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

Mixed Integer Linear Programming

- Consider linear programming 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 $A \in \mathbb{Q}^{m \times n}, b \in \mathbb{Q}^m, c \in \mathbb{Q}^n$.

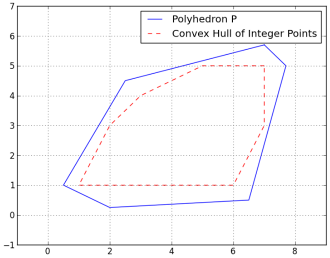
- This type of optimization problem is called a mixed integer linear programming (MILP) problem.
- If $p = n$, then we have a pure integer linear optimization problem.
- Special case: the integer variables are binary, i.e., 0 or 1.

The Geometry of Integer Programming

- Let's consider an integer linear program

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

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



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

How Hard is Integer Programming?

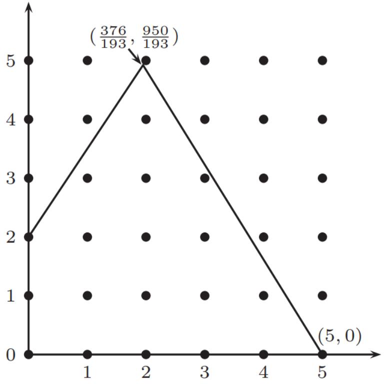
- Solving general integer programs can be much more difficult than solving linear programs.
- There in no known polynomial-time algorithm for solving general MIPs.
- Solving the associated linear programming relaxation results in an upper bound on the optimal solution to the MIP.
- In general, solving 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?

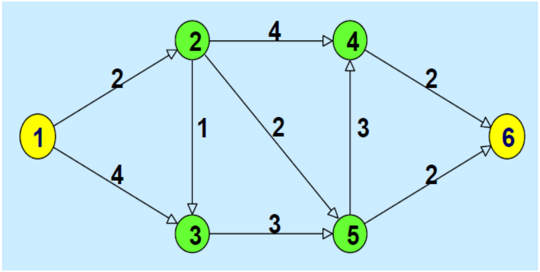
Consider the integer program

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

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

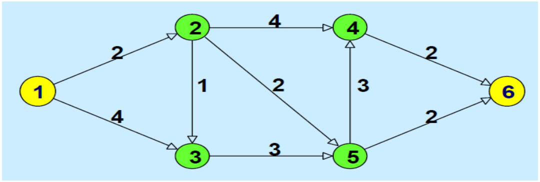


The shortest path problem



- 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



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

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

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

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

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

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

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

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

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"

MIP reformulation of ℓ_0-minimization

- Big-M assumption: $\forall i,\ |x_i| \leq M$

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad \|x\|_0 \leq k, |x_i| \leq M$$

- MIP formulation:

$$\min_{x\in\mathbb{R}^n} f(x) \quad \text{s.t.} \quad \sum_{i=1}^{n} y_i \leq k, |x_i| \leq My_i, y_i \in \{0,1\}$$

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:

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

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

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

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

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

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 $f_{ij} > 0 \Rightarrow x_{ij} = 1$.

$$
\begin{array}{lll}
\min & \displaystyle\sum_{(i,j)\in A} c_{ij}x_{ij} + d_{ij}f_{ij} & \\
\text{s.t.} & \displaystyle\sum_{j\in O(i)} f_{ij} - \sum_{j\in I(i)} f_{ji} = b_i, & \forall i \in N \\
& f_{ij} \leq Cx_{ij}, & \forall (i,j) \in A \\
& f_{ij} \geq 0, & \forall (i,j) \in A \\
& x_{ij} \in \{0,1\}, & \forall (i,j) \in A
\end{array}
$$

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:

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

- 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 $\mathrm{conv}(\mathcal{S}) \subseteq \mathcal{P}_A$ (why?).
- A is ideal if $\mathrm{conv}(\mathcal{S}) = \mathcal{P}_A$.
- If we know an ideal formulation, we can solve the IP (why?).
- How do our formulations of the knapsack problem compare by this measure?

