LINEAR PROGRAMMING

[V. CH9]: INTEGER PROGRAMMING

Phillip Keldenich Ahmad Moradi

Department of Computer Science
Algorithms Department
TU Braunschweig

January 30, 2024

Vertex Cover

For a given graph $G = (V, E)$, the Vertex Cover problem asks for a minimum-cardinality subset $C \subseteq V$ of vertices such that each edge $vw \in E$ has least one endpoint in C, i.e., $\{v, w\} \cap C \neq \emptyset$.

VERTEX COVER

For a given graph $G = (V, E)$, the Vertex Cover problem asks for a minimum-cardinality subset $C \subseteq V$ of vertices such that each edge $vw \in E$ has least one endpoint in C, i.e., $\{v, w\} \cap C \neq \emptyset$.

Trying to model this as linear program:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

What happens on a triangle (K_3, complete graph on 3 vertices)?

Vertex Cover

For a given graph $G = (V, E)$, the Vertex Cover problem asks for a minimum-cardinality subset $C \subseteq V$ of vertices such that each edge $vw \in E$ has least one endpoint in C, i.e., $\{v, w\} \cap C \neq \emptyset$.

Trying to model this as linear program:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

What happens on a triangle (K_3, complete graph on 3 vertices)?

Notes: Vertex Cover is NP-hard! LP is not NP-hard unless $\mathsf{P} = \mathsf{NP}$.

Vertex Cover

For a given graph $G = (V, E)$, the Vertex Cover problem asks for a minimum-cardinality subset $C \subseteq V$ of vertices such that each edge $vw \in E$ has least one endpoint in C, i.e., $\{v, w\} \cap C \neq \emptyset$.

Trying to model this as linear program:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

What happens on a triangle (K_3, complete graph on 3 vertices)?

Notes: Vertex Cover is NP-hard! LP is not NP-hard unless $\mathsf{P} = \mathsf{NP}$.
LP is in P, even though Simplex is not a polynomial-time algorithm.

VERTEX COVER

For a given graph $G = (V, E)$, the Vertex Cover problem asks for a minimum-cardinality subset $C \subseteq V$ of vertices such that each edge $vw \in E$ has least one endpoint in C, i.e., $\{v, w\} \cap C \neq \emptyset$.

Trying to model this as linear program:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

What happens on a triangle (K_3, complete graph on 3 vertices)?

Notes: Vertex Cover is NP-hard! LP is not NP-hard unless P = NP.
LP is in P, even though Simplex is not a polynomial-time algorithm.
Unless P = NP, we thus cannot expect to fully model Vertex Cover as LP!

Vertex Cover

For a given graph $G = (V, E)$, the Vertex Cover problem asks for a minimum-cardinality subset $C \subseteq V$ of vertices such that each edge $vw \in E$ has least one endpoint in C, i.e., $\{v, w\} \cap C \neq \emptyset$.

Trying to model this as linear program:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

What happens on a triangle (K_3, complete graph on 3 vertices)?

Notes: Vertex Cover is NP-hard! LP is not NP-hard unless $\mathsf{P} = \mathsf{NP}$.
LP is in P, even though Simplex is not a polynomial-time algorithm.
Unless $\mathsf{P} = \mathsf{NP}$, we thus cannot expect to fully model Vertex Cover as LP!
Idea: Extend LP to be able to model NP-hard problems! Any ideas?

VERTEX COVER MODEL

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

Vertex Cover Model

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \le x_v \le 1$$

$$\forall vw \in E : x_v + x_w \ge 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

VERTEX COVER MODEL

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?

Vertex Cover Model

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$
$$\forall v \in V : 0 \leq x_v \leq 1$$
$$\forall vw \in E : x_v + x_w \geq 1$$
$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?
Variable $x_v \in \{0, 1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Vertex Cover Model

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?
Variable $x_v \in \{0, 1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Value of literal $\ell = v$: x_v, $\bar{\ell} =$

Vertex Cover Model

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?

Variable $x_v \in \{0,1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Value of literal $\ell = v$: x_v, $\bar{\ell} = 1 - x_v$

VERTEX COVER MODEL

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?
Variable $x_v \in \{0, 1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Value of literal $\ell = v$: x_v, $\bar{\ell} = 1 - x_v$
Modeling a clause, e.g., $v_1 \vee \overline{v_2} \vee \overline{v_3}$:

VERTEX COVER MODEL

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?

