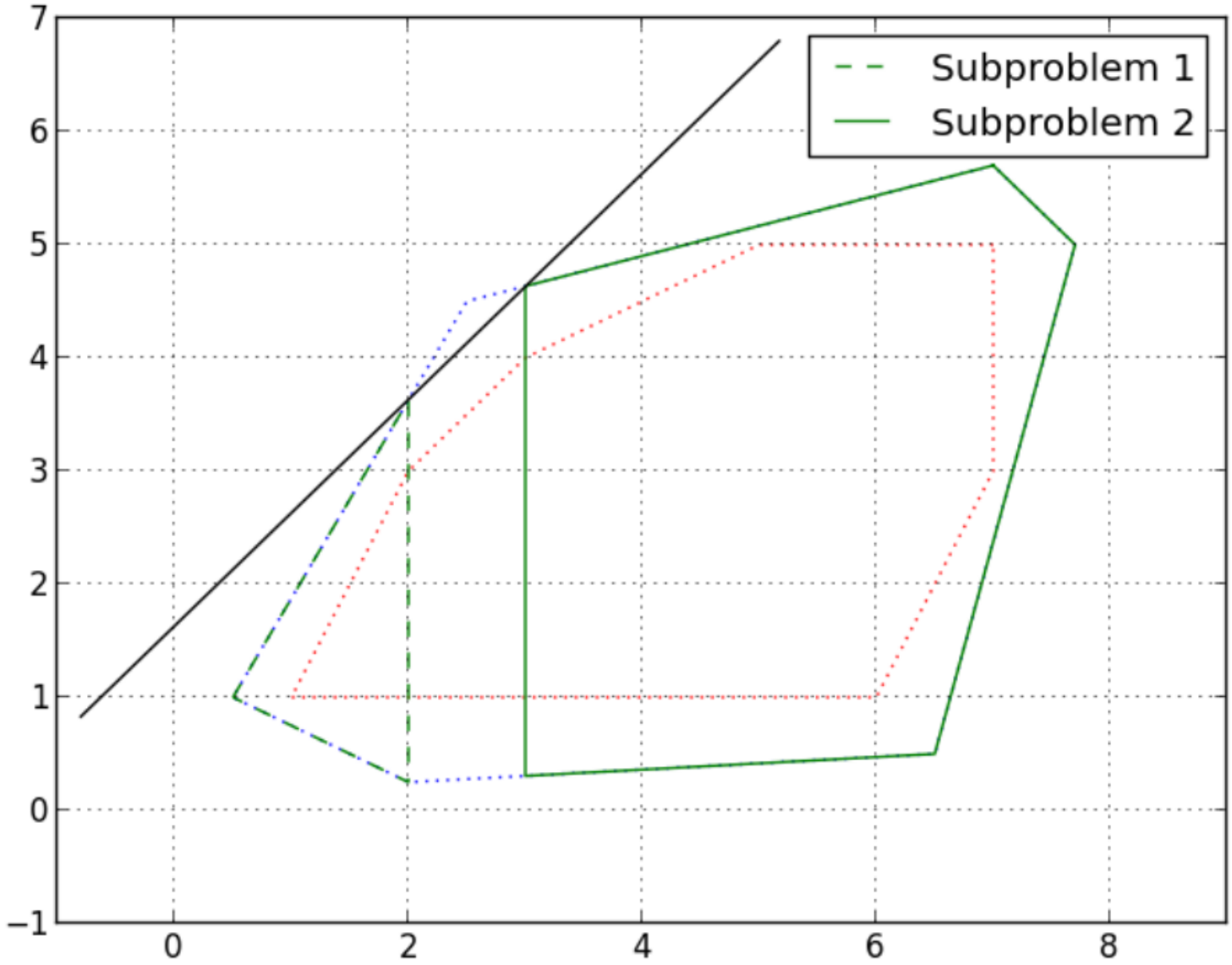
Branching in LP-based Branch and Bound

- 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



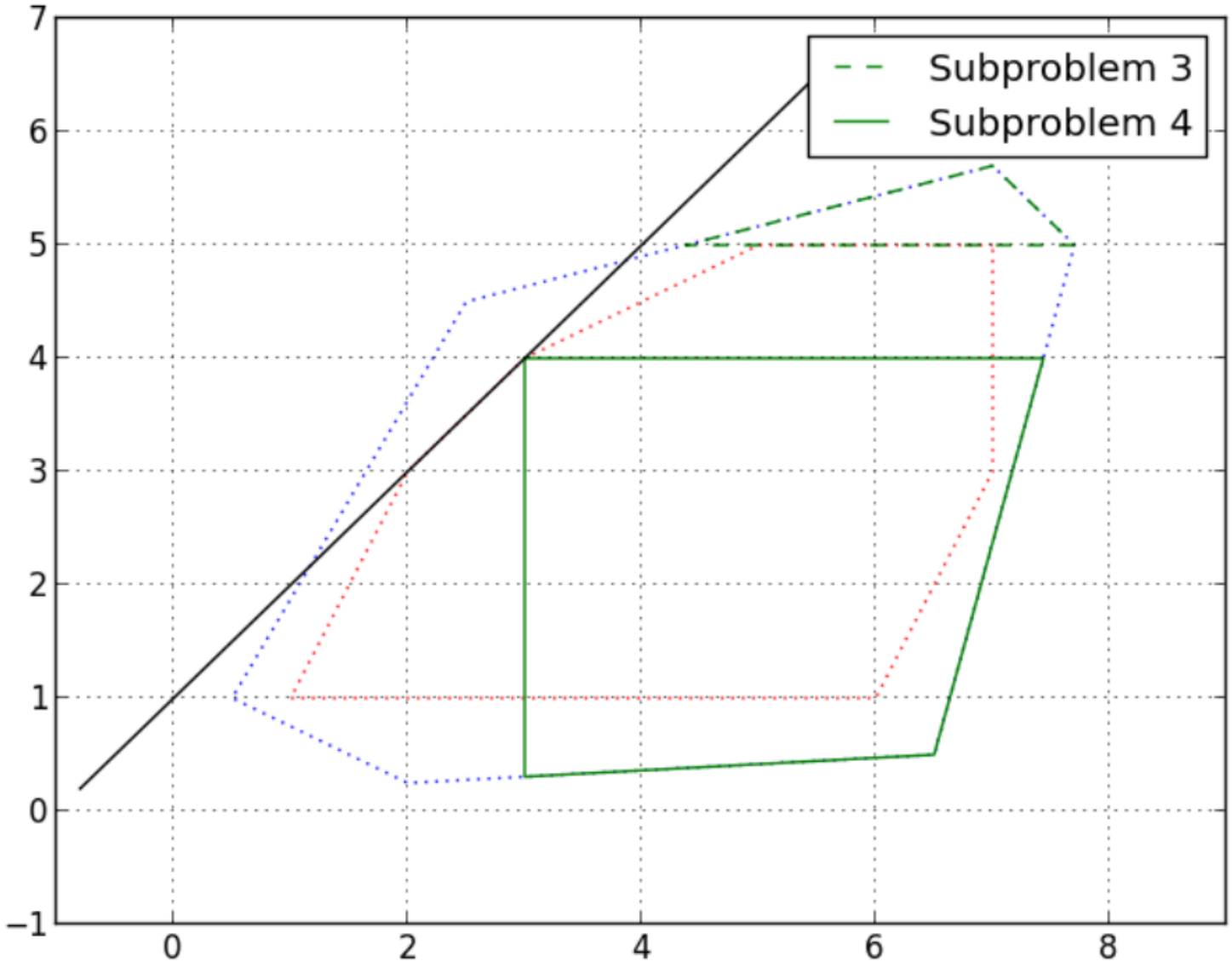
The Geometry of Branching (cont'd)



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.

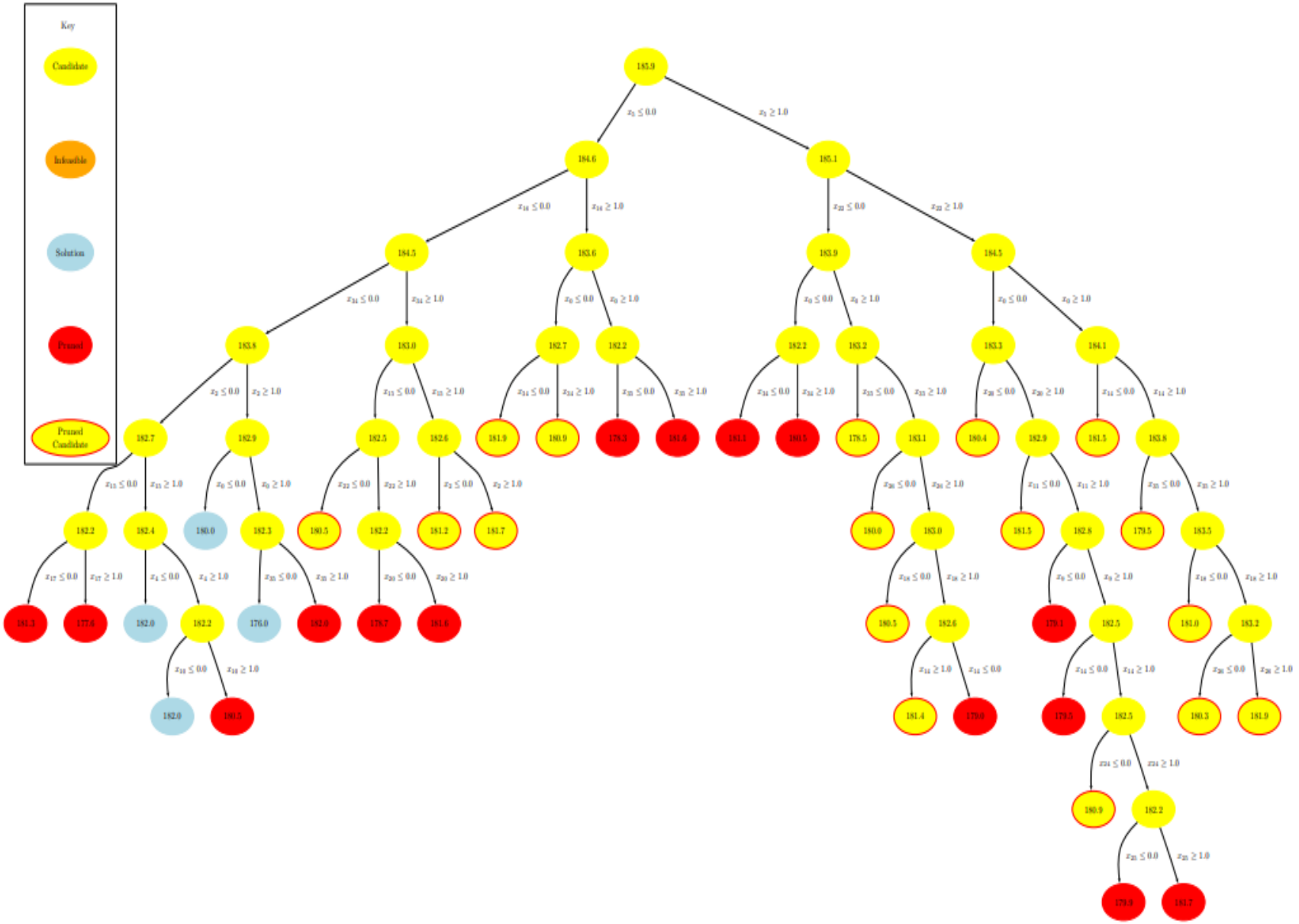
The Geometry of Branching



LP-based Branch and Bound Algorithm

- To start, derive a lower bound L using a heuristic method.
- Put the original problem on the candidate list.
- Select a problem $\mathcal{S}$ from the candidate list and solve the LP relaxation to obtain the bound $b(\mathcal{S})$.
  - If the LP is infeasible $\Rightarrow$ node can be pruned.
  - Otherwise, if $b(\mathcal{S}) \leq L \Rightarrow$ node can be pruned.
  - Otherwise, if $b(\mathcal{S}) > L$ and the solution is feasible for the MILP $\Rightarrow$ set $L \leftarrow b(\mathcal{S})$.
  - Otherwise, branch and add the new subproblem to the candidate list.
- If the candidate list in 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.