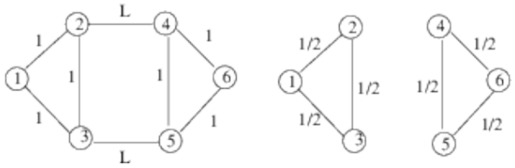
Strengthening Formulations

- Often, a given formulation can be strengthened with additional inequalities satisfied by all feasible integer solutions.

- Example: given a graph G = (V, E), a perfect matching in G is a subset M of edge set E, such that every vertex in V is adjacent to exactly one edge in M.
  - 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.
  - 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.

$$\begin{aligned} \min \quad & \sum_{e=\{i,j\}\in E} c_e x_e \\ \text{s.t.} \quad & \sum_{\{j|\{i,j\}\in E\}} x_{ij} = 1, & \forall i \in N \\ & x_e \in \{0,1\}, & \forall e = \{i,j\} \in E \end{aligned}$$

Valid Inequalities for Matching



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

The Odd Set Inequalities

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

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

- An odd cutset is any $\delta(S)$ for which 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

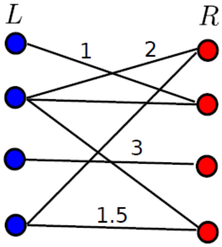
Contrast with Linear Programming

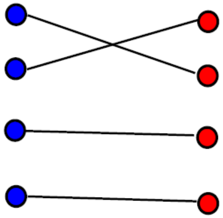
- In linear programming, the same problem can also have multiple formulations.
- In LP, however, conventional wisdom is that bigger formulations take longer to solve.
- In IP, this conventional wisdom does not hold.
- We have already seen two examples where it is not valid.
- Generally speaking, the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

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

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

The Max-Weight Bipartite Matching

Integer program (IP)

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

LP relaxation

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

- 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

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

Dual

$$\begin{aligned} \min \quad & \sum_{i} y_i \\ \text{s.t.} \quad & y_i + y_j \ge w_{ij}, \forall (i,j) \in A \\ & y \ge 0 \end{aligned}$$

- 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 $P^*_{LP} = D^*_{LP}$

Primal-Dual Interpretation

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

Dual

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

Dual Integer Program

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

- 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

- 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

- 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

- We have two conceptual reformulations of a given integer optimization problem.
- The first is in terms of disjunction:

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

- The second is in terms of valid inequalities

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

  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 $\mathrm{conv}(\mathcal{S})$ (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.

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., $\mathcal{S}_i = \mathcal{S} \cap X_i$.
- 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 $x^* \in \operatorname{argmax}_{x \in \mathcal{P}} c^\top x$.

Bounding in Branch and Bound

- The bounding problem is a problem solved to obtain a bound on the optimal solution value of a subproblem $\max_{\mathcal{S}_i} c^\top x$.
- Typically, the bounding problem is either a relaxation or a dual of the subproblem.
- 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.
- 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.

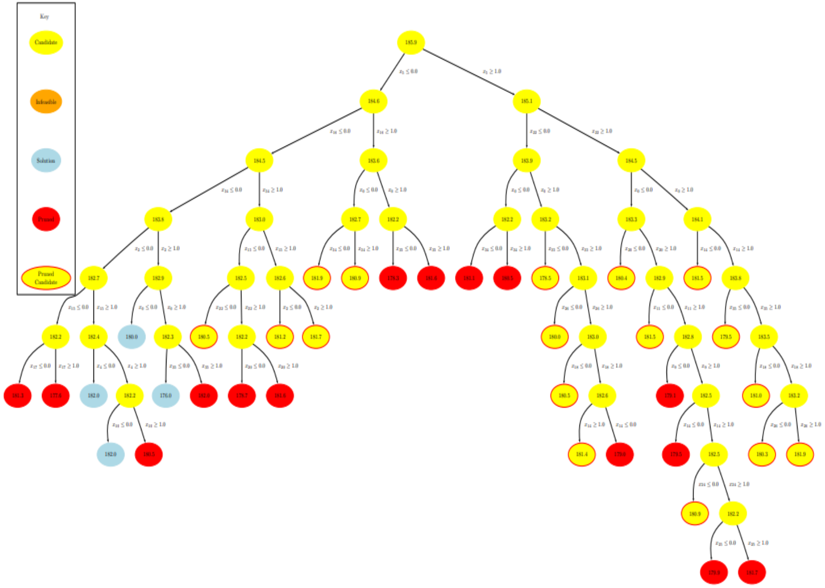
Branching in LP-based Branch and Bound

- 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?