Variable $x_v \in \{0, 1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Value of literal $\ell = v$: x_v, $\bar{\ell} = 1 - x_v$

Modeling a clause, e.g., $v_1 \vee \overline{v_2} \vee \overline{v_3}$: $x_{v_1} + (1 - x_{v_2}) + (1 - x_{v_3}) \geq 1$

VERTEX COVER MODEL

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?
Variable $x_v \in \{0, 1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Value of literal $\ell = v$: x_v, $\bar{\ell} = 1 - x_v$
Modeling a clause, e.g., $v_1 \vee \overline{v_2} \vee \overline{v_3}$: $x_{v_1} + (1 - x_{v_2}) + (1 - x_{v_3}) \geq 1$

Notes: Obviously, LP with integer variables is NP-hard.

VERTEX COVER MODEL

Adapting our model:

$$\min \sum_{v \in V} x_v \text{ s.t.}$$

$$\forall v \in V : 0 \leq x_v \leq 1$$

$$\forall vw \in E : x_v + x_w \geq 1$$

$$\forall v \in V : x_v \in \mathbb{Z}$$

Modeling SAT: How can we do that?
Variable $x_v \in \{0, 1\}$ for each Boolean variable v in the formula $\varphi = \bigwedge_i C_i$.

Value of literal $\ell = v$: x_v, $\bar{\ell} = 1 - x_v$
Modeling a clause, e.g., $v_1 \vee \overline{v_2} \vee \overline{v_3}$: $x_{v_1} + (1 - x_{v_2}) + (1 - x_{v_3}) \geq 1$

Notes: Obviously, LP with integer variables is NP-hard.
Even deciding feasibility is NP-hard (see SAT example).

Motivation

Definition

Branch and Bound

Branch and Cut

Some Cutting Plane Templates

Integer Program

A linear program where all variables are restricted to $\mathbb{Z}$ is called *integer program* (IP).

A linear program where some (but not all) variables are restricted to $\mathbb{Z}$ is called *mixed integer program* (MIP).

A linear program where all variables are restricted to $\{0, 1\}$ is called 0-1-program or binary program.

0-1-programs, IP and MIP are NP-complete.
They can be used to straightforwardly model many NP-complete problems.
Good solvers exist that can solve small to moderate size instances of many NP-hard problems.

Integer Program

A linear program where all variables are restricted to $\mathbb{Z}$ is called *integer program* (IP).

A linear program where some (but not all) variables are restricted to $\mathbb{Z}$ is called *mixed integer program* (MIP).

A linear program where all variables are restricted to $\{0, 1\}$ is called 0-1-program or binary program.

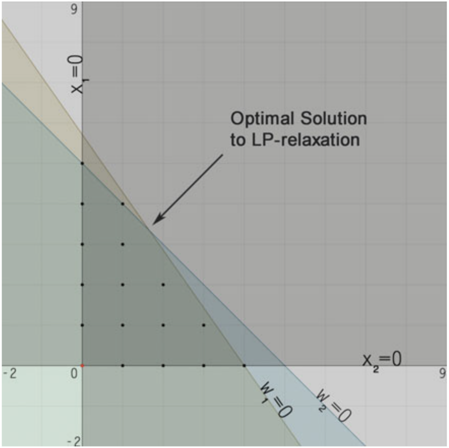
0-1-programs, IP and MIP are NP-complete.
They can be used to straightforwardly model many NP-complete problems.
Good solvers exist that can solve small to moderate size instances of many NP-hard problems.

Given some (mixed) integer program I, the LP we obtain by removing the integrality constraints is called *linear relaxation* of I.

SOLVING IPS

How do we solve integer programs?
Using a technique called Branch & Bound, or an extension of that; let's show an example.

$$\max 17x_1 + 12x_2 \text{ s.t.}$$
$$10x_1 + 7x_2 \leq 40$$
$$x_1 + x_2 \leq 5$$
$$x_1, x_2 \geq 0$$
$$x_1, x_2 \in \mathbb{Z}$$



Solving the LP relaxation (of subproblem P_0, the original problem) gives us $\zeta^0 = 68 + 1/3, x_1^0 = 5/3, x_2^0 = 10/3$.
This tells us the optimal (integer) solution is not better than ζ^0.