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.

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.

Outline

The Efficiency of Branch and Bound

- In general, our goal is to solve the problem at hand as quickly as possible.
- The overall solution time is the product of the number of nodes enumerated and the time to process each node.
- Typically, by spending more time in processing, we can achieve a reduction in tree size by computing stronger bounds.
- This highlights another of the many tradeoffs we must navigate.
- Our goal in bounding is to achieve a balance between the strength of the bound and the efficiency.
- How do we compute bounds?
  - Relaxation: Relax some of the constraints and solve the resulting mathematical optimization problem.
  - Duality: Formulate a "dual" problem and find a feasible to it.
- In practice, we will use both of these two approaches.

Relaxation

- As usual, we consider the MILP

$$z_{\text{IP}} = \max\{c^\top x \mid x \in \mathcal{S}\}$$

where

$$\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$$
$$\mathcal{S} = \mathcal{P} \cap (\mathbb{Z}_+^p \times \mathbb{R}_+^{n-p}).$$

- **Definition 1.** A relaxation of IP is a maximization problem defined as

$$z_R = \max\{z_R(x) | x \in \mathcal{S}_R\}$$

with the following two properties:

$$\mathcal{S} \subseteq \mathcal{S}_R$$
$$c^\top x \leq z_R(x), \quad \forall x \in \mathcal{S}$$

Importance of Relaxations

- The main purpose of a relaxation is to obtain an upper bound on z_{IP} .
- Solving a relaxation is is one simple method of bounding in branch and bound.
- The idea is to choose a relaxation that is much easier to solve than the original problem, but still yields a bound that is "strong enough."
- Note that the relaxation must be solved to optimality to yield a valid bound.
- We consider three types of "formulation-based" relaxations.
  - LP relaxation
  - Combinatorial relaxation
  - Lagrangian relaxation
- Relaxations are also used in some other bounding schemes we'll look at.

Obtaining and Using Relaxations

- Properties of relaxations
  - If a relaxation of (MILP) is infeasible, then so is (MILP).
  - If $z_R(x) = c^\top x$, then for $x^* \in \operatorname{argmax}_{x \in \mathcal{S}_R} Z_R(x)$, if $x^* \in \mathcal{S}$, then x^* is optimal for (MILP).
- The easiest way to obtain relaxations of IP is to drop some of the constraints defining the feasible set $\mathcal{S}$.
- It is "obvious" how to obtain an LP relaxation, but combinatorial relaxations are not as obvious.

Example: Traveling Salesman Problem

Given a graph $G = (V, E)$, a salesman must visit each of all nodes exactly once and then return to his starting point. A feasible solution is a subset of E consisting of edges to form the route.
IP Formulation:

$$\sum_{j:j\neq i}^{n} x_{ij} = 1, \quad \forall i \in V$$

$$\sum_{i:i\neq j}^{n} x_{ij} = 1, \quad \forall j \in V$$

$$\sum_{i\in S}\sum_{j\notin S} x_{ij} \geq 1, \quad \forall S \subset V, S \neq \emptyset$$

where x_{ij} is a binary variable indicating whether the salesman goes directly from town i to town j.

Combinatorial Relaxations of the TSP

- The Traveling Salesman Problem has several well-known combinatorial relaxations.
- Assignment Problem
  - The problem of assigning n people to n different tasks.
  - Can be solved in polynomial time.
  - Obtained by dropping the subtour elimination constraints.
- Minimum 1-tree Problem
  - A 1-tree in a graph is a spanning tree of nodes $\{2, \cdots, n\}$ plus exactly two edges incident to node one.
  - A minimum 1-tree can be found in polynomial time.
  - This relaxation is obtained by dropping all subtour elimination constraints involving node 1 and also all degree constraints not involving node 1.

Exploiting Relaxations

- How can we use our ability to solve a relaxation to full advantage?
- The most obvious way is simply to straightforwardly use the relaxation to obtain a bound.
- However, by solving the relaxation repeatedly, we can get additional information.
- For example, we can generate extreme points of $\text{conv}(\mathcal{S}_R)$.
- In an indirect way (using the Farkas Lemma), we can even obtain facet-defining inequalities for conv($\mathcal{S}_R$).
- We can use this information to strengthen the original formulation.
- This is one of the basic principles of many solution methods.

Lagrangian Relaxation

- The idea is again based on relaxing a set of constraints from the original formulation.
- We try to push the solution towards feasibility by penalizing violation of the dropped constraints.
- Suppose our IP is defined by

$$\begin{aligned} \max \quad & c^\top x \\ \text{s.t.} \quad & A^1 x \leq b^1 \\ & A^2 x \leq b^2 \\ & x \in \mathbb{Z}^n_+ \end{aligned}$$

  where optimizing over $Q = \{x \in \mathbb{Z}^n_+ \mid A^2 x \leq b^2\}$ is "easy."
- Lagrangian Relaxation:

$$LR(\lambda) : Z_R(\lambda) = \max_{x \in Q} \{(c - \lambda A^1)^\top x + \lambda^\top b^1)\}.$$

Properties of the Lagrangian Relaxation

- For any $\lambda \geq 0$, $LR(\lambda)$ is a relaxation of IP (why?).
- Solving $LR(\lambda)$ yields an upper bound on the value of the optimal solution.
- Because of our assumptions, $LR(\lambda)$ can be solved easily.
- Recalling LP duality, one can think of λ as a vector of "dual variables."
- If the solution to the relaxation is integral, it is optimal if the primal and dual solutions are complementary, as in LP.

A (Very) Brief Tour of Duality

- If we have a dual to the primal problem, then we can easily obtain bounds on the value of an optimal solution.
- The advantage of a dual is that we need not solve it to optimality.
- Any feasible solution to the dual yields a valid bound.
- The three main categories of duals used computationally are
  - LP duals
  - Lagrangian duals

A Quick Overview of LP Duality

- We consider the LP relaxation of (MILP) in standard form

$$\{x \in \mathbb{R}^n_+ | \bar{A}x = b\}$$

  where $\bar{A} = [A|I]$ (the additional columns are for the slack variables).
- Recall that there always exists an optimal solution that is basic.
- We construct basic solutions by
  - Choosing a basis B of m linearly independent columns of $\bar{A}$.
  - Solving the system $Bx_B = b$ to obtain the values of the basic variables.
  - Setting remaining variables to value 0.
- If $x_B \geq 0$, then the associated basic solution is feasible.
- With respect to any basic feasible solution, it is easy to determine the impact of increasing a given activity.
- The reduced cost

$$\bar{c}_j = c_j - c_B^\top B^{-1} \bar{A}_j$$

  of (nonbasic) variable j tells us how the objective function value changes if we increase the level of activity j by one unit.

The LP Value Function

- From the resource (dual) perspective, the quantity $u = c_B B^{-1}$ is a vector that tells us the marginal economic value of each resource.
- Thus, the vector u gives us a **price** for each resource.
- This price vector can be seen as the gradients of the value function

$$\phi_{LP}(\beta) = \max_{x \in \mathcal{S}(\beta)} c^\top x,$$

  of an LP, where for a given $\beta \in \mathbb{R}^m, \mathcal{S}(d) = \{x \in \mathbb{R}^n_+ | \bar{A}x = d\}$
- We let $\phi_{LP}(\beta) = -\infty$ if $\mathcal{S}(\beta) = \emptyset$.
- These gradients can be seen as linear over-estimators of the value function.
- The dual problems we'll consider are essentially aimed at producing such over-estimators.
- We'll generalize to non-linear functions.

The LP Dual

- To understand the structure of the value function in more detail, first note that it is easy to see ϕ_{LP} is concave.
- Now consider an optimal basis matrix B for the LP.
  - The gradient of ϕ_{LP} at b is $\hat{u} = c_B B^{-1}$.
  - Since $\phi_{LP}(b) = \hat{u}^\top b$ and ϕ_{LP} is concave, we know that $\phi_{LP}(\beta) \leq \hat{u}^\top \beta$ for all β.
- The traditional LP dual problem can be viewed as that of finding a linear function that bounds the value function from above and has minimum value at b.

The LP Dual (cont'd)

- As we have seen, for any $u \in \mathbb{R}^m$, the following gives a upper bound on $\phi_{LP}(b)$.

$$\begin{aligned} g(u) = \max_{x \geq 0} [c^\top x + u^\top (b - \bar{A}x)] &\geq c^\top x^* + u^\top (b - \bar{A}x^*) \\ &= c^\top x^* \\ &= \phi_{LP}(b) \end{aligned}$$

- With some simplification, we can obtain an explicit form for this function

$$\begin{aligned} g(u) &= \max_{x \geq 0} [c^\top x + u^\top (b - \bar{A}x)] \\ &= u^\top b + \max_{x \geq 0} (c^\top - u^\top \bar{A})x \end{aligned}$$

- Note that

$$\max_{x \geq 0} (c^\top - u^\top \bar{A})x = \begin{cases} 0, & \text{if } c^\top - u^\top \bar{A} \leq \mathbf{0}^\top \\ \infty, & \text{otherwise} \end{cases}$$

The LP Dual (cont'd)

- So we have

$$g(u) = \begin{cases} u^\top b, & \text{if } c^\top - u^\top \bar{A} \leq \mathbf{0}^\top \\ \infty, & \text{otherwise} \end{cases}$$

  which is again a linear over-estimator of the value function.

- An LP dual problem is obtained by computing the strongest linear over-estimator with respect to b.

$$\min_{u \in \mathbb{R}^m} g(u) = \min\{b^\top u \mid u^\top \bar{A} \geq c^\top\}$$

Combinatorial Representation of the LP Value Function

- From the fact that there is always an extremal optimum to (LPD), we conclude that the LP value function can be described combinatorially.

$$\phi_{LP}(\beta) = \min_{u \in \mathcal{E}} u^\top \beta$$

for $\beta \in \mathbb{R}^m$, where

$$\mathcal{E} = \{c_B \bar{A}_E^{-1} \mid E \text{ is the index set of a dual feasible bases of } \bar{A}\}$$

- Note that $\mathcal{E}$ is also the set of extreme points of the dual polyhedron $\{u \in \mathbb{R}^m \mid u^\top \bar{A} \geq c^\top\}$

The MILP Value Function

- We now generalize the notions seen so far to the MILP case.
- The value function associated with (MILP) is

$$\phi(\beta) = \max_{x \in \mathcal{S}(\beta)} c^\top x$$

  for $\beta \in \mathbb{R}^m$, where $\mathcal{S}(\beta) = \{x \in \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+ \mid \bar{A}x = \beta\}$.
- Again, we let $\phi(\beta) = -\infty$ if $\beta \in \{\beta \in \mathbb{R}^m \mid \mathcal{S}(\beta) = \emptyset\}$.