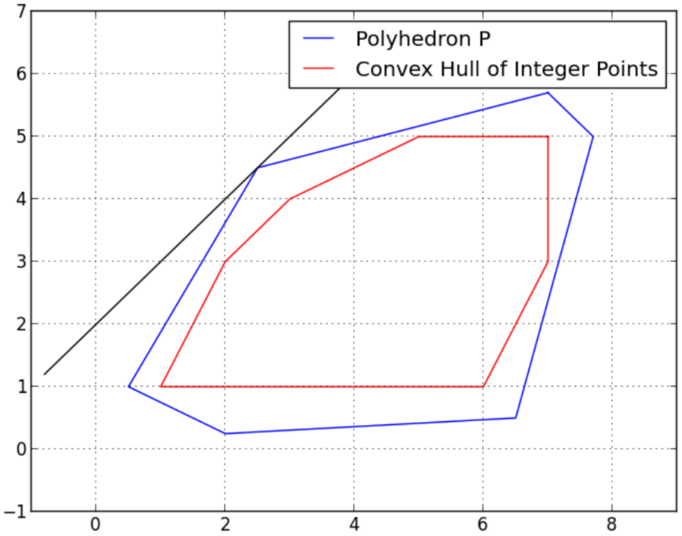
LP-based Branch and Bound Algorithm

- 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.

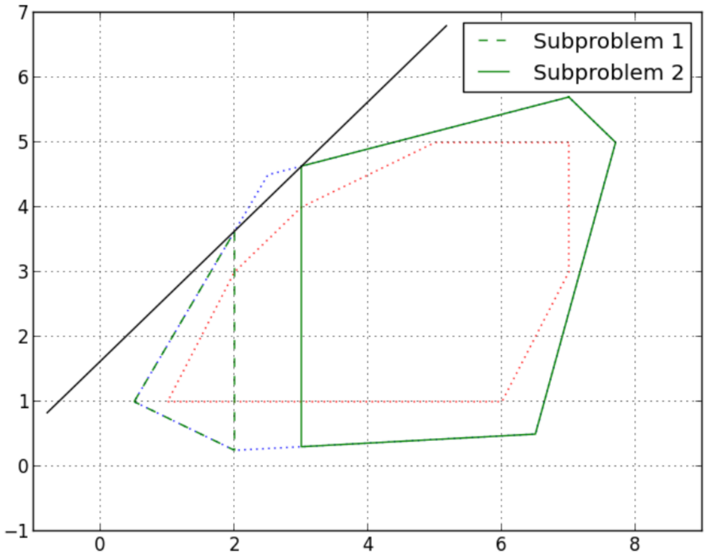
Branch and Bound Tree



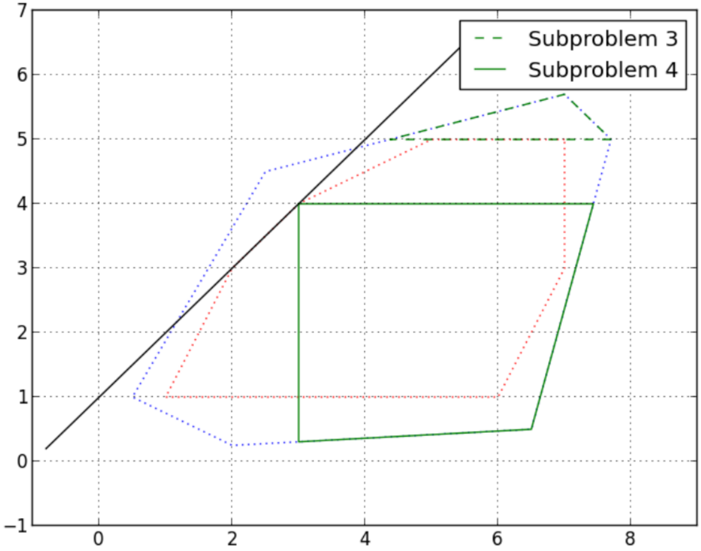
The Geometry of Branching



The Geometry of Branching (cont'd)



The Geometry of Branching



Continuing the Algorithm After Branching

- After branching, we solve each of the subproblems recursively.
- 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.

