Algorithms for Mixed Integer Linear Programming

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

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

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:

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

- 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.

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.
- 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.

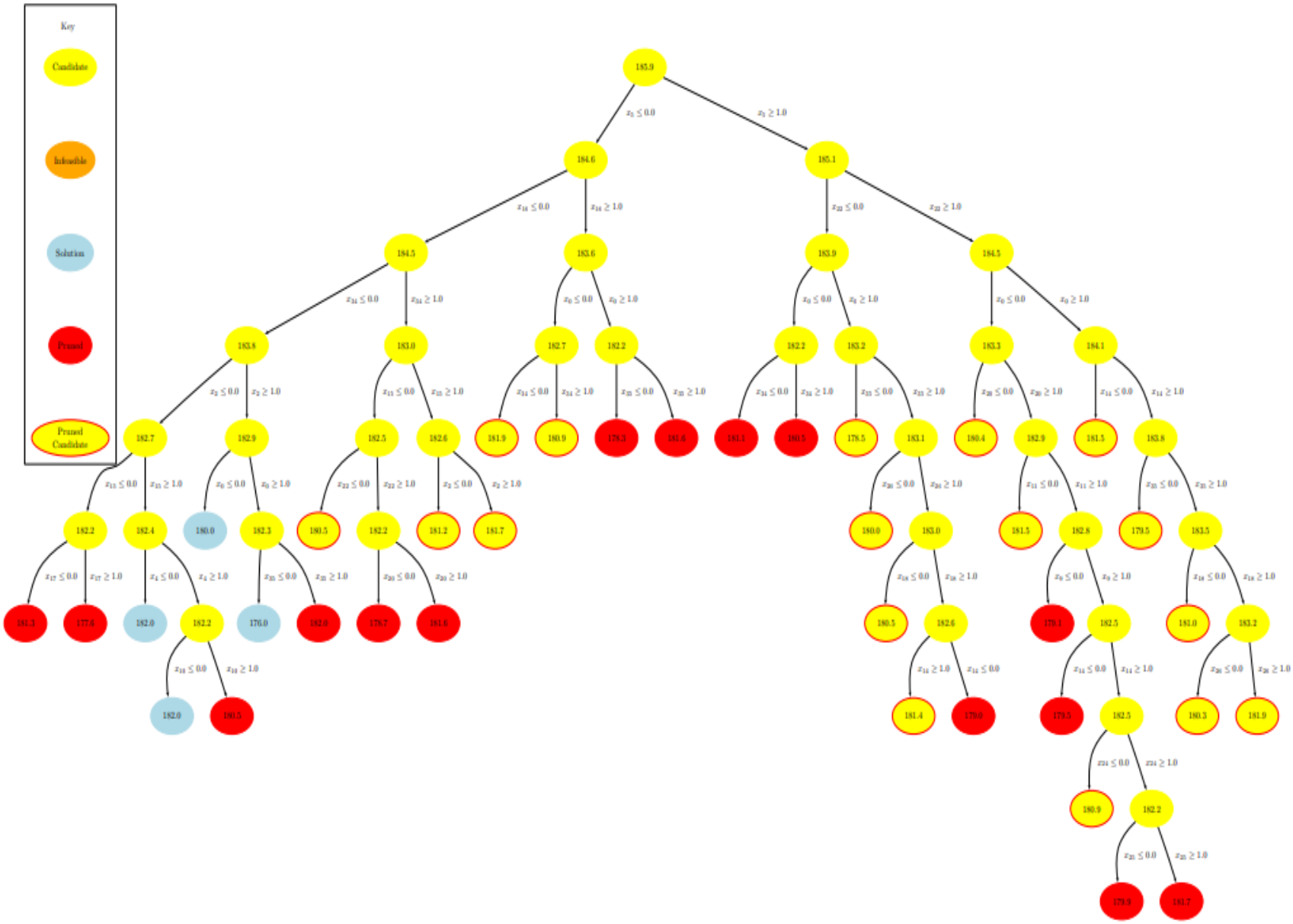
Branching in LP-based Branch and Bound

- 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?