A General Dual Problem

- A dual function $F : \mathbb{R}^m \to \mathbb{R}$ is one that satisfies $F(\beta) \geq \phi(\beta)$ for all $\beta \in \mathbb{R}^m$.
- How to select such a function?
- We choose may choose one that is easy to construct/evaluate or for which $F(b) \approx \phi(b)$.
- This results in the following generalized dual of (MILP).

$$\min \quad \{F(b) : F(\beta) \geq \phi(\beta), \beta \in \mathbb{R}^m, F \in \mathbf{Y}^m\} \qquad (D)$$

  where $\mathbf{Y}^m \subseteq \{f \mid f : \mathbb{R}^m \to \mathbb{R}\}$.
- We call F^* strong for this instance if F^* is a feasible dual function and $F^*(b) = \phi(b)$.
- This dual instance always has a solution F^* that is strong if the value function is bounded and $\mathbf{Y}^m = \{f \mid f : \mathbb{R}^m \to \mathbb{R}\}$. Why?

LP Dual Function

- It is straightforward to obtain a dual function using linear programming.
- Simply take the dual of any LP relaxation.
- In practice, working with this dual just means using dual simplex to solve the relaxations.
- Note again that since dual simplex maintains a dual feasible solution at all times, we can stop anytime we like.
- In particular, as soon as the upper bound goes below the current lower bound, we can stop solving the LP.
- This can save significant effort.
- With an LP dual, we can "close the gap" by adding valid inequalities to strengthen the LP relaxation.
- The size of the gap in this case is a measure of how well we are able to approximate the convex hull of feasible solutions (near the optimum).

The Lagrangian Dual

- We can obtain a dual function from a Lagrangian relaxation by letting

$$L(\beta, u) = \max_{x \in \mathcal{S}_R(\beta)} (c - uA'')^\top x + u^\top \beta''$$

  where $\mathcal{S}_R(d) = \{x \in \mathbb{Z}_+^n \mid A'x \leq d\}$

- Then the Lagrangian dual function, ϕ_{LD}, is

$$\phi_{LD}(\beta) = \min_{u \geq 0} L(\beta, u)$$

Dual Functions from Branch-and-Bound

- As before, let $\mathcal{T}$ be set of the terminating nodes of the tree. Then, assuming we are branching on variable disjunctions, in a leaf node $t \in \mathcal{T}$,the relaxation we solve is:

$$\phi^t(\beta) = \max_{x \geq 0} \{c^\top x \mid \bar{A}x = \beta, l^t \leq x \leq u^t\}.$$

- The dual at node t:

$$\begin{aligned} \min \quad & \pi^t \beta + \underline{\pi}^t l^t + \bar{\pi}^t u^t \\ \text{s.t.} \quad & \pi^t A + \underline{\pi}^t + \bar{\pi}^t \geq c^\top, \\ & \underline{\pi} \geq 0, \bar{\pi} \leq 0 \end{aligned}$$

- We obtain the following strong dual function:

$$\max_{t \in \mathcal{T}} \{\hat{\pi}^t \beta + \hat{\underline{\pi}}^t l^t + \hat{\bar{\pi}}^t u^t\}$$

where $(\hat{\pi}^t, \hat{\underline{\pi}}^t, \hat{\bar{\pi}})$ is an optimal solution to the dual at node t.

The Duality Gap

- In most cases, the the value of an optimal solution to a given dual problem is not equal to the value of an optimal solution to (MILP).
- The difference between these values for a particular instance is known as the duality gap or just the gap.
- It is typically reported as a percentage of the value of the best known solution (this is called the relative gap).
- The size of the relative gap is a rough measure of the difficulty of a problem.
- It can help us estimate how long it will take to solve a given problem by branch and bound.

Strong Duality

- When the duality gap is guaranteed to be zero, we say we have a strong dual.
- For linear programs, the LP dual is a strong dual.
- For integer programs, the dual (D) is a strong dual, since the value function itself is a solution for which the gap is zero.
- Of course, obtaining a description of the value function is more difficult than solving the integer program itself.

Outline

Branching

- We have now spent several lectures discussing methods for bounding.
- Obtaining tight bounds is the most important aspect of the branch-and-bound algorithm.
- Branching effectively is a very close second.
- Choosing an effective method of branching can make orders of magnitude difference in the size of the search tree and the solution time.

Disjunctions and Branching

- Recall that branching is generally achieved by selecting an admissible disjunction $\{X_i\}_{i=1}^k$ and using it to partition $\mathcal{S}$, e.g., $\mathcal{S}_i = \mathcal{S} \cap X_i$.
- The way this disjunction is selected is called the branching method and is the topic we now examine.
- Generally speaking, we want $x^* \notin \cup_{1 \leq i \leq k} X_i$, where x^* is the (infeasible) solution produced by solving the bounding problem associated with a given subproblem.

Split Disjunctions

- The most easily handled disjunctions are those based on dividing the feasible region using a given hyperplane.
- In such cases, each term of the disjunction can be imposed by addition of a single inequality.
- A hyperplane defined by a vector $\pi \in \mathbb{R}^n$ is said to be integer if $\pi_i \in \mathbb{Z}$ for $0 \leq i \leq p$ and $\pi_i = 0$ for $p+1 \leq i \leq n$.
- Note that if π is integer, then we have $\pi^\top x \in \mathbb{Z}$ whenever $x \in \mathbb{Z}^p \times \mathbb{R}^{n-p}$.
- Then the disjunction defined by

$$X_1 = \{x \in \mathbb{R}^n \mid \pi^\top x \leq \pi_0\}, X_2 = \{x \in \mathbb{R}^n \mid \pi^\top x \geq \pi_0 + 1\},$$

  is valid when $\pi_0 \in \mathbb{Z}$.
- This is known as a split disjunction.

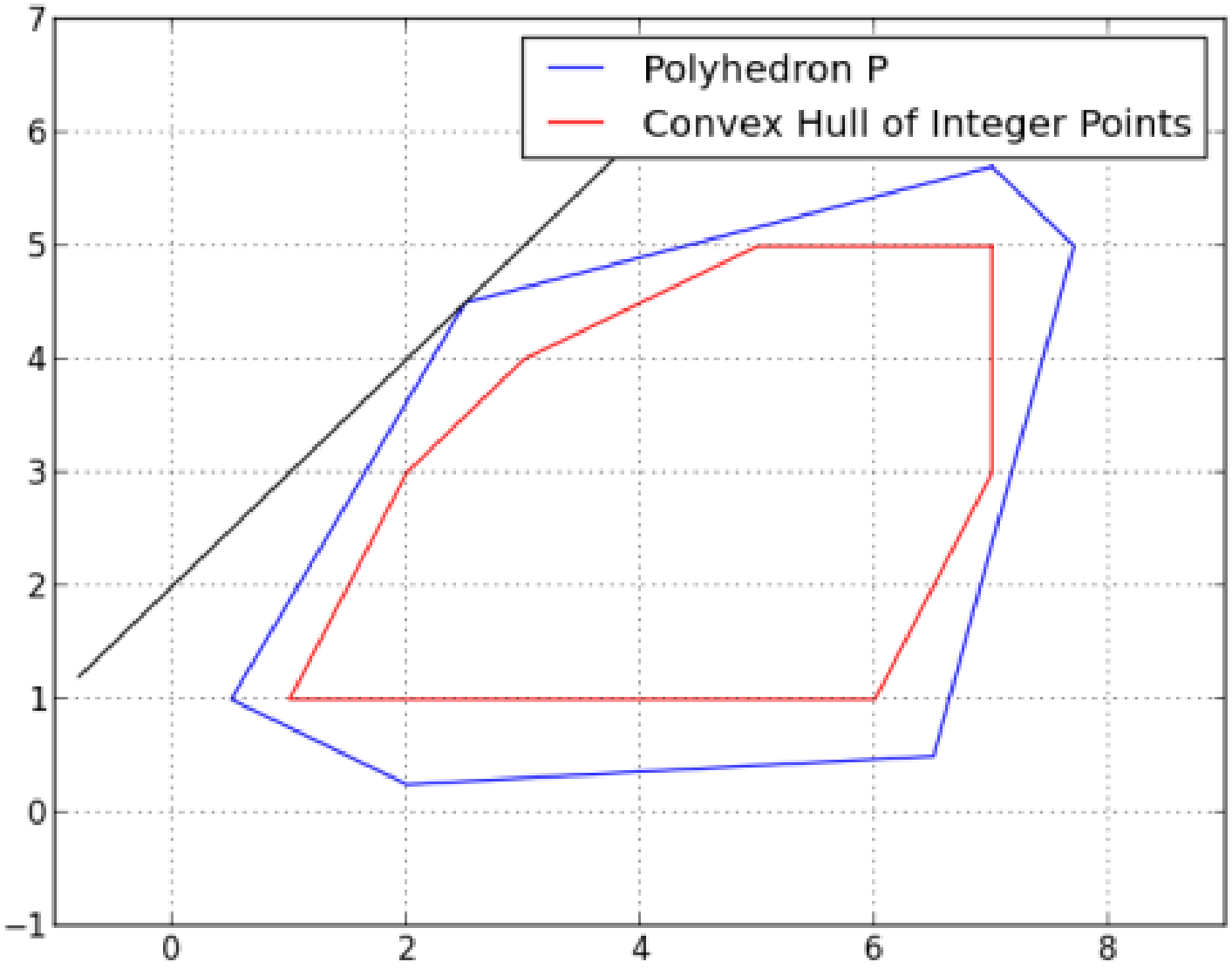
Variable Disjunctions

- The simplest split disjunction is to take $\pi = e_i$ for $0 \leq i \leq p$, where e_i is the i^{th} unit vector.
- If we branch using such a disjunction, we simply say we are branching on x_j.
- For such a branching disjunction to be admissible, we should have $\pi_0 < x_i^* < \pi_0 + 1$.
- In the special case of a 0-1 IP, this dichotomy reduces to

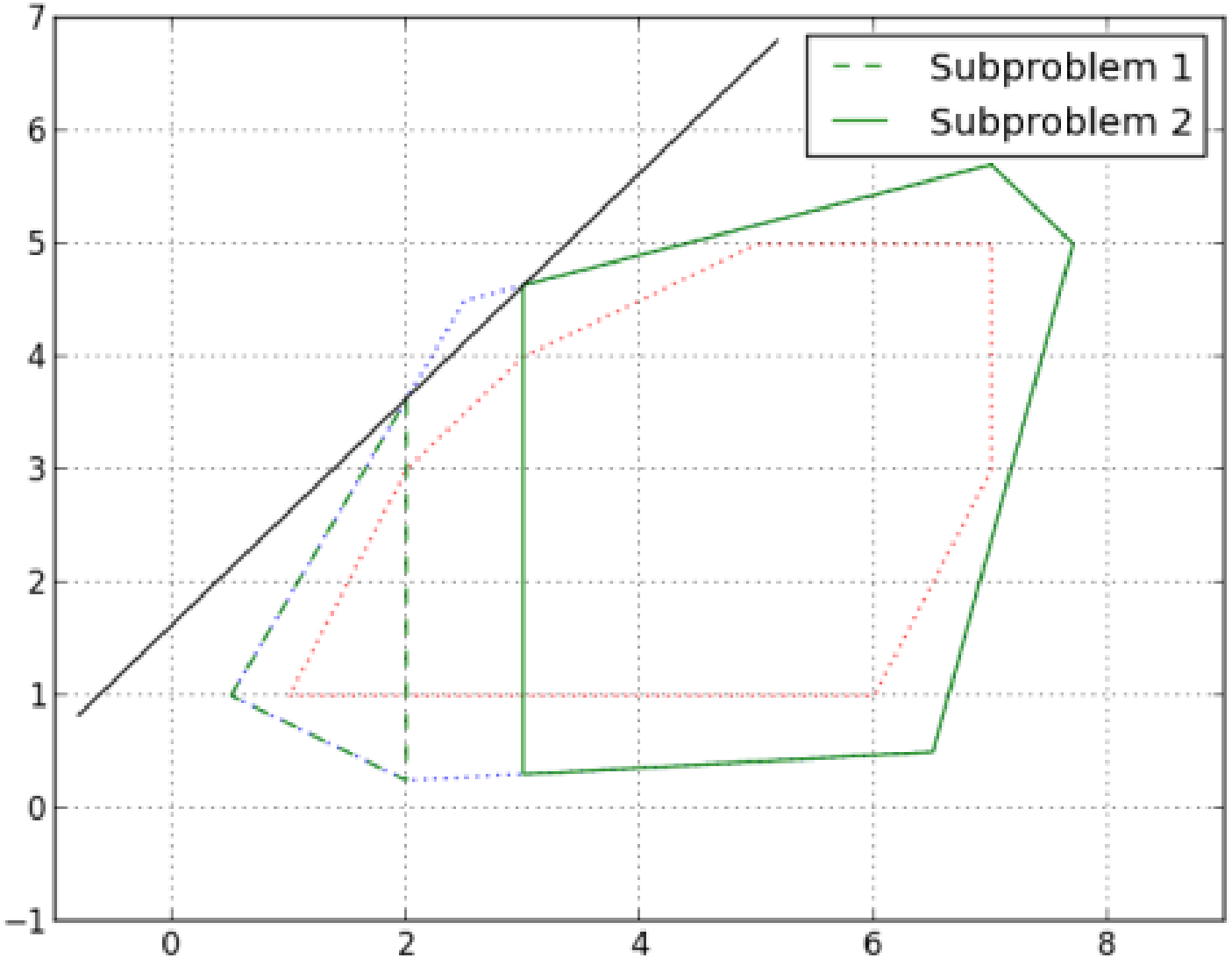
$$x_j = 0 \quad \text{OR} \quad x_j = 1$$