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.

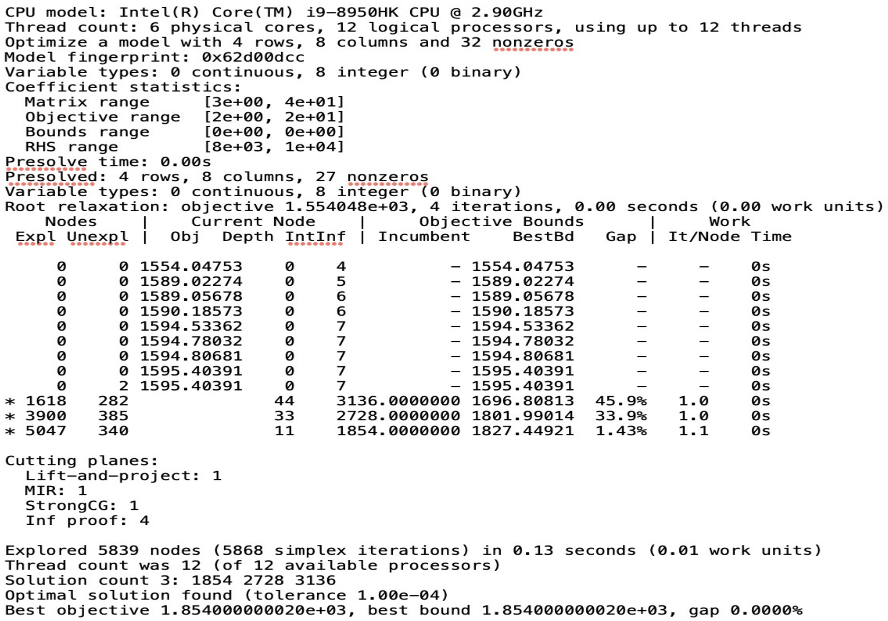
An example solved by Gurobi

```
Aeq = [22     13     26     33     21      3     14     26
    39     16     22     28     26     30     23     24
    18     14     29     27     30     38     26     26
    41     26     28     36     18     38     16     26];
beq = [ 7872 10466 11322 12058]';
q = [2     10     13     17      7      5      7      3]';

N = 8;
lb = zeros(N,1);

% Gurobi
model.A = sparse(Aeq);
model.obj = q;
model.rhs = beq;
model.sense = '=';
model.vtype = 'I';
model.lb = lb;
model.modelsense = 'min';
params.outputflag = 1;
result = gurobi(model, params);
u = result.x;
```