Branch & Bound: Example

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

BRANCH & BOUND: EXAMPLE

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

- We branch, i.e., split P_i into (at least two) new subproblems.

Branch & Bound: Example

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

- We branch, i.e., split P_i into (at least two) new subproblems.
- Together, the subproblems must cover all possible integer solutions in P_i.

Branch & Bound: Example

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

- We branch, i.e., split P_i into (at least two) new subproblems.
- Together, the subproblems must cover all possible integer solutions in P_i.
- None of the subproblems must contain the non-integral optimal solution of P_i.

Branch & Bound: Example

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

- We branch, i.e., split P_i into (at least two) new subproblems.
- Together, the subproblems must cover all possible integer solutions in P_i.
- None of the subproblems must contain the non-integral optimal solution of P_i.
- Ideally, they should also be disjoint, i.e., each solution is contained in at most one of the new problems.

Branch & Bound: Example

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

- We branch, i.e., split P_i into (at least two) new subproblems.
- Together, the subproblems must cover all possible integer solutions in P_i.
- None of the subproblems must contain the non-integral optimal solution of P_i.
- Ideally, they should also be disjoint, i.e., each solution is contained in at most one of the new problems.
- In the example, $x_1^0 = 5/3$; in any integer solution, we must have $x_1 \leq 1$ or $x_1 \geq 2$. We create two new subproblems P_1 (by adding $x_1 \leq 1$) and P_2 (by adding $x_1 \geq 2$) to the original constraints.

BRANCH & BOUND: EXAMPLE

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

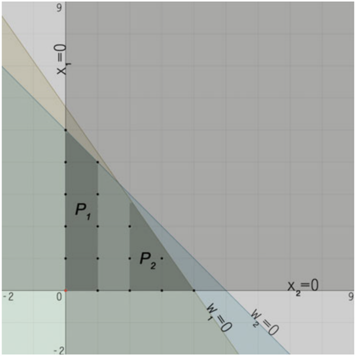
- We branch, i.e., split P_i into (at least two) new subproblems.
- Together, the subproblems must cover all possible integer solutions in P_i.
- None of the subproblems must contain the non-integral optimal solution of P_i.
- Ideally, they should also be disjoint, i.e., each solution is contained in at most one of the new problems.
- In the example, $x_1^0 = 5/3$; in any integer solution, we must have $x_1 \leq 1$ or $x_1 \geq 2$. We create two new subproblems P_1 (by adding $x_1 \leq 1$) and P_2 (by adding $x_1 \geq 2$) to the original constraints.
- In general, we can take any integer variable x with non-integral value θ and use $x \leq \lfloor \theta \rfloor$ and $x \geq \lceil \theta \rceil$ as new constraints.

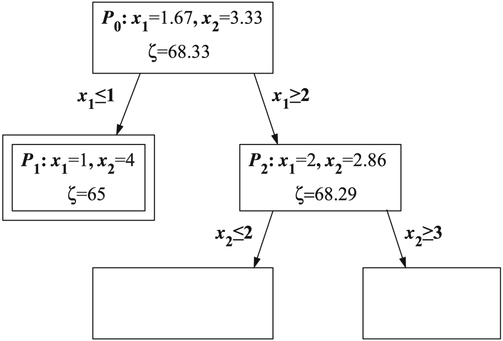
BRANCH & BOUND: EXAMPLE

How do we continue when the LP relaxation of a subproblem P_i has a non-integral optimal solution?

- We branch, i.e., split P_i into (at least two) new subproblems.
- Together, the subproblems must cover all possible integer solutions in P_i.
- None of the subproblems must contain the non-integral optimal solution of P_i.
- Ideally, they should also be disjoint, i.e., each solution is contained in at most one of the new problems.
- In the example, $x_1^0 = 5/3$; in any integer solution, we must have $x_1 \leq 1$ or $x_1 \geq 2$. We create two new subproblems P_1 (by adding $x_1 \leq 1$) and P_2 (by adding $x_1 \geq 2$) to the original constraints.
- In general, we can take any integer variable x with non-integral value θ and use $x \leq \lfloor \theta \rfloor$ and $x \geq \lceil \theta \rceil$ as new constraints.
- The optimal integer solution to P_i is the best integer solution found recursively in the subproblems.