- In general IP, branching on a variable involves imposing new bound constraints in each one of the subproblems.
- This is is the most common method of branching and is easily handled implicitly in most cases.
- What are the benefits of such a scheme?

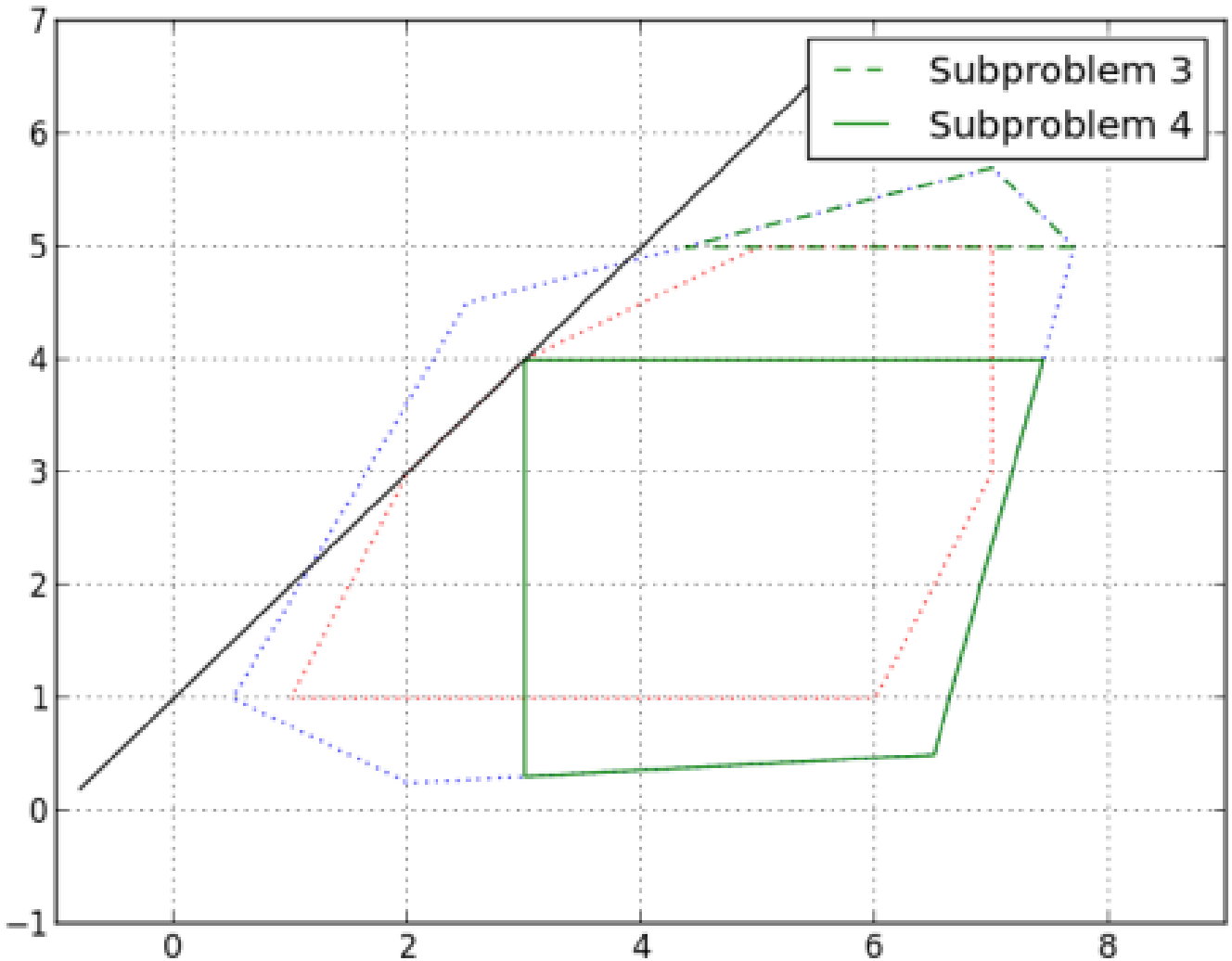
The Geometry of Branching



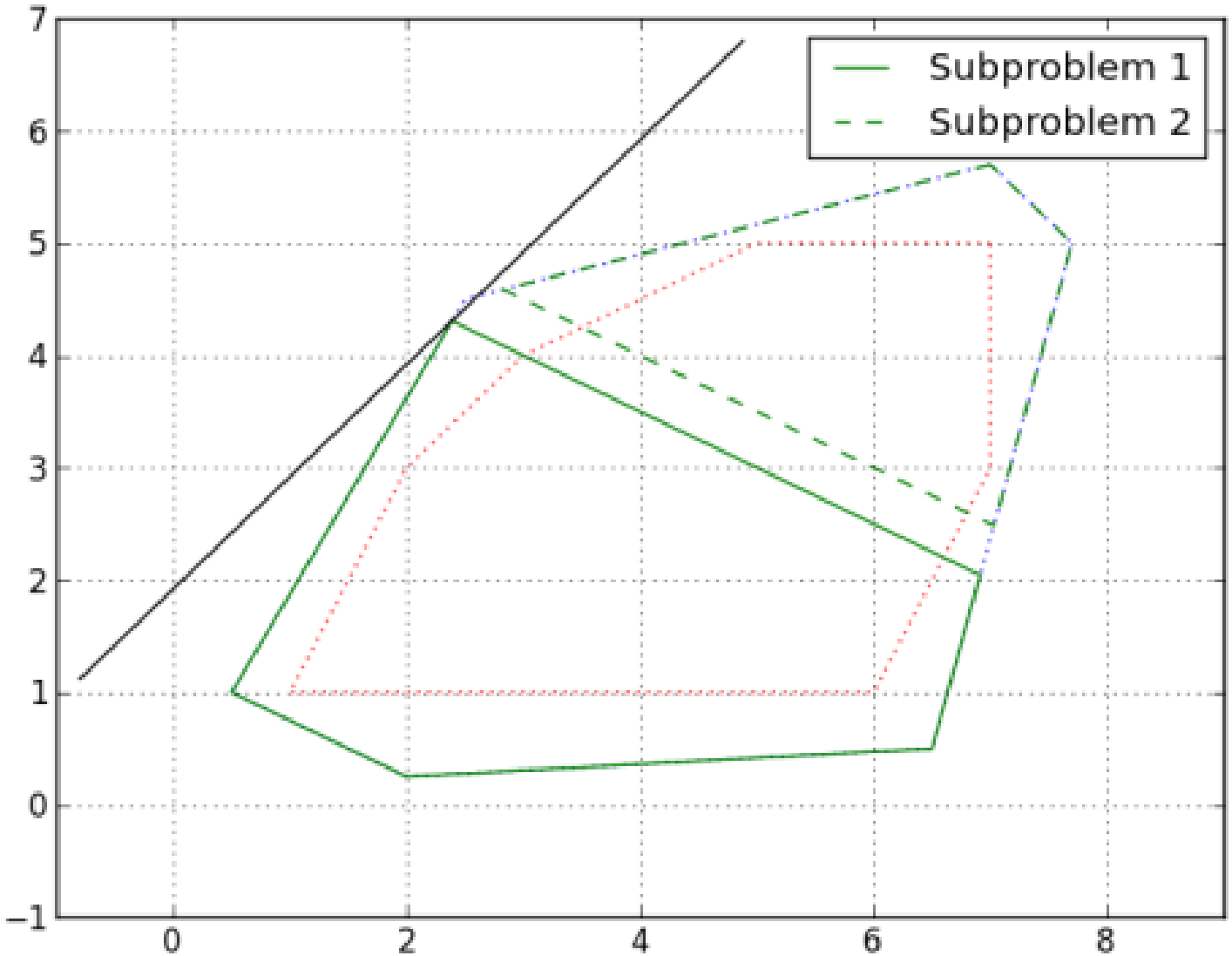
The Geometry of Branching (Variable Disjunction)