An example solved by Gurobi



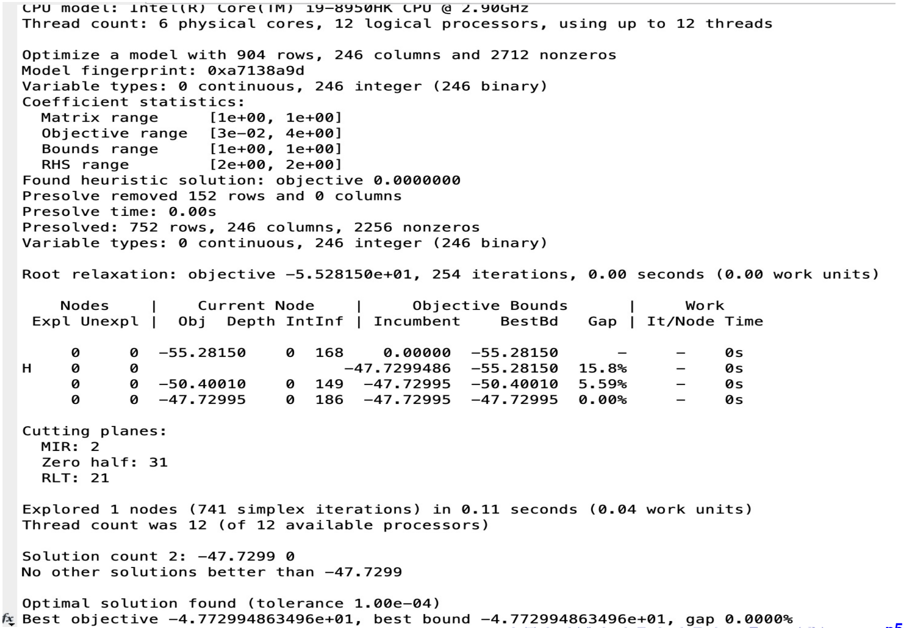
```
CPU model: Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads
Optimize a model with 4 rows, 8 columns and 32 nonzeros
Model fingerprint: 0x62d00dcc
Variable types: 0 continuous, 8 integer (0 binary)
Coefficient statistics:
  Matrix range     [3e+00, 4e+01]
  Objective range  [2e+00, 2e+01]
  Bounds range     [0e+00, 0e+00]
  RHS range        [8e+03, 1e+04]
Presolve time: 0.00s
Presolved: 4 rows, 8 columns, 27 nonzeros
Variable types: 0 continuous, 8 integer (0 binary)
Root relaxation: objective 1.554048e+03, 4 iterations, 0.00 seconds (0.00 work units)
    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0 1554.04753    0    4          - 1554.04753      -     -    0s
     0     0 1589.02274    0    5          - 1589.02274      -     -    0s
     0     0 1589.05678    0    6          - 1589.05678      -     -    0s
     0     0 1590.18573    0    6          - 1590.18573      -     -    0s
     0     0 1594.53362    0    7          - 1594.53362      -     -    0s
     0     0 1594.78032    0    7          - 1594.78032      -     -    0s
     0     0 1594.80681    0    7          - 1594.80681      -     -    0s
     0     0 1595.40391    0    7          - 1595.40391      -     -    0s
     0     2 1595.40391    0    7          - 1595.40391      -     -    0s
* 1618   282              44    3136.0000000 1696.80813  45.9%   1.0    0s
* 3900   385              33    2728.0000000 1801.99014  33.9%   1.0    0s
* 5047   340              11    1854.0000000 1827.44921  1.43%   1.1    0s

Cutting planes:
  Lift-and-project: 1
  MIR: 1
  StrongCG: 1
  Inf proof: 4

Explored 5839 nodes (5868 simplex iterations) in 0.13 seconds (0.01 work units)
Thread count was 12 (of 12 available processors)
Solution count 3: 1854 2728 3136
Optimal solution found (tolerance 1.00e-04)
Best objective 1.854000000020e+03, best bound 1.854000000020e+03, gap 0.0000%
```

Another example solved by Gurobi



```
CPU model: Intel(R) Core(TM) i9-8950HK CPU @ 2.90GHz
Thread count: 6 physical cores, 12 logical processors, using up to 12 threads

Optimize a model with 904 rows, 246 columns and 2712 nonzeros
Model fingerprint: 0xa7138a9d
Variable types: 0 continuous, 246 integer (246 binary)
Coefficient statistics:
  Matrix range     [1e+00, 1e+00]
  Objective range  [3e-02, 4e+00]
  Bounds range     [1e+00, 1e+00]
  RHS range        [2e+00, 2e+00]
Found heuristic solution: objective 0.0000000
Presolve removed 152 rows and 0 columns
Presolve time: 0.00s
Presolved: 752 rows, 246 columns, 2256 nonzeros
Variable types: 0 continuous, 246 integer (246 binary)

Root relaxation: objective -5.528150e+01, 254 iterations, 0.00 seconds (0.00 work units)

    Nodes    |    Current Node    |     Objective Bounds      |     Work
 Expl Unexpl |  Obj  Depth IntInf | Incumbent    BestBd   Gap | It/Node Time

     0     0  -55.28150    0  168    0.00000  -55.28150      -     -    0s
H    0     0                       -47.7299486  -55.28150  15.8%     -    0s
     0     0  -50.40010    0  149  -47.72995  -50.40010  5.59%     -    0s
     0     0  -47.72995    0  186  -47.72995  -47.72995  0.00%     -    0s

Cutting planes:
  MIR: 2
  Zero half: 31
  RLT: 21

Explored 1 nodes (741 simplex iterations) in 0.11 seconds (0.04 work units)
Thread count was 12 (of 12 available processors)

Solution count 2: -47.7299 0
No other solutions better than -47.7299

Optimal solution found (tolerance 1.00e-04)
Best objective -4.772994863496e+01, best bound -4.772994863496e+01, gap 0.0000%
```

Outline

The Efficiency of Branch and Bound

- 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

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

  where

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

- **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:

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

Importance of Relaxations

- The main purpose of a relaxation is to obtain an upper bound on z_{IP} .
- Solving a relaxation is 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.