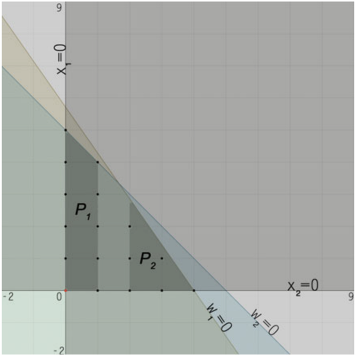
RESULT OF FIRST BRANCHING

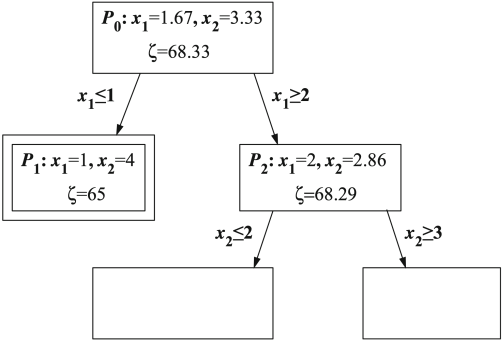




The subproblems form a *search tree*. The relaxation of the left child problem P_1 has an integral solution. It does not need another branch and becomes a leaf of the search tree (double box).

RESULT OF FIRST BRANCHING

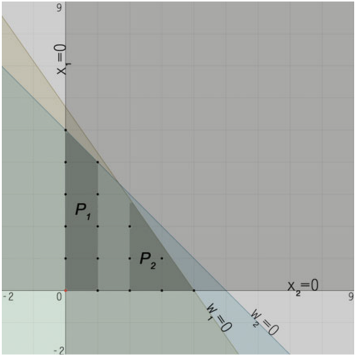


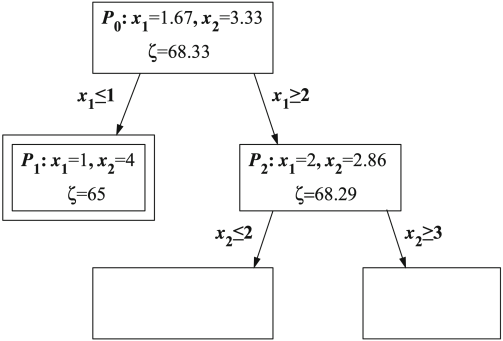


The subproblems form a *search tree*. The relaxation of the left child problem P_1 has an integral solution. It does not need another branch and becomes a leaf of the search tree (double box).

What if the right child had an objective value $\zeta \leq 65$?

RESULT OF FIRST BRANCHING

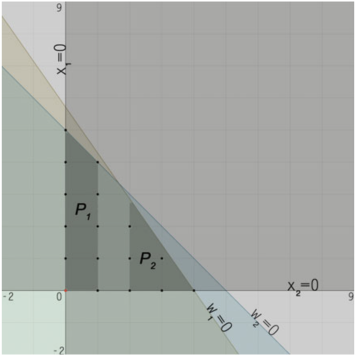


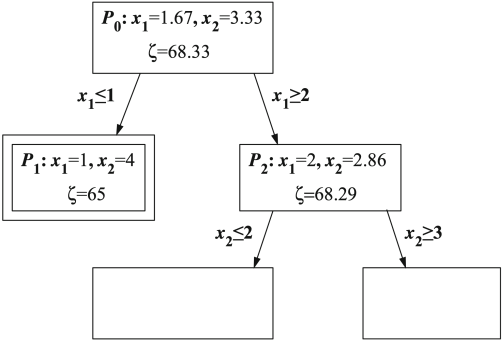


The subproblems form a *search tree*. The relaxation of the left child problem P_1 has an integral solution. It does not need another branch and becomes a leaf of the search tree (double box).

What if the right child had an objective value $\zeta \leq 65$?
We could make it a leaf because its bound is not better than a solution we already found!