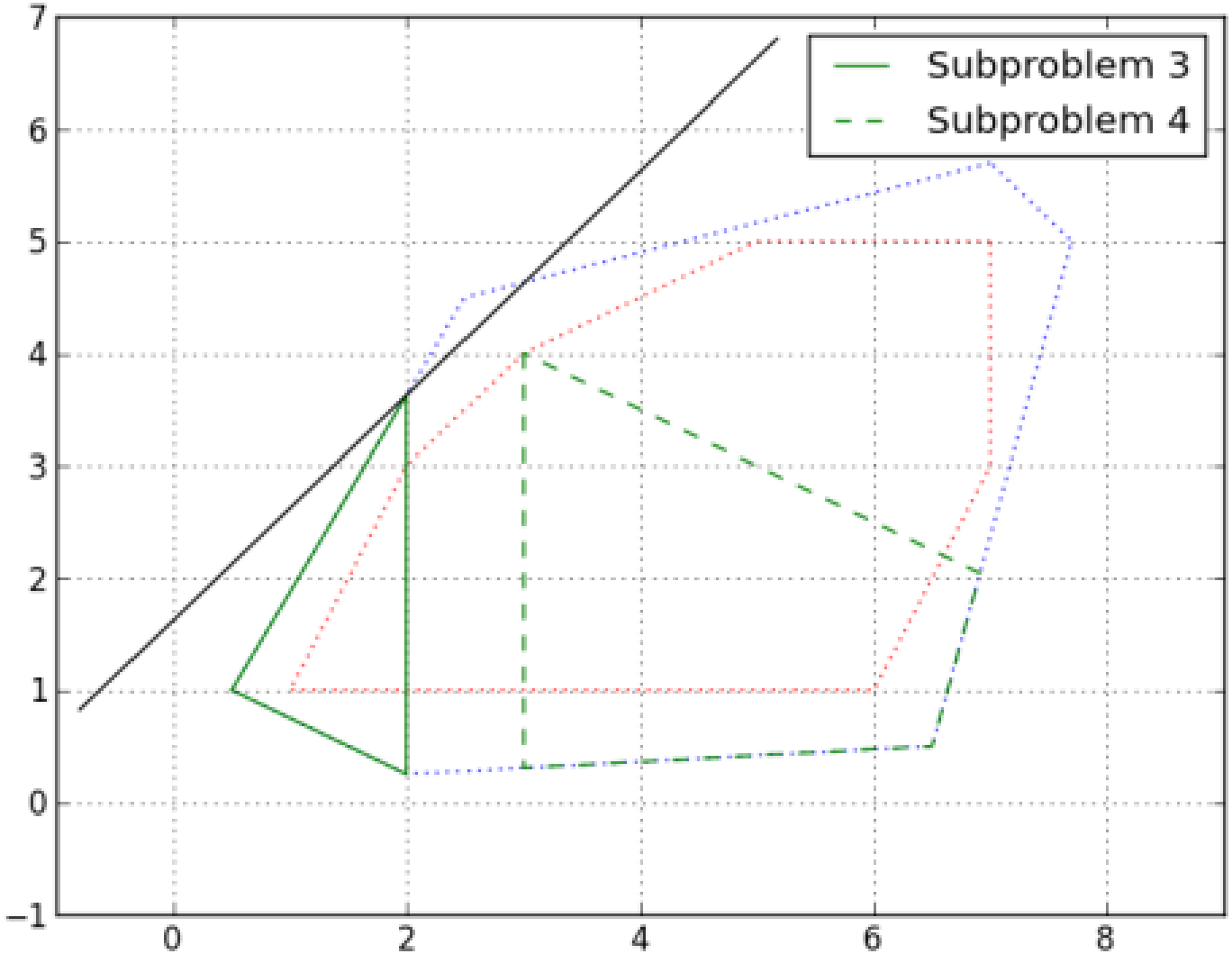
The Geometry of Branching (Variable Disjunction)



The Geometry of Branching (General Split Disjunction)



The Geometry of Branching (General Split Disjunction)



Other Disjunctions

- A generalized upper bound (GUB) is of the form:

$$\sum_{j \in Q} x_j = 1, \quad x \in \{0,1\}^Q$$

- Suppose $|Q| = 10$ and we branch on the disjunction $x_1 \leq 0$ OR $x_1 \geq 1$.
- How many possible solutions to the above equation are there in each of the branches? Is this a "good" disjunction to branch on?
- Consider the disjunction $\sum_{j=1}^{5} x_j = 0$ OR $\sum_{j=6}^{10} x_j = 0$.
- Is this better?

Logical Disjunctions

- We can derive other types of branching based on logical considerations.
- Example #1:
  - y_i binary variable and $y_i = 0 \Rightarrow \pi^\top x \leq \pi_0$.
  - Possible branching:

$$y_i = 1,$$
$$y_i = 0 \text{ and } \pi^\top x \leq \pi_0.$$

  - This avoids using the big M method.
- Example #2: Solving the TSP with Lagrangian relaxation.

Choosing a Branching Disjunction

- What is the real goal of branching?
- This may depend on the goal of the search
  - Find the best feasible solution possible in a limited time.
  - Find the provably optimal solution as quickly as possible.
- It is difficult to know how our branching decision will impact these goals, but we may want to choose a branching that
  - Decreases the upper bound,
  - Increases the lower bound, or
  - Result in one or more (nearly) infeasible subproblem.
- Most of the times, we focus on decreasing the upper bound.

Choosing a Branching Disjunction (cont'd)

- There are many possible disjunctions to choose from.
- We generally choose the branching disjunction based on the predicted amount of progress it will produce towards our goal.
- If the goal is to minimize time to optimality, bound improvement is often used as a proxy.
- How do we efficiently predict the bound improvement that will result from the imposition of a given disjunction?

Strong Branching

- Strong branching provides the most accurate estimate, but is computationally very expensive.
- The idea is to compute the actual change in bound by solving the bounding problems resulting from imposing the disjunction.
- This can be very costly. How can we moderate this?
  - Do only a limited number of dual-simplex pivots for each candidate for each child.
  - Use this as an estimate.
- Empirically, this reduces number of nodes, but this must be traded against the computational expense.

Pseudocost Branching

- An alternative to strong branching is pseudocost branching
- This is suitable primarily for branching on branching on variables.
- The pseudocost of a variable is an estimate derived by averaging observed changes resulting from branching on each variable.
- For each variable, we maintain an "up pseudocost"(P_j^+) and a "down pseudocost" (P_j^-).
- Then the change in bound for each child can be estimated as:

$$D_j^+ = P_j^+(1 - f_j)$$
$$D_j^- = P_j^- f_j,$$

  where $f_j = x_j^* - \lfloor x_j^* \rfloor$.
- In other words, D_j^+ and D_j^- are estimates of the change in bound that will result from imposing $x_j \geq \lfloor x_j^* \rfloor$ and $x_j \geq \lceil x_j^* \rceil$, respectively.

Pseudocost Initialization

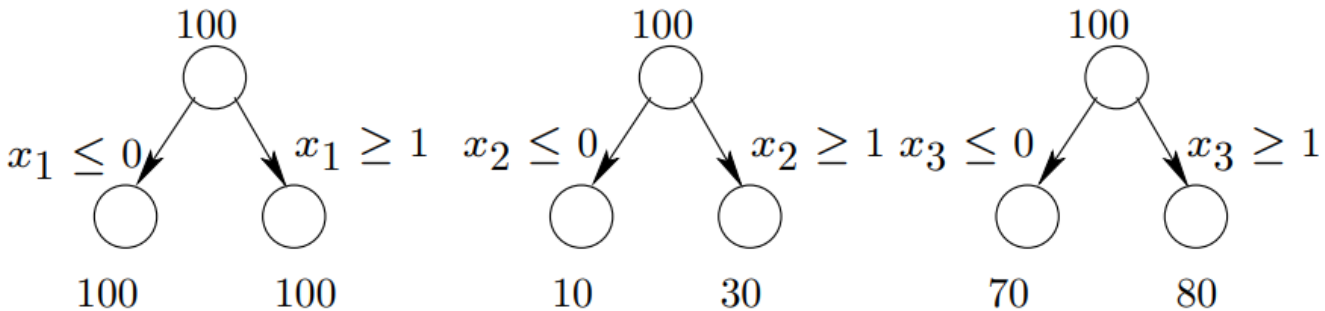
- Is it reasonable to assume that effect of branching on a particular variable is actually roughly the same in different parts of the tree?
- Empirical evidence shows that this is the case.
- Another important question is how to get initial estimates before any branching has occurred.
- This can be done initially using strong branching.
- After initialization, we switch to pseudocost branching, updating the pseudocost estimates after each bounding operation.
- A more systematic approach to doing this is to use what is called reliability branching.

Reliability Branching

- Strong branching is effective in reducing the number of nodes, but can be costly.
- Using pseudocosts is inexpensive, but requires good initialization.
- Reliability branching combines both.
  - Use strong branching in the early stages of the tree. Initialize/update pseudo-costs of variables using these bounds.
  - Once strong branching (or actual branching) has been carried out η number of times on a variable, only use pseudo-costs after that.
  - η is called reliability parameter.
  - What does $\eta = 0$ imply? What does $\eta = \infty$ imply?
  - Empirically $\eta = 4$ seems to be quite effective.

Comparing Branching Candidates

- So far we have seen, how to evaluate a candidate in several ways.
- Sometimes the choice of candidate is clear after this evaluation.



- Are we minimizing or maximizing?
- Which candidate would you choose?

Local Branching

- Local branching is a branching scheme that emphasizes finding feasible solutions over improving the upper bound.
- Consider the solution x^* to an LP relaxation at a certain node in the tree of a binary program.
- Let $\mathcal{S}$ be the set: $\{j \mid x_j^* = 0\}$.
- Consider the following disjunction for small k.

$$\sum_{j \in \mathcal{S}} x_j \leq k \text{ OR } \sum_{j \in \mathcal{S}} x_j \geq k + 1$$

- Is this a valid rule? Which child is easier to solve?
- For full details, see Local Branching by Fischetti and Lodi.
- We will discuss this and other methods when we talk about primal heuristics.

Valid Inequalities by Branching

- Note this one of the subproblems obtained by imposing a given binary disjunction is infeasible, the we obtain a valid inequality!
- This is in some sense what a valid inequality is.
- For the problem in Figure 1, branching on the valid disjunction $x_2 - x_1 \leq 1$ OR $x_2 - x_1 \geq 2$ immediately solves the problem.
- This may make it seem easy to find valid inequalities, but we will see later why this is not the case

Outline

Describing $\text{conv}(\mathcal{S})$

- We have seen that, in theory, $\text{conv}(\mathcal{S})$ is a polyhedron and has a finite description.
- If we "simply" construct that description, we could turn our MILP into an LP.
- So why aren't IPs easy to solve?
  - The size of the description is generally HUGE!
  - The number of facets of the TSP polytope for an instance with 120 nodes is more than 10^{100} times the number of atoms in the universe.
  - It is physically impossible to write down a description of this polytope.
  - Not only that, but it is very difficult in general to generate these facets (this problem is not polynomially solvable in general).

Basic Bounding Methods

- Our discussions of branch and bound has so far focused on the use of three basic bounding methods.
  - LP relaxation
  - Lagrangian relaxation
  - Combinatorial relaxation
- Branch and bound is fundamentally based on the dynamic generation and imposition of valid disjunctions.
- We will now show how disjunctions can also be exploited to generate inequalities valid for $\text{conv}(\mathcal{S})$.

Cutting Planes

- Recall that the inequality denoted by (π, π_0) is valid for a polyhedron $\mathcal{P}$ if $\pi x \leq \pi_0, \forall x \in \mathcal{P}$.
- The term cutting plane usually refers to an inequality valid for $\text{conv}(\mathcal{S})$, but which is violated by the solution obtained by solving the (current) LP relaxation.
- Cutting plane methods attempt to improve the bound produced by the LP relaxation by iteratively adding cutting planes to the initial LP relaxation.
- Adding such inequalities to the LP relaxation may improve the bound (this is not a guarantee).
- Note that when π and π_0 are integer, then π, π_0 is a split disjunction for which $X_2 = \emptyset$.

The Separation Problem

- Formally, the problem of generating a cutting plane can be stated as follows.
  Separation Problem: Given a polyhedron $\mathcal{Q} \in \mathbb{R}^n$ and $x^* \in \mathbb{R}^n$ determine whether $x^* \in \mathcal{Q}$ and if not, determine (π, π_0), a valid inequality for $\mathcal{Q}$ such that $\pi x^* > \pi_0$.
- This problem is stated here independent of any solution algorithm.
- However, it is typically used as a subroutine inside an iterative method for improving the LP relaxation.
- In such a case, x^* is the solution to the LP relaxation (of the current formulation, including previously generated cuts).
- We will see later that the difficulty of solving this problem exactly is strongly tied to the difficulty of the optimization problem itself.

Generic Cutting Plane Method

Let $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$ be the initial formulation for

$$\max\{c^\top x \mid x \in \mathcal{S}\}, \quad \mathcal{S} = \mathcal{P} \cap \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+.$$

Algorithm 1: Cutting plane method

1 $\mathcal{P}_0 \leftarrow \mathcal{P}$, $k \leftarrow 0$.
2 **while** *TRUE* **do**
3 Solve the LP relaxation $\max\{c^\top x | x \in \mathcal{P}_k\}$ to obtain solution x^k.
4 Solve the problem of separating x^k from $\text{conv}(\mathcal{S})$.
5 **if** $x^k \in \text{conv}(\mathcal{S})$ **then** STOP;
6 **else** Get an inequality (π^k, π^k_0) valid for $\text{conv}(\mathcal{S})$ but $(\pi^k)^\top x^k > \pi^k_0$;
7 $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cap \{x \in \mathbb{R}^n \mid (\pi^k)^\top x \leq \pi^k_0\}$.
8 $k \leftarrow k + 1$.