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

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

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

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

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.

Outline

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.
- 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.

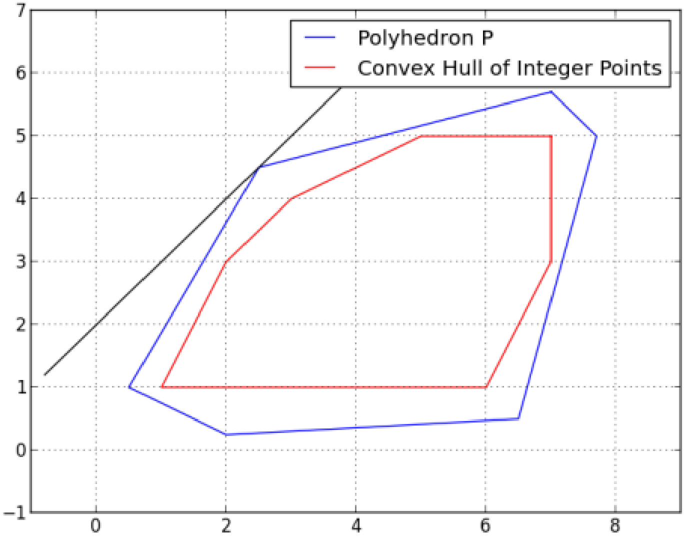
Variable Disjunctions

- 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_i.
- 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

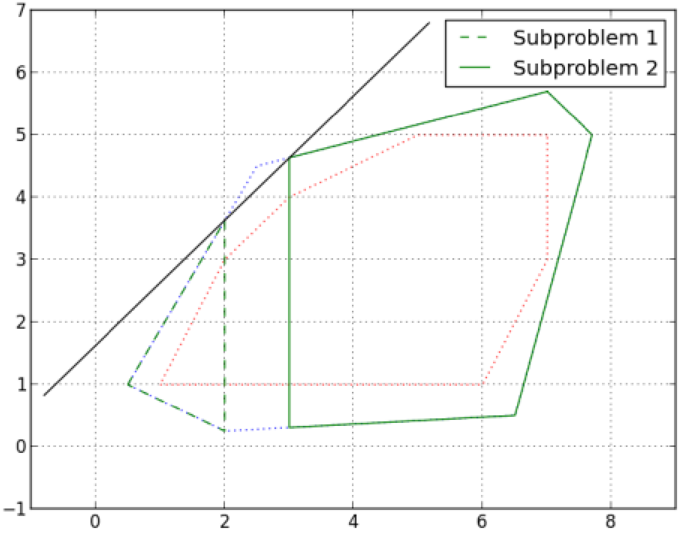
$$x_j = 0 \quad \text{OR} \quad x_j = 1$$

- 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?