Result of First Branching

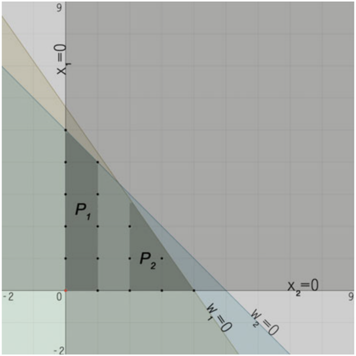


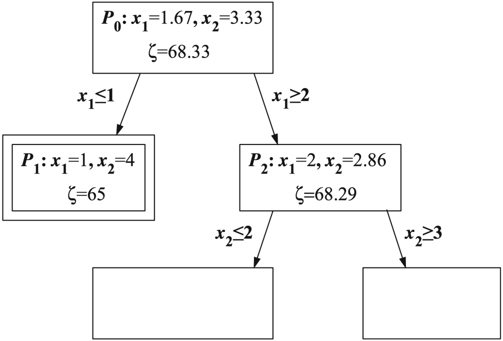


The subproblems form a *search tree*. The relaxation of the left child problem P_1 has an integral solution. It does not need another branch and becomes a leaf of the search tree (double box).

What if the right child had an objective value $\zeta \leq 65$?
We could make it a leaf because its bound is not better than a solution we already found!
This is called pruning and important for making Branch & Bound efficient in practice.

Result of First Branching





The subproblems form a *search tree*. The relaxation of the left child problem P_1 has an integral solution. It does not need another branch and becomes a leaf of the search tree (double box).

What if the right child had an objective value $\zeta \leq 65$?
We could make it a leaf because its bound is not better than a solution we already found!
This is called pruning and important for making Branch & Bound efficient in practice.
Pruning relies on good bounds, i.e., strong LP relaxations. If optimal solutions are much worse than the bounds we obtain, pruning can only be applied rarely and the number of subproblems rises.

Continuing

Exploring a node of the search tree means:

- solving the LP relaxation (most expensive step),
- deciding whether and how to branch.

Continuing

Exploring a node of the search tree means:

- solving the LP relaxation (most expensive step),
- deciding whether and how to branch.

We usually explore nodes in a (sort of) depth-first order. This has several advantages:

- Memory requirements: DFS needs essentially $O(\text{depth})$.
  BFS needs to store a level of the tree (often $\Omega(\text{nodes})$).

Continuing

Exploring a node of the search tree means:

- solving the LP relaxation (most expensive step),
- deciding whether and how to branch.

We usually explore nodes in a (sort of) depth-first order. This has several advantages:

- Memory requirements: DFS needs essentially $O(\text{depth})$.
  BFS needs to store a level of the tree (often $\Omega(\text{nodes})$).
- Integer solutions are often deep in the tree. We need them to prune; earlier is better. When aborting the search, e.g., due to a timeout, we want to have a good solution.

Continuing

Exploring a node of the search tree means:

- solving the LP relaxation (most expensive step),
- deciding whether and how to branch.

We usually explore nodes in a (sort of) depth-first order. This has several advantages:

- Memory requirements: DFS needs essentially $O(\text{depth})$.
  BFS needs to store a level of the tree (often $\Omega(\text{nodes})$).
- Integer solutions are often deep in the tree. We need them to prune; earlier is better. When aborting the search, e.g., due to a timeout, we want to have a good solution.
- Warm Starting: In DFS, the next problem we solve is very often only one added constraint away from the previously solved one. We can hope that we can use the previous optimal basis as a starting point for solving the next problem with much fewer iterations than starting from scratch. Let's see how that could be done!

DUAL SIMPLEX WARM STARTING

Consider our original problem P_0 and its related problem P_2 (P_0 with $x_1 \geq 2$).
Optimal dictionary for P_0:

$$\begin{array}{rcccc}
\zeta = & \frac{205}{3} - & \frac{5}{3}w_1 - & \frac{1}{3}w_2 \\
\hline
x_1 = & \frac{5}{3} - & \frac{1}{3}w_1 + & \frac{7}{3}w_2 \\
x_2 = & \frac{10}{3} + & \frac{1}{3}w_1 - & \frac{10}{3}w_2
\end{array}$$

What happens when we add $x_1 \geq 2$?