Question to be Answered

- How do we solve the separation problem?
- Will this algorithm terminate?
- If it does terminate, are we guaranteed to obtain an optimal solution?

Methods for Generating Cutting Planes

- Methods for generating cutting planes attempt to solve separation problem.
- In most cases, the separation problems that arises cannot be solved exactly, so we either
  - solve the separation problem heuristically, or
  - solve the separation problem exactly, but for a relaxation.
- The template paradigm for separation consists of restricting the class of inequalities considered to just those with a specific form.
- This is equivalent, in some sense, to solving the separation problem for a relaxation.
- Separation algorithm can be generally divided into two classes
  - Algorithms that do not assume any specific structure.
  - Algorithms that only work in the presence of specific structure.

Generating Cutting Planes: Two Basic Viewpoints

- There are a number of different points of view from which one can derive the standard methods used to generate cutting planes for general MILPs.
- As we have seen before, there is an algebraic point of view and a geometric point of view.
- Algebraic:
  - Take combinations of the known valid inequalities.
  - Use rounding to produce stronger ones.
- Geometric:
  - Use a disjunction (as in branching) to generate several disjoint polyhedra whose union contains $\mathcal{S}$.
  - Generate inequalities valid for the convex hull of this union.
- Although these seem like very different approaches, they turn out to be very closely related.

Generating Valid Inequalities: Algebraic Viewpoint

- Consider the polyhedron $\mathcal{P} = \{x \in \mathbb{R}^n \mid Ax \leq b\}$.
- Valid inequalities for $\mathcal{P}$ can be obtained by taking nonnegative linear combinations of the rows of (A, b).
- Except for one pathological case, all valid inequalities for $\mathcal{P}$ are either equivalent to or dominated by an inequality of the form

$$uAx \leq ub, \quad u \in \mathbb{R}^m_+.$$

- We are simply taking combinations of inequalities existing in the description, so any such inequalities will be redundant for the LP relaxation.

Generating Valid Inequalities for $\text{conv}(\mathcal{S})$

- All inequalities valid for P are also valid for $\text{conv}(\mathcal{S})$, but they are not cutting planes.
- We can do better.
- We need the following simple principle: if $a \leq b$ and a is an integer, then $a \leq \lfloor b \rfloor$.
- Believe it or not, this simple fact is all we need to generate all valid inequalities for $\text{conv}(\mathcal{S})$!

Chvátal Inequalities

- Suppose we can find a $u \in \mathbb{R}^m_+$ such that $\pi = uA$ is integer and $\pi_0 = ub \notin \mathbb{Z}$.
- In this case, we have $\pi^\top x \in \mathbb{Z}$ for all $x \in \mathcal{S}$, and so $\pi^\top x \leq \lfloor \pi_0 \rfloor$ for all $x \in \mathcal{S}$.
- In other words, $(\pi, \lfloor \pi_0 \rfloor)$ is both a valid inequality and a split disjunction.

Chvátal-Gomory Inequalities

- If we allow the non-negativity constraints to be combined with the constraints of $\mathcal{P}$ (with weight vector v), then integrality of π requires

$$uA_I - v_I \in \mathbb{Z}^p$$
$$uA_C - v_C = 0$$

So

$$v_i \geq \alpha_i - \lfloor \alpha_i \rfloor, \qquad \text{for } 0 \leq i \leq p,$$
$$v_I = \pi_i \geq 0, \qquad \text{for } p+1 \leq i \leq n.$$

- We then obtain that

$$\sum_{0 \leq i \leq p} \lfloor uA_i \rfloor x_i \leq \lfloor ub \rfloor$$

is valid for all $u \in \mathbb{R}^m_+$ such that $uA_C \geq 0$

- This is the Chvátal-Gomory Inequality

The Chvátal-Gomory Procedure

1. Choose a weight vector $u \in \mathbb{R}^m_+$ such that $uA_C \geq 0$.
2. Obtain the valid inequality $\sum_{0 \leq i \leq p}(uA_i)x_i \leq ub$.
3. Round the coefficients down to obtain $\sum_{0 \leq i \leq p}\lfloor(uA_i)\rfloor x_i \leq ub$. Why can we do this?
4. Finally, round the right hand side down to obtain the valid inequality

$$\sum_{0 \leq i \leq p} \lfloor(uA_i)\rfloor x_i \leq \lfloor ub \rfloor$$

- This procedure is called the Chvátal-Gomory rounding procedure, or simply the C-G procedure.
- Surprisingly, for pure ILPs $(p = n)$, any inequality valid for $\text{conv}(\mathcal{S})$ can be produced by a finite number of iterations of this procedure!
- This is not true for the general mixed case.

Assessing the Procedure

- Although it is theoretically possible to generate any valid inequality using the C-G procedure, this is not true in practice.
- The two biggest challenges are numerical errors and slow convergence.
- The inequalities produced may be very weak we may not even obtain a supporting hyperplane.
- This is is because the rounding only "pushes" the inequality until it meets some point in $\mathbb{Z}^n$, which may or may not even be in $\mathcal{S}$.
- The coefficients of the generated inequality must be relatively prime to ensure the generated hyperplane even includes an integer point!

Proposition. Let $S = \{x \in \mathbb{Z}^n | \sum_{j \in N} a_j x_j \leq b\}$ where $a_j \in \mathbb{Z}$ for $j \in N$, and let $k = gcd\{a_1, \cdots, a_n\}$. Then

$$\text{conv}(\mathcal{S}) = \{x \in \mathbb{R}^n | \sum_{j \in N} (a_j/k) x_j \leq \lfloor b/k \rfloor\}$$

Gomory Inequalities

- Let s consider T, the set of solutions to a pure ILP with one equation:

$$T = \left\{ x \in \mathbb{Z}_+^n | \sum_{j=1}^{n} a_j x_j = a_0 \right\}$$

- For each j, let $f_j = a_j - \lfloor aj \rfloor$. Then equivalently

$$T = \left\{ x \in \mathbb{Z}_+^n | \sum_{j=1}^{n} f_j x_j = f_0 + k \text{ for some integer } k \right\}$$

- Since $\sum_{j=1}^{n} f_j x_j \geq 0$ and $f_0 < 1$, then $k \geq 0$ and so

$$\sum_{j=1}^{n} f_j x_j \geq f_0$$

  is a valid inequality for $\mathcal{S}$ called a Gomory inequality.
- Note that this has been derived as a split disjunction with $X_2 = \emptyset$.

Gomory Cuts from the Tableau

- Gomory cutting planes can also be derived directly from the tableau while solving an LP relaxation.
- Consider the set

$$\{(x, s) \in \mathbb{Z}_+^{n+m} | Ax + Is = b\}$$

  in which the LP relaxation of an ILP is put in standard form.
- We assume for now that A has integral coefficients so that the slack variables also have integer values implicitly.
- The tableau corresponding to basis matrix B is

$$B^{-1}Ax + B^{-1}s = B^{-1}b$$

- Each row of this tableau corresponds to a weighted combination of the original constraints.
- The weight vectors are the rows of B^{-1}.

Gomory Cuts from the Tableau (cont.)

- A row of the tableau is obtained by combining the equations in the standard representation with weight vector $\lambda = B_j^{-1}$ to obtain

$$\sum_{j=1}^{n}(\lambda A_j)x_j + \sum_{i=1}^{m}\lambda_i s_i = \lambda b,$$

  where A_j is the j^{th} column of A and λ is a row of B^{-1}.

- Applying the previous procedure, we can obtain the valid inequality

$$\sum_{j=1}^{n}(\lambda A_j - \lfloor \lambda A_j \rfloor)x_j + \sum_{i=1}^{m}(\lambda_i - \lfloor \lambda_i \rfloor)s_i \geq \lambda b - \lfloor \lambda b \rfloor,$$

- We will show that this Gomory cut is equivalent to the C-G inequality with weights $u_i = \lambda_i - \lfloor \lambda_i \rfloor$.

Gomory Versus C-G

- To show the Gomory cut is a C-G cut, we first apply the C-G procedure directly to the tableau row, resulting in the inequality

$$\sum_{j=1}^{n} \lfloor \lambda A_j \rfloor x_j + \sum_{i=1}^{m} \lfloor \lambda_i \rfloor s_i = \lfloor \lambda b \rfloor,$$

- Note that we can also obtain this inequality by adding the Gomory cut and the original tableau row.
- Now, we substitute out the slack variables using the equation

$$s = b - Ax$$

to obtain

$$\sum_{j=1}^{n} \left(\lfloor \lambda A_j \rfloor - \sum_{i=1}^{m} \lfloor \lambda_i \rfloor a_{ij} \right) x_j \leq \lfloor \lambda b \rfloor - \sum_{i=1}^{m} \lfloor \lambda_i \rfloor b_i$$

Gomory Versus C-G (cont.)

- The final inequality from the previous slide can be re-written as

$$\sum_{j=1}^{n}\left(\sum_{i=1}^{m}(\lambda_i - \lfloor\lambda_i\rfloor)a_{ij}\right)x_j \leq \sum_{i=1}^{m}(\lambda_i - \lfloor\lambda_i\rfloor)b_i$$

  which is a C-G inequality.

- The substitution of slack variables is more than just a textbook procedure to show the Gomory cut is a C-G cut.

- In practice, the slack variables are substituted out in this fashion in order to derive a cut in terms of the original variables.

Strength of Gomory Cuts from the Tableau

- Consider a row of the tableau in which the value of the basic variable is not an integer.
- Applying the procedure from the last slide, the resulting inequality will only involve nonbasic variables and will be of the form

$$\sum_{j \in NB} f_j x_j \geq f_0$$

  where $0 \leq f_j < 1$ and $0 < f_0 < 1$
- The left-hand side of this cut has value zero with respect to the solution to the current LP relaxation.
- We can conclude that the generated inequality will be violated by the current solution to the LP relaxation.

A Finite Cutting Plane Procedure

- Under mild assumptions on the algorithm used to solve the LP, this yields a general algorithm for solving (pure) ILPs.

Generating All Valid Inequalities

- Any valid inequality that can be obtained through iterative application of the C-G procedure (or is dominated by such an inequality) is a C-G inequality.
- For pure integer ILPs, all valid inequalities are C-G inequalities. **Theorem 1.** Let $(\pi, \pi_0) \in \mathbb{Z}^{n+1}$ be a valid inequality for $\mathcal{S} = \{x \in \mathbb{Z}^n_+ \mid Ax \leq b\} \neq \emptyset$. Then (π, π_0) is a C-G inequality for $\mathcal{S}$.
- The C-G rank denoted $r(\pi, \pi_0)$ of an inequality (π, π_0) valid for $\mathcal{P}$ is defined recursively as follows.
  - All inequalities valid for the elementary closure $\mathcal{P}^1 = e(\mathcal{P})$ are rank 1.
  - The polyhedron $\mathcal{P}^2 = e(\mathcal{P}^1)$ is the rank 2 closure-inequalities valid for it that are not rank 1 are rank 2 inequalities.
  - An inequality is rank k if it is valid for the rank k closure $\mathcal{P}^k = e(\mathcal{P}^{k-1})$ and not for $\mathcal{P}^{k-1}$.
- The C-G rank of $\mathcal{P}$ is the maximum rank of any facet-defining inequality of $\text{conv}(\mathcal{S})$.