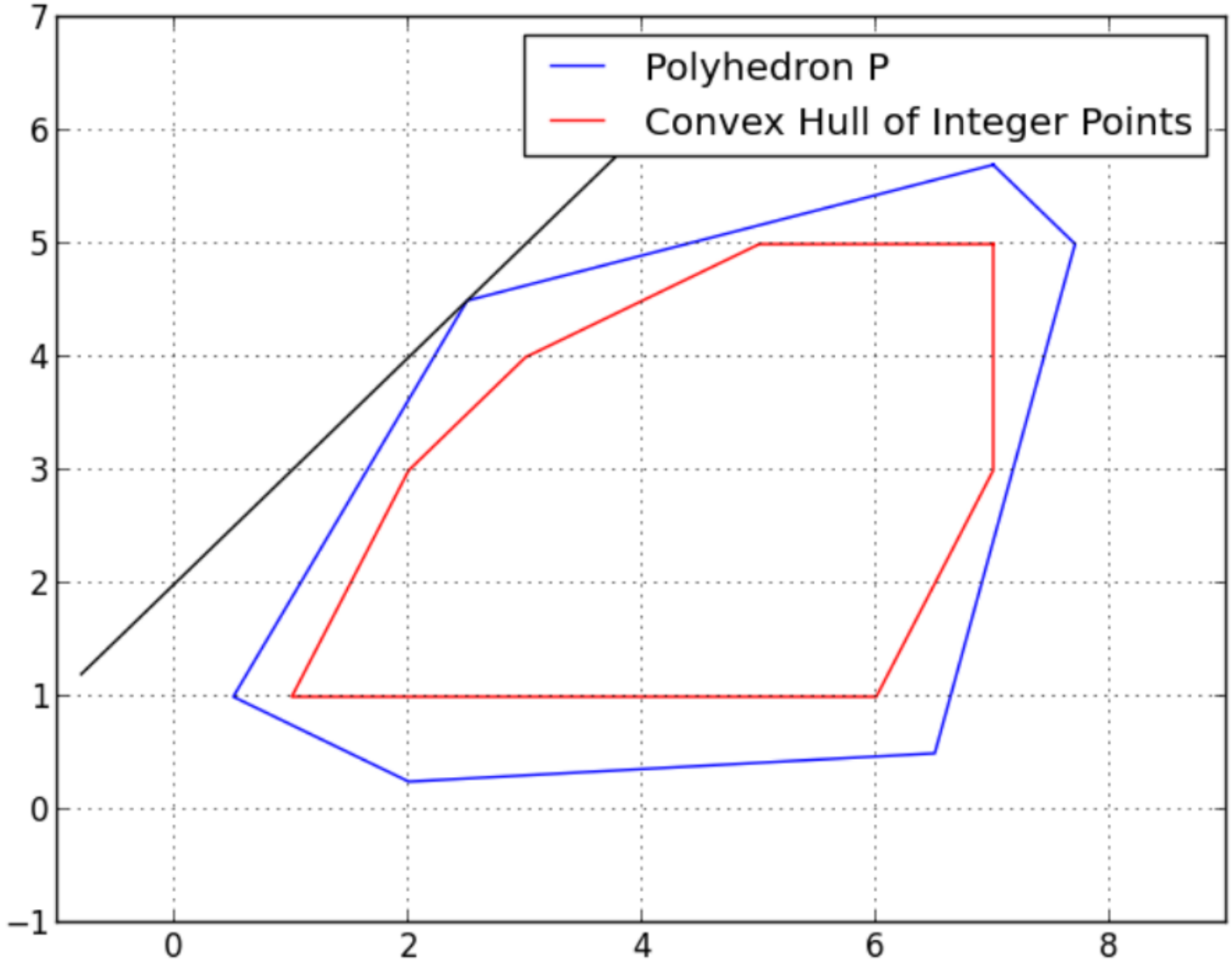
LP-based Branch and Bound Algorithm

- 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.