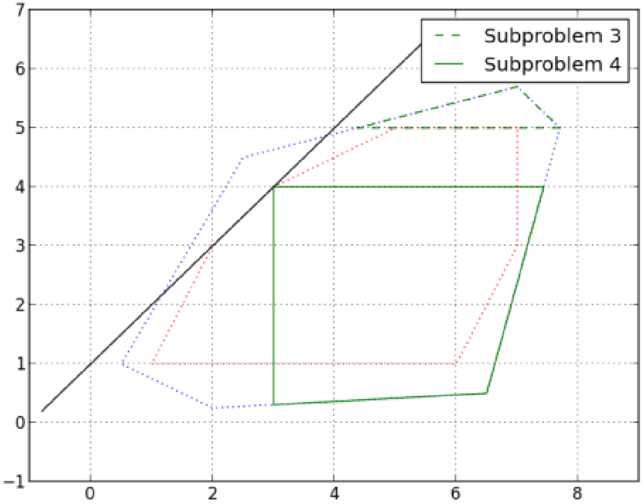
The Geometry of Branching



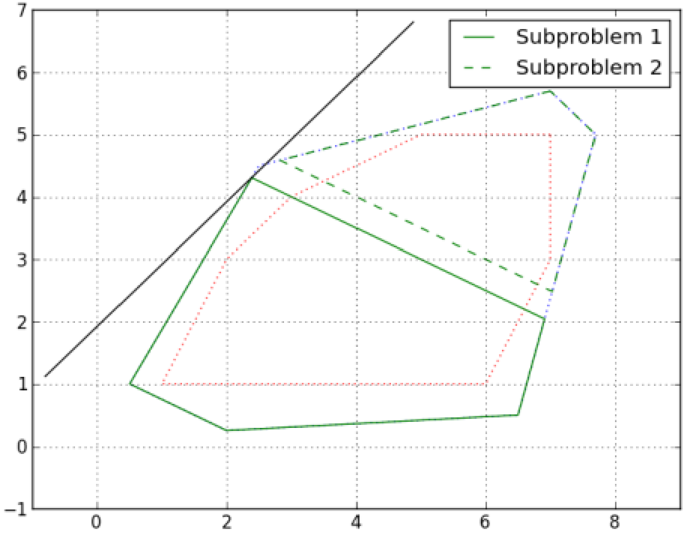
The Geometry of Branching (Variable Disjunction)



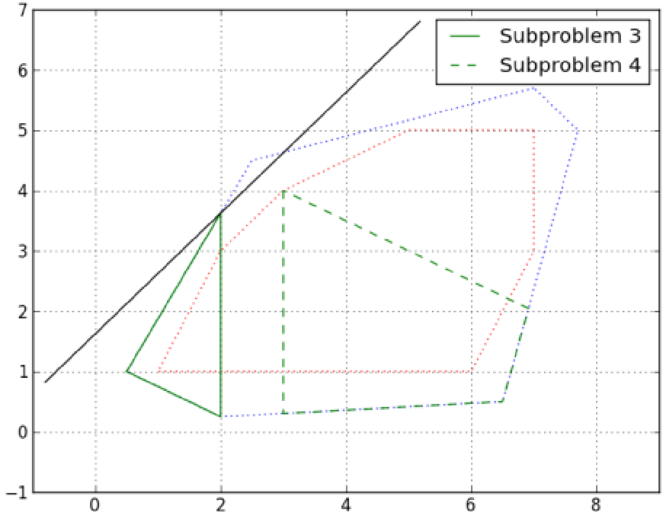
The Geometry of Branching (Variable Disjunction)



The Geometry of Branching (General Split Disjunction)



The Geometry of Branching (General Split Disjunction)



Outline

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

- We have seen that, in theory, $\mathrm{conv}(\mathcal{S})$ 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).

Cutting Planes

- Recall that the inequality denoted by (π, π_0) is valid for a polyhedron $\mathcal{P}$ if $\pi^\top x \leq \pi_0, \forall x \in \mathcal{P}$.
- The term cutting plane usually refers to an inequality valid for $\mathrm{conv}(\mathcal{S})$, 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