Dual Simplex Warm Starting

Consider our original problem P_0 and its related problem P_2 (P_0 with $x_1 \geq 2$).
Optimal dictionary for P_0:

$$\begin{array}{rrrr}
\zeta = & \frac{205}{3} - & \frac{5}{3}w_1 - & \frac{1}{3}w_2 \\
\hline
x_1 = & \frac{5}{3} - & \frac{1}{3}w_1 + & \frac{7}{3}w_2 \\
x_2 = & \frac{10}{3} + & \frac{1}{3}w_1 - & \frac{10}{3}w_2
\end{array}$$

What happens when we add $x_1 \geq 2$? We get a slack variable $g_1 = x_1 - 2 = -1/3 - w_1/3 + 7w_2/3$.

DUAL SIMPLEX WARM STARTING

Consider our original problem P_0 and its related problem P_2 (P_0 with $x_1 \geq 2$).
Optimal dictionary for P_0:

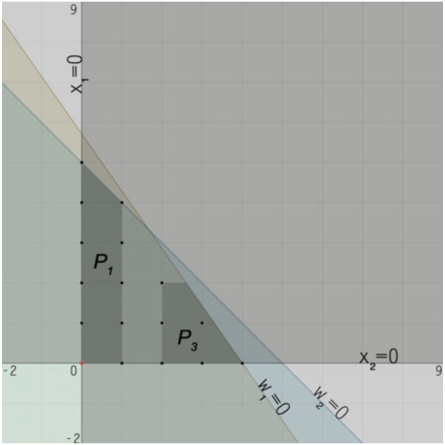
$$\begin{array}{rrrr}
\zeta = & \frac{205}{3} - & \frac{5}{3}w_1 - & \frac{1}{3}w_2 \\
\hline
x_1 = & \frac{5}{3} - & \frac{1}{3}w_1 + & \frac{7}{3}w_2 \\
x_2 = & \frac{10}{3} + & \frac{1}{3}w_1 - & \frac{10}{3}w_2
\end{array}$$

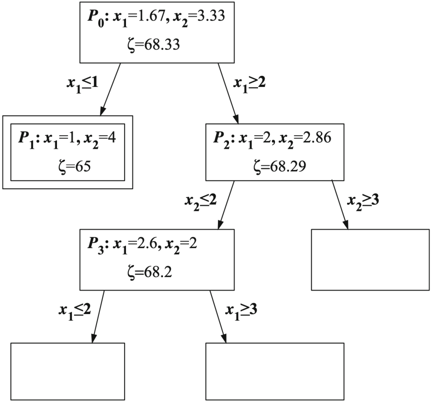
What happens when we add $x_1 \geq 2$? We get a slack variable $g_1 = x_1 - 2 = -1/3 - w_1/3 + 7w_2/3$. We can add that variable as basic. That makes the new dictionary primally infeasible. It is dually feasible however, so we can use dual Simplex.

$$\begin{array}{rrrr}
\zeta = & \frac{205}{3} - & \frac{5}{3}w_1 - & \frac{1}{3}w_2 \\
\hline
x_1 = & \frac{5}{3} - & \frac{1}{3}w_1 + & \frac{7}{3}w_2 \\
x_2 = & \frac{10}{3} + & \frac{1}{3}w_1 - & \frac{10}{3}w_2 \\
g_1 = & -\frac{1}{3} - & \frac{1}{3}w_1 + & \frac{7}{3}w_2
\end{array}$$

Continuing Our Example

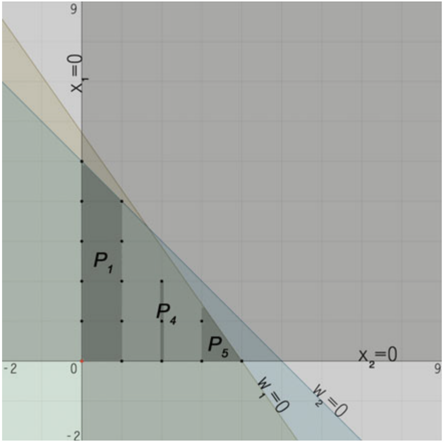
After exploring P_3:

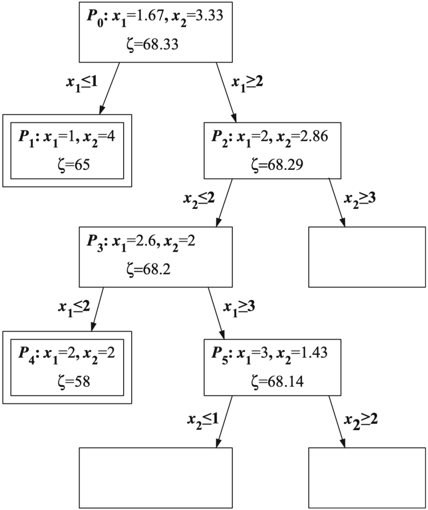




CONTINUING OUR EXAMPLE

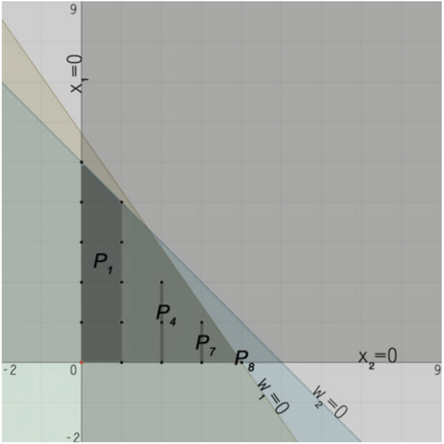
After exploring P_4, P_5:

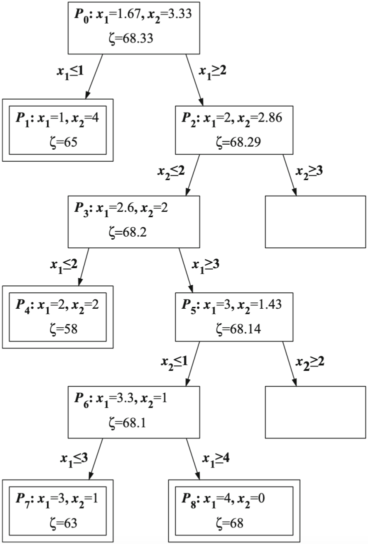




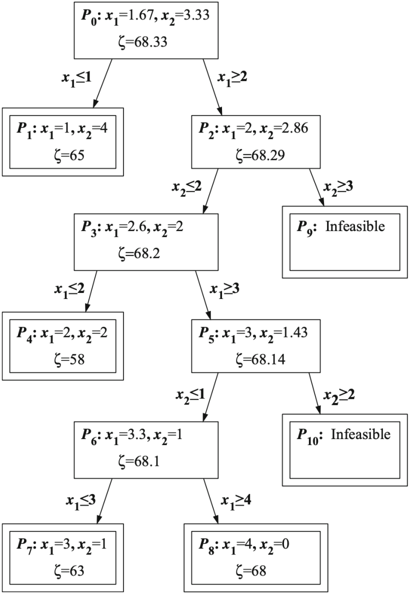
CONTINUING OUR EXAMPLE

After exploring P_6, P_7, P_8:





Final Search Tree



Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
  - Take the next P_i out of Q.

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Compute the optimal solution x^i with value ζ^i for the LP relaxation of P_i.

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Compute the optimal solution x^i with value ζ^i for the LP relaxation of P_i.
  - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
    - Take the next P_i out of Q.
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of P_i.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
    - If x^i is integral, update $B = x^i, v_B = \zeta^i$, and continue with next P_i.

BRANCH & BOUND ALGORITHM

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Compute the optimal solution x^i with value ζ^i for the LP relaxation of P_i.
  - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
  - If x^i is integral, update $B = x^i, v_B = \zeta^i$, and continue with next P_i.
  - Select non-integral variable x with value θ from x^i.

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Compute the optimal solution x^i with value ζ^i for the LP relaxation of P_i.
  - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
  - If x^i is integral, update $B = x^i, v_B = \zeta^i$, and continue with next P_i.
  - Select non-integral variable x with value θ from x^i.
  - Add $P_i \cup \{x \leq \lfloor\theta\rfloor\}$ and $P_i \cup \{x \geq \lceil\theta\rceil\}$ to Q.