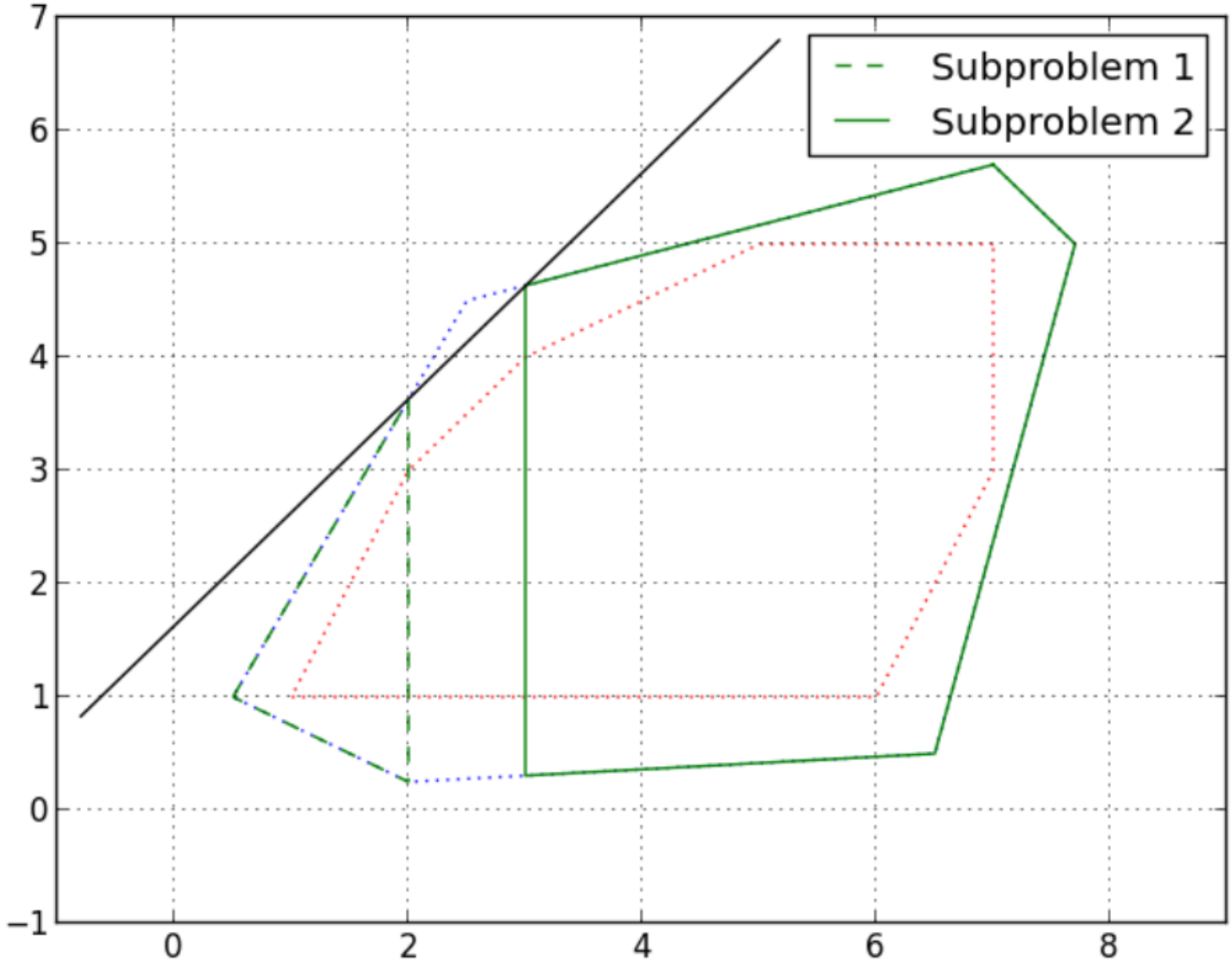
Branch and Bound Tree



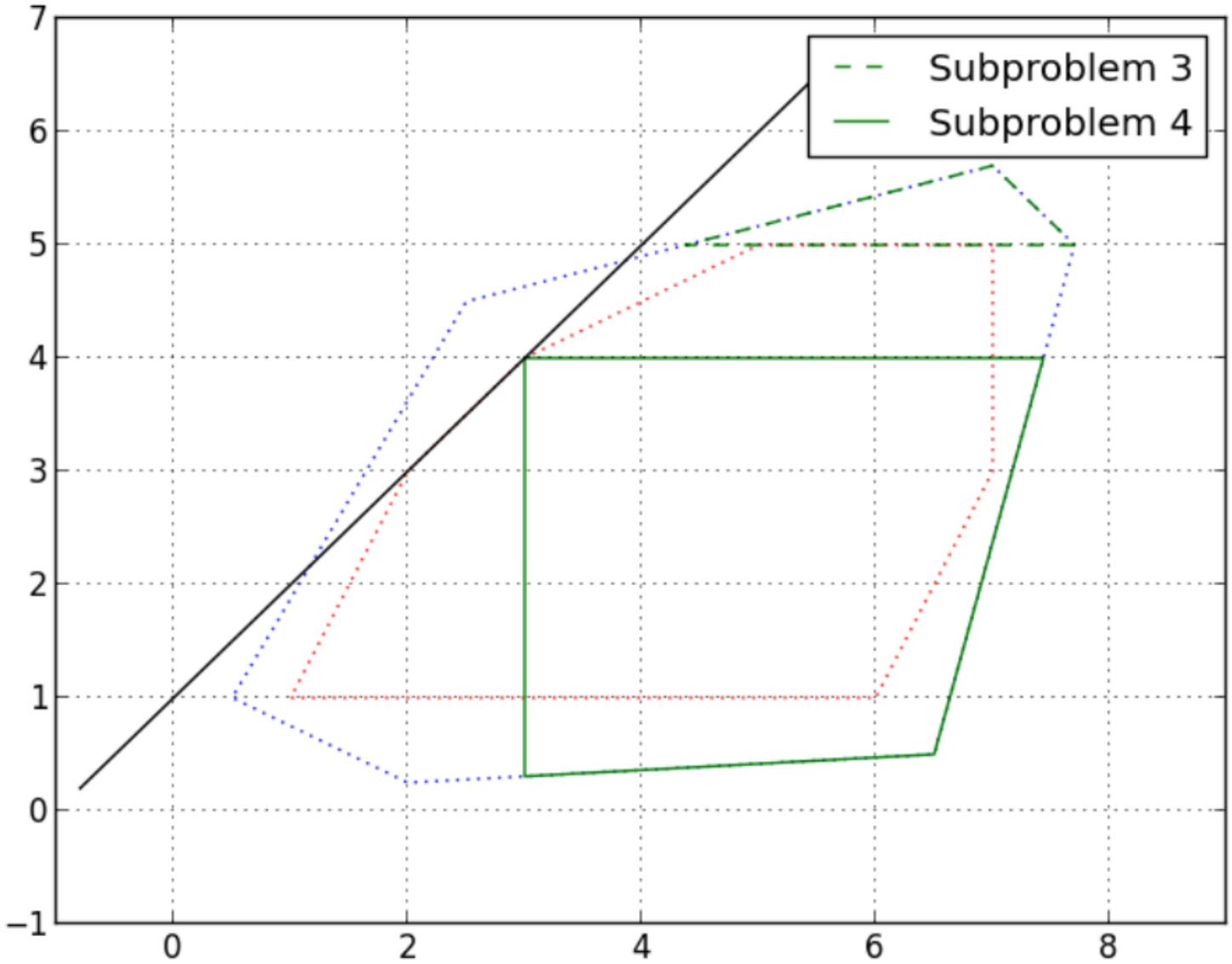
The Geometry of Branching



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 1.85400000020e+03, best bound 1.85400000020e+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

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

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 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

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

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

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

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.