- The problem of generating a cutting plane can be stated as: **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 $\pi^\top x^* > \pi_0$.
- 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 $\mathrm{conv}(\mathcal{S})$.
5     if $x^k \in \mathrm{conv}(\mathcal{S})$ then STOP;
6     else Get an inequality $(\pi^k, \pi_0^k)$ valid for $\mathrm{conv}(\mathcal{S})$ but
        $(\pi^k)^\top x^k > \pi_0^k$ ;
7     $\mathcal{P}_{k+1} \leftarrow \mathcal{P}_k \cap \{x \in \mathbb{R}^n \mid (\pi^k)^\top x \leq \pi_0^k\}$.
8     $k \leftarrow k + 1$.
```

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

Consider the MILP

$$z_{IP} = \max c^\top x, \text{ s.t. } x \in S,$$

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

- All inequalities valid for $\mathcal{P}$ are also valid for $\mathrm{conv}(\mathcal{S})$, but they are not cutting planes.
- We need the following simple principle: if $a \le b$ and a is an integer, then $a \le \lfloor b \rfloor$.
- This simple fact is all we need to generate all valid inequalities for $\mathrm{conv}(\mathcal{S})$!
- Example: suppose that $2x_1 + x_2 \le 3/2$ is valid for $\mathcal{P}$, then $2x_1 + x_2 \le 1$ is also valid for $\mathrm{conv}(\mathcal{S})$.

Chvátal Inequalities

- split $A = [A_I, A_C]$ according to integer and continuous variables
- Suppose we can find a $u \in \mathbb{R}^m_+$ such that $\pi = A^\top u$ is integer ($A_I^\top u \in \mathbb{Z}^p$ and $A_C^\top u = 0$) and $\pi_0 = u^\top b \notin \mathbb{Z}$.
- 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
$$\{x \in \mathcal{P} \mid \pi^\top x \geq \lfloor \pi_0 \rfloor + 1\} = \emptyset$$
- Such an inequality is called a Chvátal inequality
- Note that we have not used the non-negativity constraints in deriving this inequality

Chvátal-Gomory Inequalities

- Assume that $\mathcal{P} \subset \mathbb{R}^n_+$ and let $u \in \mathbb{R}^n_+$ be such that $A_C^\top u \geq 0$
- Since the variables are nonnegative, we have $u^\top A_C x_C \geq 0$ and

$$\sum_{i=1}^{p} (u^\top A_i) x_i \leq u^\top b, \quad \forall x \in \mathcal{P}$$

- Again, because the variables are nonnegative, we have

$$\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq u^\top b, \quad \forall x \in \mathcal{P}$$

- Finally, we have:

$$\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq \lfloor u^\top b \rfloor, \quad \forall x \in \mathcal{S}$$

- This is the Chvátal-Gomory inequality

Chvátal-Gomory Inequalities: another derivation

- We explicitly add the non-negativity constraints to the formulation along the other constraints with associated multipliers $v \in \mathbb{R}^n_+$
- We cannot round the coefficients to make them integral, so we require π integral

$$\begin{aligned} \pi_i &= u^\top A_i - v_i \in \mathbb{Z} \quad \text{for } 1 \le i \le p \\ \pi_i &= u^\top A_i - v_i = 0 \quad \text{for } p+1 \le i \le n \end{aligned}$$

- v_i will be non-negative as as long as we have

$$\begin{aligned} v_i &\geq u^\top A_i - \lfloor u^\top A_i \rfloor, \quad && \text{for } 0 \le i \le p, \\ v_i &= u^\top A_i \ge 0, && \text{for } \; p+1 \le i \le n. \end{aligned}$$

- Taking $v_i = u^\top A_i - \lfloor uA_i \rfloor$ for $1 \le i \le p$, we obtain

$$\sum_{i=1}^{p} \pi_i x_i = \sum_{i=1}^{p} \lfloor uA_i \rfloor x_i \le \lfloor ub \rfloor = \pi_0$$

is a C-G inequality for all $u \in \mathbb{R}^m_+$ such that $A_C^\top u \ge 0$

The Chvátal-Gomory Procedure

1. Choose a weight vector $u \in \mathbb{R}^m_+$ such that $A_C^\top u \geq 0$.
2. Obtain the valid inequality $\sum_{i=1}^{p} (u^\top A_i) x_i \leq u^\top b$.
3. Round the coefficients down to obtain $\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq u^\top b$.
4. Finally, round the right hand side down to obtain the valid inequality

$$\sum_{i=1}^{p} \lfloor u^\top A_i \rfloor x_i \leq \lfloor u^\top b \rfloor$$

- 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.

Gomory Inequalities

- Consider the set of solutions to a pure ILP with one equation:

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

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

$$T = \left\{ x \in \mathbb{Z}_+^n \mid \sum_{j=1}^{n} f_j x_j = f_0 + \lfloor a_0 \rfloor - \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \right\}$$

- Since $\sum_{j=1}^{n} f_j x_j \geq 0$ and $f_0 < 1$, then $\lfloor a_0 \rfloor - \sum_{j=1}^{n} \lfloor a_j \rfloor x_j \geq 0$ and so

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

is a valid inequality for $\mathcal{S}$ called a Gomory inequality.

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?