Valid Inequalities from Disjunctions

- Valid inequalities for $\text{conv}(\mathcal{S})$ can also be generated based on disjunctions.
- Let $\mathcal{P}_i = \{x \in \mathbb{R}^n_+ \mid A^i x \leq b^i\}$ be such that $\mathcal{S} \subseteq \cup_{i=1}^n \mathcal{P}_i$. Then inequalities valid for $\cup_{i=1}^n \mathcal{P}_i$ are also valid for $\text{conv}(\mathcal{S})$.
- The following procedure generates such inequalities.
  Proposition 2. If (π^1, π^1_0) is valid for $\mathcal{S}_1 \subseteq \mathbb{R}^n_+$ and (π^2, π^2_0) is valid for $\mathcal{S}_1 \subseteq \mathbb{R}^n_+$, then

$$\sum_{j=1}^{n} \min(\pi^1_j, \pi^2_j) x_j \leq \max(\pi^1_0, \pi^2_0), \quad x \in \mathcal{S}_1 \cup \mathcal{S}_2$$

- In fact, all valid inequalities for the union of two polyhedra can be obtained in this way.
  Proposition 3. If $\mathcal{P}_i = \{x \in \mathbb{R}^n_+ \mid A^i x \leq b^i\}$ for $i = 1, 2$ are nonempty polyhedra, then (π, π_0) is a valid inequality for $\text{conv}(\mathcal{P}_1 \cup \mathcal{P}_2$ if and only if there exist $u^1, u^2 \in \mathbb{R}^m$ such that $\pi \leq u^i A^i$ and $\pi_0 \geq u^i b^i$ for $i = 1, 2$.

Gomory Mixed Integer Inequalities

- Let's consider again the set of solutions T to an IP with one equation.
- This time, we write T equivalently as

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

- Since $k \le -1$ or $k \ge 0$, we have these two disjunctions

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

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

The Gomory Mixed Integer Cut

- Applying Proposition 2, 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 and can also be obtained easily from the tableau.
- In the case of the mixed integer set

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

the GMI cut is

$$\sum_{\substack{0 \le j \le p \\ f_j \le f_0}} \frac{f_j}{f_0} x_j + \sum_{\substack{0 \le j \le p \\ f_j > f_0}} \frac{1-f_j}{1-f_0} x_j + \sum_{\substack{p+1 \le j \le n \\ a_j > 0}} \frac{a_j}{f_0} x_j - \sum_{\substack{p+1 \le j \le n \\ a_j < 0}} \frac{a_j}{1-f_0} x_j \ge 1$$

Gomory Mixed Integer Cuts from the Tableau

- Let's consider how to generate mixed integer Gomory cuts from the tableau when solving an MILP of the form

$$\mathcal{Q} = \{x \in \mathbb{Z}^p_+ \times \mathbb{R}^{n-p}_+ \mid Ax \leq b\}.$$

- We first introduce a slack variable for each inequality in the formulation.
- Solving the LP relaxation, we look for a row in the tableau in which an integer variable is basic and has a fractional variable.
- We apply the GMI procedure to produce a cut.
- Finally, we substitute out the slack variables in order to express the cut in terms of the original variables only.

Geometric Interpretation of GMI Cuts

- To understand the geometric interpretation of GMI cuts, we consider a relaxation of associated with a basis of the LP relaxation.
- We simply relax the non-negativity constraints on the basic variables to obtain

$$T = \{(x, s) \in \mathbb{Z}^{n+m} \mid Ax + Is = b, x_N \geq 0, s_N \geq 0\},$$

  where x_N and s_N are the non-basic variables associated with basis B.
- This is equivalent to relaxing the non-binding constraints.
- The convex hull of T is the so-called corner polyhedron associated with the basis B.

Example: Corner Polyhedron

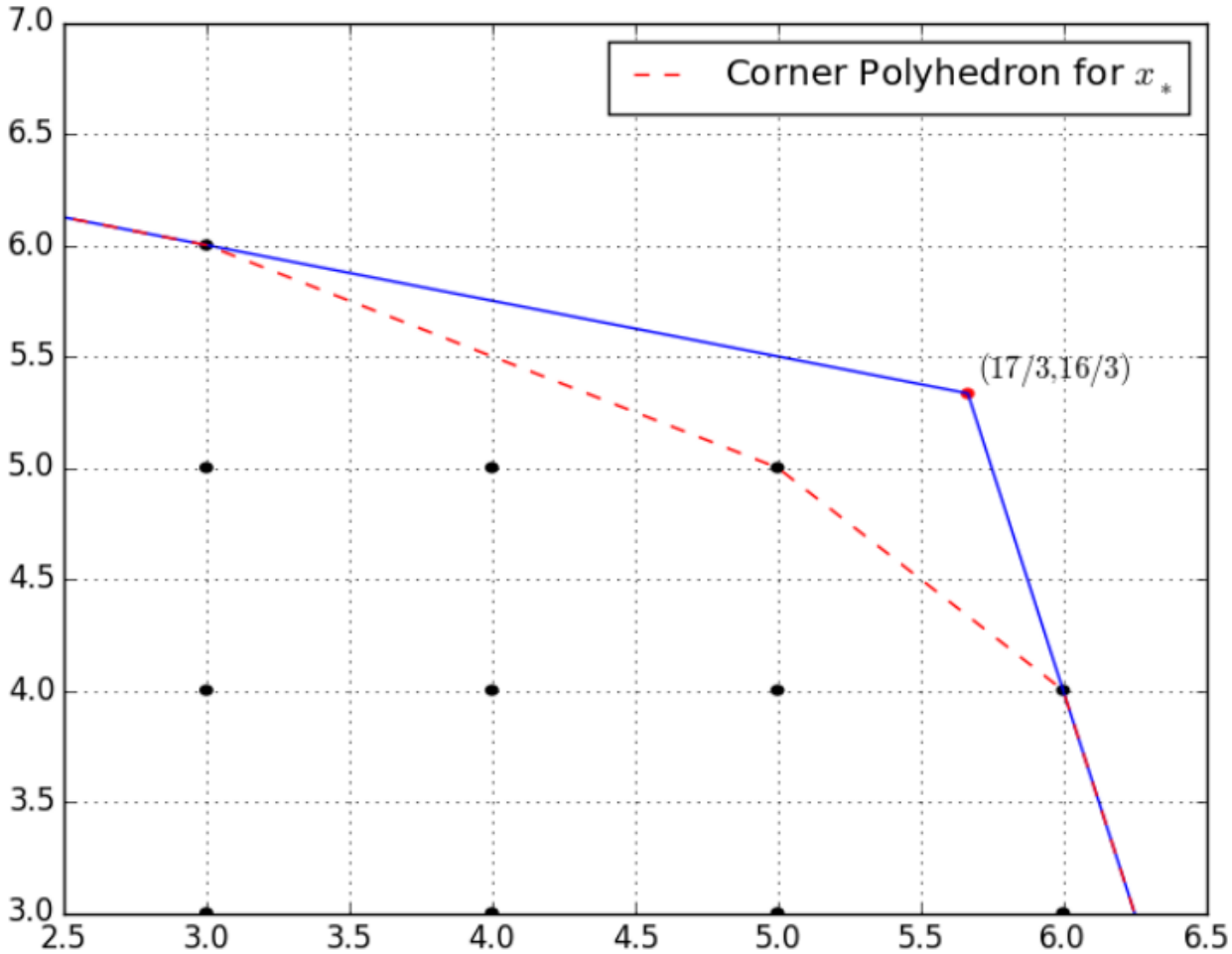


Figure: The corner polyhedron associated with the optimal basis of the LP relaxation of the earlier example.

Split Inequalities

- Let (α, β) be a split disjunction and define

$$\mathcal{P}_1 = \mathcal{P} \cap \{x \in \mathbb{R}^n \mid \alpha^\top x \leq \beta\}$$
$$\mathcal{P}_2 = \mathcal{P} \cap \{x \in \mathbb{R}^n \mid \alpha^\top x \geq \beta + 1\}$$

- Any inequality valid for $\text{conv}(\mathcal{P}_1 \cup \mathcal{P}_2)$ is valid for $\mathcal{S}$ and is called a split inequality.

Separation Problem for Split Inequalities

- The LP can be generalized straightforwardly to produce the most violated split cut

$$\begin{aligned} \max \quad & \pi\hat{x} - \pi^0 \\ \text{s.t.} \quad & \pi \leq uA + u_0\alpha, \\ & \pi \leq vA - v_0\alpha, \\ & \pi^0 \geq ub + u_0\beta, \\ & \pi^0 \geq vb - v_0(\beta + 1), \end{aligned}$$

$$\sum_{i=1}^{m} u_i + u_0 + \sum_{i=1}^{m} v_i + v_0 = 1,$$

$$u, u_0, v, v_0 \geq 0, \quad \alpha \in \mathbb{Z}^n, \beta \in \mathbb{Z}$$

- The separation problem is a mixed integer nonlinear optimization problem, however, and is not easy to solve.

Outline

Branch and Cut

- Branch and cut is an LP-based branch-and-bound scheme in which the linear programming relaxations are augmented by valid inequalities.
- The valid inequalities are generated dynamically using separation procedures.
- We iteratively try to improve the current bound by adding valid inequalities.
- In practice, branch and cut is the method typically used for solving difficult mixed-integer linear programs.
- It is a very complex amalgamation of techniques whose application must be balanced very carefully.

Computational Components of Branch and Cut

- Modular algorithmic components
  - Initial preprocessing and root node processing
  - Bounding
  - Cut generation
  - Primal heuristics
  - Node pre/post-processing (bound improvement, conflict analysis)
  - Node pre-bounding
- Overall algorithmic strategy
  - Search strategy
  - Bounding strategy
  - Branching strategy

Tradeoffs

- Control of branch and cut is about tradeoffs.
- We are combining many techniques and must adjust levels of effort of each to accomplish an end goal.
- Algorithmic control is an optimization problem in itself!
- Many algorithmic choices can be formally cast as optimization problems.
- What is the objective?
  - Time to optimality
  - Time to first "good" solution
  - Balance of both?

Preprocessing and Probing

- Often, it is possible to simplify a model using logical arguments.
- Most commercial IP solvers have a built-in preprocessor.
- Effective preprocessing can pay large dividends.
- Let the upper and lower bounds on x_j be u_j and l_j.
- The most basic type of preprocessing is calculating **implied bounds**.
- Let (π, π_0) be a valid inequality.
- If $\pi_1 > 0$, then

$$x_1 \leq (\pi_0 - \sum_{j:\pi_j>0} \pi_j l_j - \sum_{j:\pi_j<0} \pi_j u_j)/\pi_1$$

- If $\pi_1 < 0$, then

$$x_1 \geq (\pi_0 - \sum_{j:\pi_j>0} \pi_j l_j - \sum_{j:\pi_j<0} \pi_j u_j)/\pi_1$$

Basic Preprocessing

- Again, let (π, π_0) be any valid inequality for $\mathcal{S}$.
- The constraint $\pi x \leq \pi_0$ is redundant if

$$\sum_{j:\pi_j>0} \pi_j u_j + \sum_{j:\pi_j<0} \pi_j l_j \leq \pi_0$$

- $\mathcal{S}$ is empty (IP is infeasible) if

$$\sum_{j:\pi_j>0} \pi_j l_j + \sum_{j:\pi_j<0} \pi_j u_j > \pi_0$$

- For any IP of the form $\max\{cx \mid Ax \leq b, l \leq x \leq u\}$, $x \in \mathbb{Z}^n$
  - If $a_{ij} \geq 0, \forall i \leq m$ and $c_j < 0$, then $x_j = l_j$ in any optimal solution.
  - If $a_{ij} \leq 0, \forall i \leq m$ and $c_j > 0$, then $x_j = u_j$ in any optimal solution.