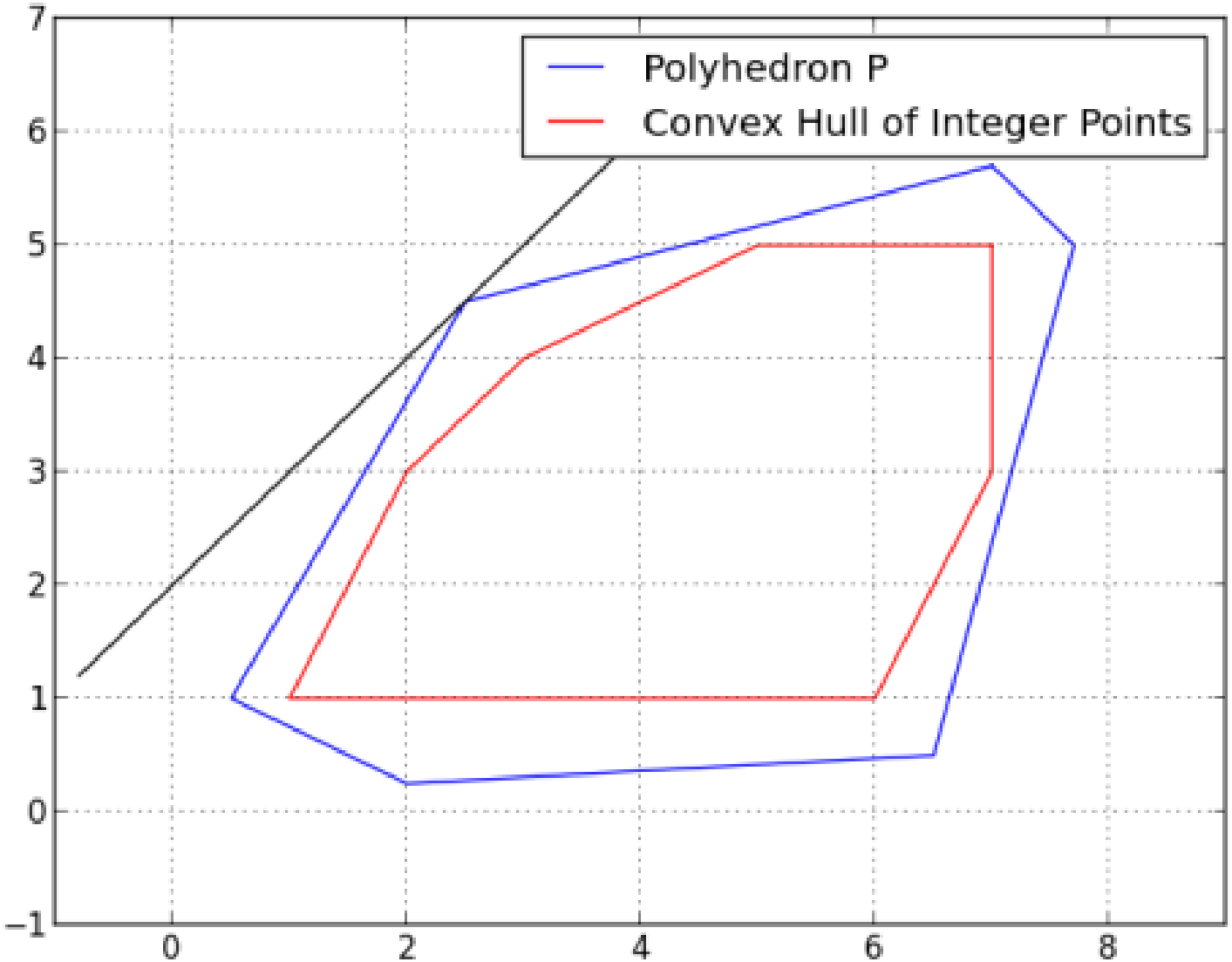
Variable Disjunctions

- 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

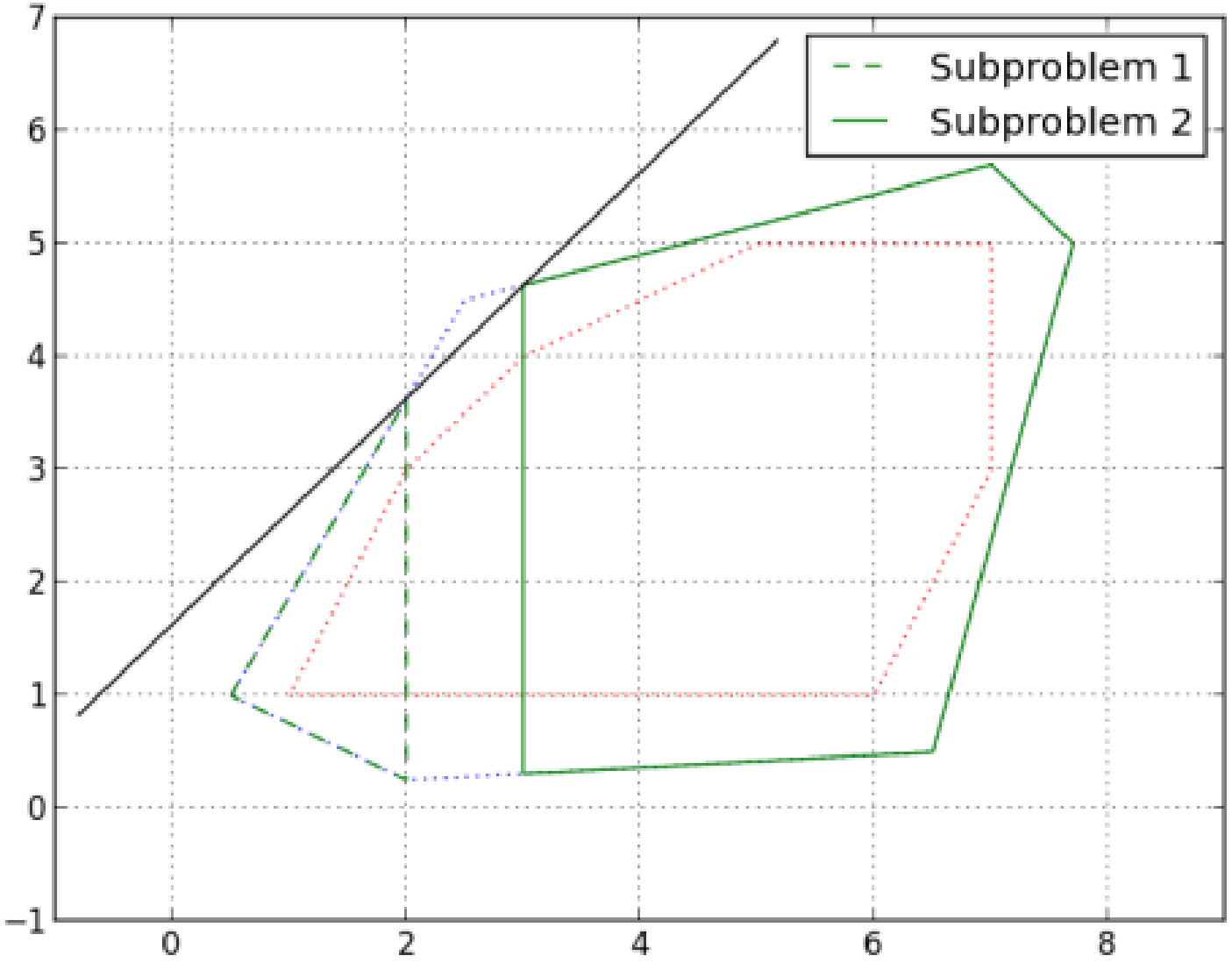
$$x_j = 0 \quad \text{OR} \quad x_j = 1$$