Probing for Integer Programs

- It is also possible in many cases to fix variables or generate new valid inequalities based on logical implications.
- Consider (π, π_0), a valid inequality for 0-1 integer program.
- If $\pi_k > 0$ and $\pi_k + \sum_{j:\pi_j<0} \pi_j > \pi_0$, then we can fix x_k to zero.
- Similarly, if $\pi_k < 0$ and $\sum_{j:\pi_j<0, j\neq k} \pi_j > \pi_0$, then we can fix x_k to one.
- Example: Generating logical inequalities

Improving Coefficients

- Suppose again that (π, π_0) is a valid inequality for a 0-1 integer program.
- Suppose that $\pi_k > 0$ and $\sum_{j:\pi_j>0, j\neq k} \pi_j < \pi_0$
- If $\pi_k > \pi_0 - \sum_{j:\pi_j>0, j\neq k} \pi_j$, then we can set
  - $\pi_k \leftarrow \pi_k - (\pi_0 - \sum_{j:\pi_j>0, j\neq k} \pi_j)$, and
  - $\pi_0 \leftarrow \sum_{j:\pi_j>0, j\neq k} \pi_j$
- Similarly, suppose that $\pi_k < 0$ and $\pi_k + \sum_{j:\pi_j>0, j\neq k} \pi_j < \pi_0$
- Then we can again set $\pi_k \leftarrow \pi_k - (\pi_0 - \pi_j - \sum_{j:\pi_j>0, j\neq k} \pi_j)$

Preprocessing and Probing in Branch and Bound

- In practice, these rules are applied iteratively until none applies.
- Applying one of the rules may cause a new rule to apply.
- Bound improvement by reduced cost can be reapplied whenever a new bound is computed.
- Furthermore, all rules can be reapplied after branching.
- These techniques can make a very big difference.

Root Node Processing

- Typically, more effort is put into processing the root node than other nodes in the tree.
- Work done in the root node will impact the processing of every subsequent node.
- Dual bounding
  - Cut generation in the root node can be thought of as an additional pre-processing step ro the formulation before enumeration.
  - Cut generation in the root node can also be used to predict effectiveness of such techniques elsewhere in the tree.
- Primal bounding
  - Primal bounds found in the root node can have a big impact on the search.
  - They help to improvement variable bounds by reduced cost and can also lead to more effective/efficient search strategies.
  - As with cut generation, we use performance in the root node as an indicator of efficacy throughout the tree.

Bound Improvement by Reduced Cost

- Consider an integer program $\max_{x \in \mathbb{Z}^n} \{cx | Axleb, 0 \leq x \leq u\}$.
- Suppose the linear programming relaxation has been solved to optimality and row zero of the tableau looks like

$$z = \bar{a}_{00} + \sum_{j \in NB_1} \bar{a}_{0j} x_j + \sum_{j \in NB_2} \bar{a}_{0j} (x_j - u_j)$$

  where NB_1 are the nonbasic variables at 0 and NB_2 are the nonbasic variables at their upper bounds u_j.
- In addition, suppose that a lower bound z on the optimal solution value for IP is known.
- Then in any optimal solution

$$x_j \leq \lfloor \frac{\bar{a}_{00} - z}{-\bar{a}_{0j}} \rfloor \text{ for } j \in NB_1, \text{and}$$

$$x_j \geq u_j - \lfloor \frac{\bar{a}_{00} - z}{\bar{a}_{0j}} \rfloor textforj \in NB_2$$

Other Techniques

- Bound improvement in the root node
  - Whenever a new lower bound is found by a heuristic or otherwise, we can apply bound improvement in the root node.
  - To do so, we save the reduced costs of the variables in the root node.
  - We can do this for multiple bases obtained during the processing of the root node.
  - The bound improvements found in this way can be immediately applied to all candidate and active nodes.
- Techniques similar to those applied in the root node can also be applied during the processing of individual nodes.
- New implications may be available once branching constraints are applied.

Conflict Analysis

- Whenever a node is found to be infeasible, we derive a conflict.
- The branching constraints imposed to arrive at that node cannot all be imposed simultaneously.
- These conflicts can be used to derive cuts and may also contribute to enhanement of the conflict graph.

Bounding

- For now, we focus on the use of cutting plane methods for bounding.
- We will discuss decomposition-based bounding in a later lecture.
- The bounding loop is essentially a cutting plen method for solving the subproblem, but with some kind of early termination criteria.
- After termination, branching is performed to continue the algorithm.
- The bounding loop consists of several steps applied iteratively (not necessarily in this order).
  - Solve the current LP relaxation.
  - Decide whether node can be fathomed (by infeasibility or bound).
  - Generate inequalities violated by the solution to the LP relaxation.
  - Perform primal heuristics.
  - Apply node pre/post-processing.
  - Manage/improve LP relaxation (add/remove cuts, change bounds)
  - Decide whether to branch

Solving the LP Relaxation

- The LP relaxation is typically solved using a simplex-based algorithm.
  - This yields the advantage of efficient warm-starting of the solution process.
  - Many standard cut generation techniques require a basic solution.
- Interior point methods may be useful in some cases where they are much more effective (set packing/partitioning is one case in which this is typical).
- It may also be fruitful in some cases to explore the use of alternatives, such as the Volume Algorith.

Cut Generation

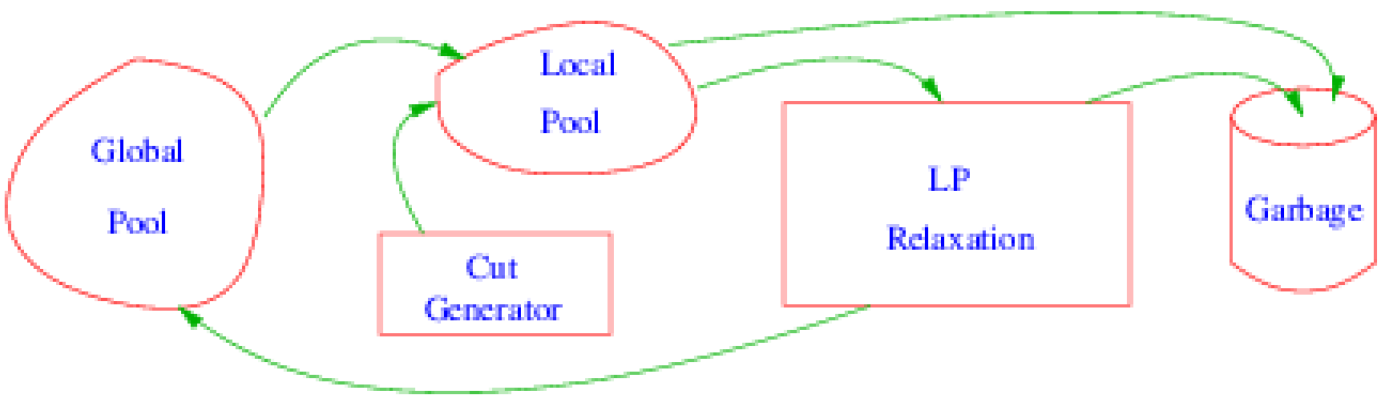
- Standard methods for generating cuts
  - Gomory, GMI, MIR, and other tableau-based disjunctive cuts.
  - Cuts from the node packing relaxation (clique, odd hole)
  - Knapsack cuts (cover cuts).
  - Single node flow cuts (flow cover).
  - Simple cuts from pre-processing (probing, etc).
- We must choose from among these various method which ones to apply in each node.
- We must in general decide on a general level of effort we want to put into cut generation.

Managing the LP Relaxations

- In practice, the number of inequalities generated can be HUGE.
- We must be careful to keep the size of the LP relaxations small or we will sacrifice efficiency.
- This is done in two ways:
  - Limiting the number of cuts that are added each iteration.
  - Systematically deleting cuts that have become ineffective.
- How do we decide which cuts to add?
- And what do we do with the rest?
- What is an ineffective cut?
  - One whose dual value is (near) zero.
  - One whose slack variable is basic.
  - One whose slack variable is positive.

Managing the LP Relaxations

- Below is a graphical representation of how the LP relaxation is managed in practice.
- Newly generated cuts enter a buffer (the local cut pool).
- Only a limited number of what are predicted to be the most effective cuts from the local are added in each iteration.
- Cuts that prove effective locally may eventually sent to a global pool for future use in processing other subproblems.



Cut Generation and Management

- A significant question in branch and cut is what classes of valid inequalities to generate and when?
- It is generally not a good idea to try all cut generation procedures on every fractional solution arising.
- For generic mixed-integer programs, cut generation is most important in the root node.
- Using cut generation only in the root node yields a procedure called cut and branch.
- Depending on the structure of the instance, different classes of valid inequalities may be effective.
- Sometimes, this can be predicted ahead of time (knapsack inequalities).
- In other cases, we have to use past history as a predictor of effectiveness.
- Generally, each procedure is only applied at a dynamically determined frequency.

Deciding Which Cuts to Add

- Predicting what cuts will be effective is difficult in general.
- Degree of violation is an easy-to-apply criteria, but may not be the most natural or intuitive measure.
- Other measures
  - Bound improvement (difficult to predict/calculate)
  - Euclidean distance from point to be cut off.
- It is possible to generate cuts using a different measure than that which is used to add them from the local pool.
- This might be done because generation by a criteria other than degree of violation is difficult.

Deciding When to Branch

- Because the cutting plane algorithm is a finite algorithm in itsef (at least in the pure integer case), there is no strict requirement to branch.
- The decision to branch is thus a practical matter.
- Typically, branching is undertaken when "tailing off" occurs, i.e., bound improvement slows.
- Detecting when this happens is not straightforward and there are many ways of doing it.
- Ultimately, branching and cutting (using the same disjunction) have the same impact on the bound.
- Tailing off may simply be a result of numerical issues
- We will consider the numerics of the solution process in a later lecture.

Balancing the Effort of Branching and Bounding

- To a large extent, the more effort one puts into branching, the smaller the search tree will be.
- Branching effort can be tuned by adjusting
  - How many branching candidates to consider.
  - How much efort should be put into estimating impact (pseduo-cost estimates versus strong branching, etc.).
  - The same can be said about efforts to improve both primal and dual bounds.
    - For dual bounds, we need to determine how much effort to spend generating various classes of inequalities.
    - For primal bounds, we need to determine how much effort to put into primal heuristics.
  - One of the keys to making the overall algorithm work well is to tune the amount of effort allocated to each of these techniques.
  - This is a very difficult thing to do and the proper balance is different for different classes of problems.

Primal Bounding Strategy

- The strategy space for primal heuristics is similar to that for cuts.
- We have a collection of different heuristics that can be applied.
- We need to determine which heuristics to apply and how often.
- Generally speaking, we do this dynamically based on the historical effectiveness of each method.

Computational Aspects of Search Strategy

- The search must find the proper balance between several factors.
  - Primal bound improvement versus dual bound improvement.
  - The savings accrued by diving versus the effectiveness of best first.
- When we are confident that the primal bound is near optimal, such as when the gap is small, a diving strategy is more appropriate.
- We can also adjust our strategy based on what the user's desire is.

Adjusting Strategy Based on User Desire

- In general, there is always a tradeoff between improvement of the dual and the primal bound.
- The user may have partcular desires about which of these is more important.
- Some solvers change their strategy according to the emphasis preferred by the user.
  - Proving optimality
  - Finding good solution quickly