- 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?

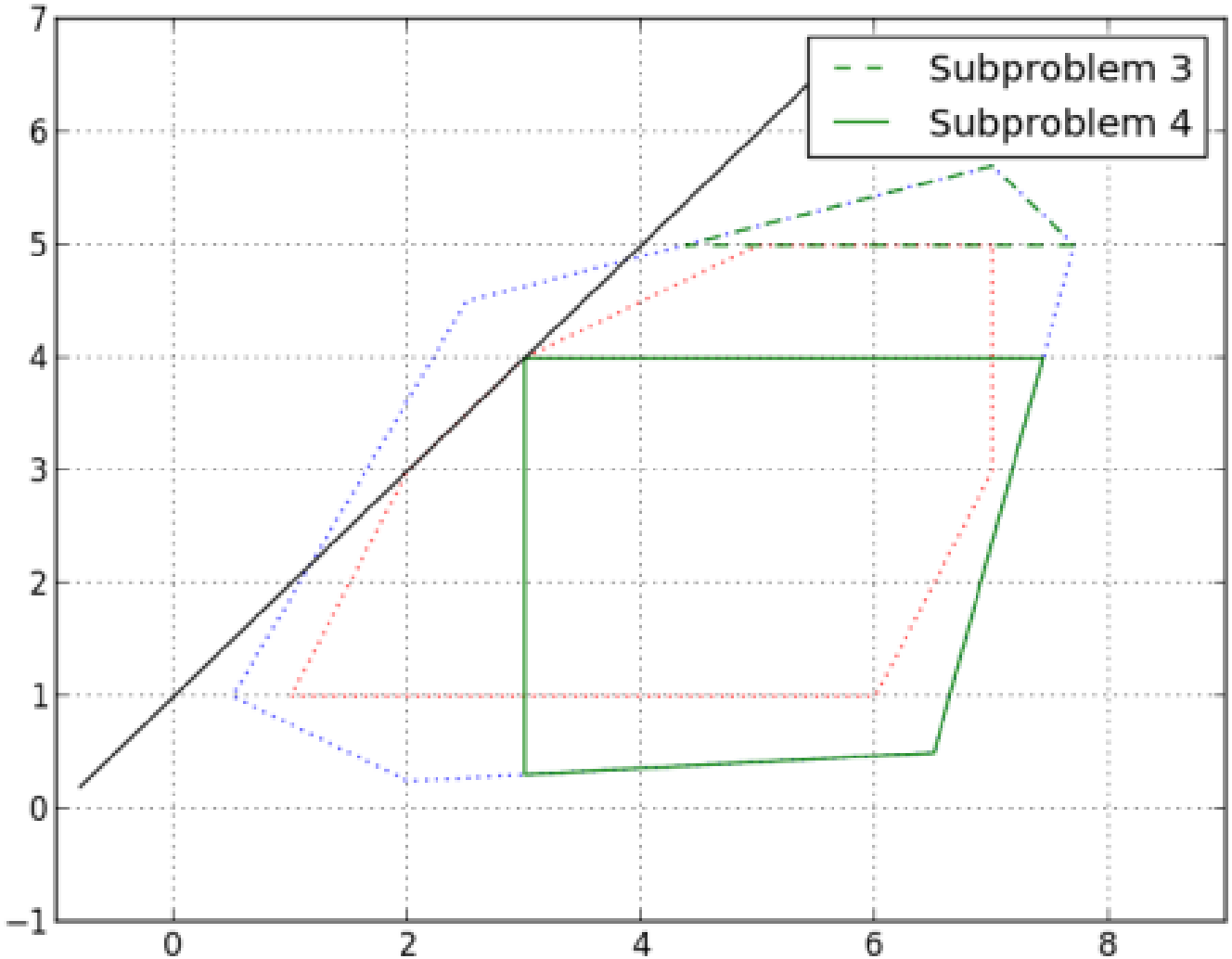
The Geometry of Branching



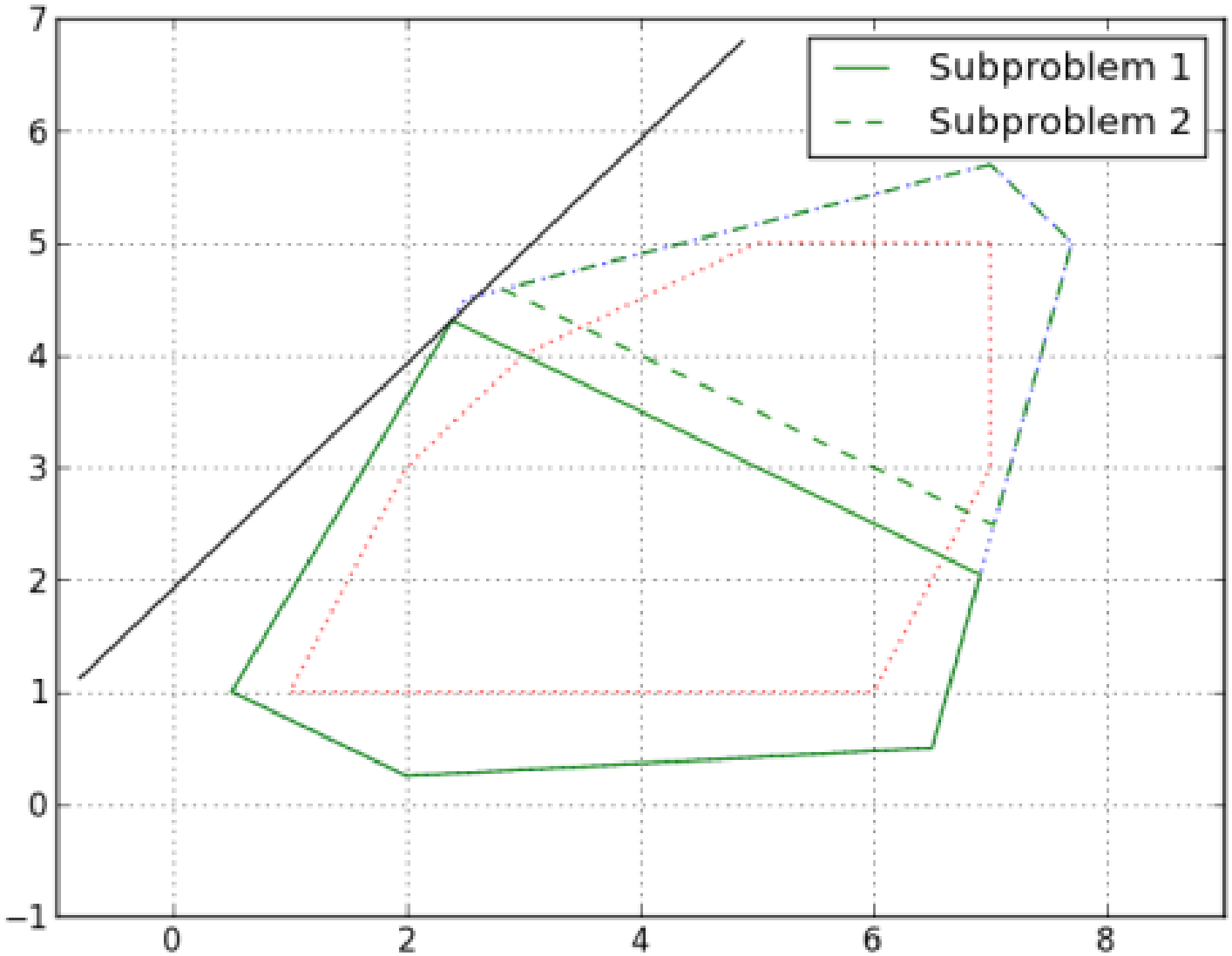
The Geometry of Branching (Variable Disjunction)



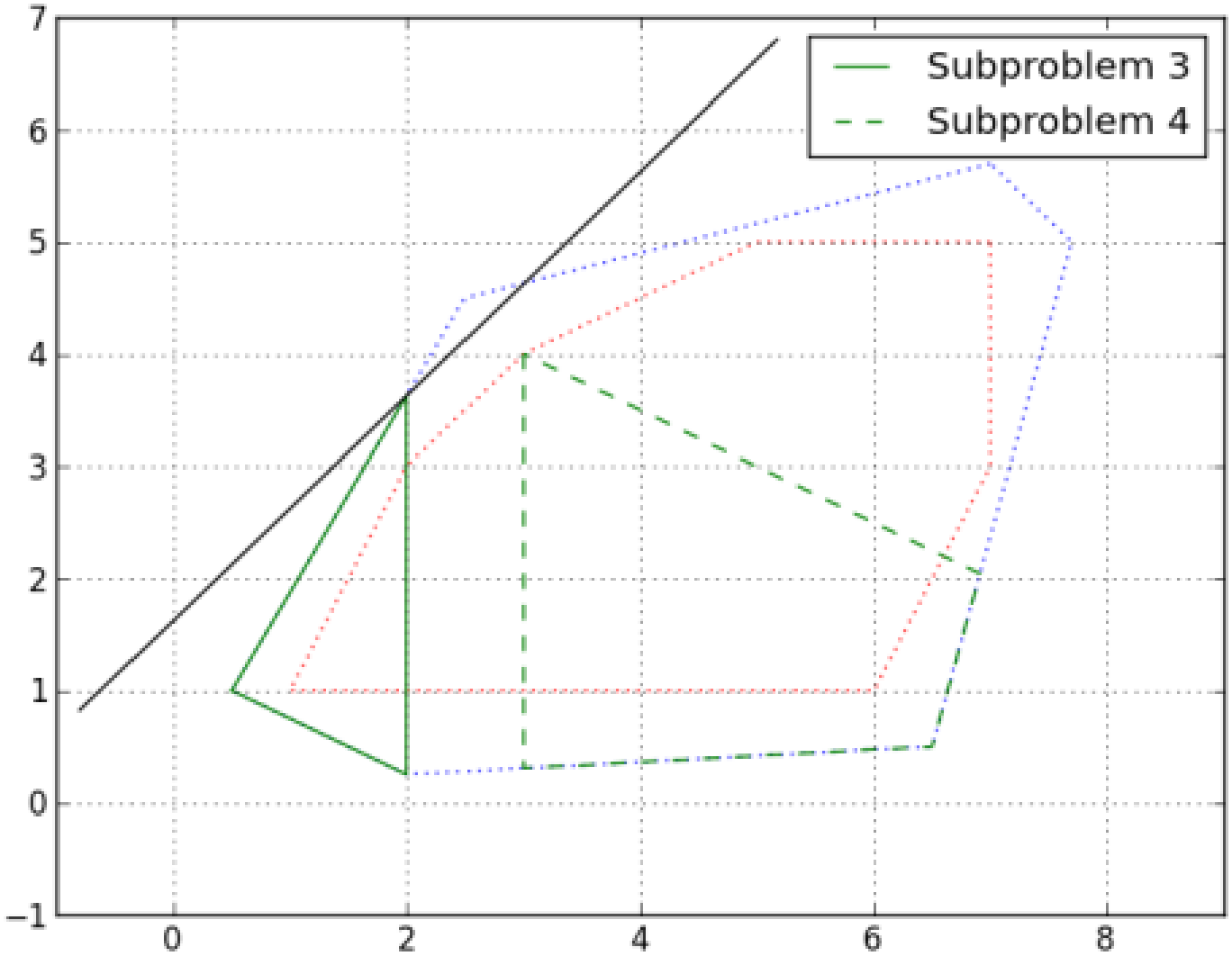
The Geometry of Branching (Variable Disjunction)



The Geometry of Branching (General Split Disjunction)



The Geometry of Branching (General Split Disjunction)



Outline

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

- 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).

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

- 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 $\text{conv}(\mathcal{S})$.
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 $k \leftarrow k + 1$.

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

Consider the MILP

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

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

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

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

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

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

$$v_i \geq u^\top A_i - \lfloor u^\top A_i \rfloor, \quad \text{for } 0 \leq i \leq p,$$
$$v_i = u^\top A_i \geq 0, \quad \text{for } p+1 \leq i \leq n.$$

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

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

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

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.