Branch & Bound Algorithm

We maintain a stack (or (priority) queue) Q of unexplored search nodes, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$).
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Compute the optimal solution x^i with value ζ^i for the LP relaxation of P_i.
  - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
  - If x^i is integral, update $B = x^i, v_B = \zeta^i$, and continue with next P_i.
  - Select non-integral variable x with value θ from x^i.
  - Add $P_i \cup \{x \leq \lfloor\theta\rfloor\}$ and $P_i \cup \{x \geq \lceil\theta\rceil\}$ to Q.
- If $B = \perp$, report infeasibility. Otherwise, return optimal solution B.

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

The idea is to analyze the solutions of linear relaxations, and to dynamically identify certain types of linear constraints that

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

The idea is to analyze the solutions of linear relaxations, and to dynamically identify certain types of linear constraints that

- are satisfied by all integral solutions, but

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

The idea is to analyze the solutions of linear relaxations, and to dynamically identify certain types of linear constraints that

- are satisfied by all integral solutions, but
- are not satisfied by the solution to the current linear relaxation.

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

The idea is to analyze the solutions of linear relaxations, and to dynamically identify certain types of linear constraints that

- are satisfied by all integral solutions, but
- are not satisfied by the solution to the current linear relaxation.

Such inequalities can be dynamically added to and removed from the problem (without changing the set of integral solutions). They are called *cutting planes* or simply *cuts*. They can often drastically improve the quality of the bounds given by linear relaxations, help prune nodes of the search tree and identify integral solutions earlier.

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

The idea is to analyze the solutions of linear relaxations, and to dynamically identify certain types of linear constraints that

- are satisfied by all integral solutions, but
- are not satisfied by the solution to the current linear relaxation.

Such inequalities can be dynamically added to and removed from the problem (without changing the set of integral solutions). They are called *cutting planes* or simply *cuts*. They can often drastically improve the quality of the bounds given by linear relaxations, help prune nodes of the search tree and identify integral solutions earlier.

Cuts are usually found by heuristic procedures. Modern solvers already contain a set of such procedures that have proven useful for many practical problems. Implementing such procedures efficiently and balancing the additional effort put into finding cuts against the runtime benefits they provide is an important part of engineering a good solver.

Cutting Planes

There are several ways to extend Branch & Bound, usually with the goal of making it faster, at least for many interesting and practically relevant NP-hard problems.
One extremely important extension, implemented by all serious (M)IP solvers, is the addition of *cutting planes*.

The idea is to analyze the solutions of linear relaxations, and to dynamically identify certain types of linear constraints that

- are satisfied by all integral solutions, but
- are not satisfied by the solution to the current linear relaxation.

Such inequalities can be dynamically added to and removed from the problem (without changing the set of integral solutions). They are called *cutting planes* or simply *cuts*. They can often drastically improve the quality of the bounds given by linear relaxations, help prune nodes of the search tree and identify integral solutions earlier.

Cuts are usually found by heuristic procedures. Modern solvers already contain a set of such procedures that have proven useful for many practical problems. Implementing such procedures efficiently and balancing the additional effort put into finding cuts against the runtime benefits they provide is an important part of engineering a good solver.

Furthermore, many problems allow the implementation of problem-specific cuts that are not part of general-purpose solvers. These often require additional knowledge about the problem or are too expensive or too specialized to be included in general-purpose solvers.

Gomory Cuts

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Gomory Cuts

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Consider the case where all x_j with non-zero coefficients are integer variables. Is that case rare?

Gomory Cuts

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Consider the case where all x_j with non-zero coefficients are integer variables. Is that case rare?
No! Many slack variables are integral, e.g., if all coefficients in their constraint are integral.

Gomory Cuts

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Consider the case where all x_j with non-zero coefficients are integer variables. Is that case rare?
No! Many slack variables are integral, e.g., if all coefficients in their constraint are integral.
Split into integral and fractional part:

$$\lfloor x_i^* \rfloor + (x_i^* - \lfloor x_i^* \rfloor) = x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j + \sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j$$

Gomory Cuts

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Consider the case where all x_j with non-zero coefficients are integer variables. Is that case rare?
No! Many slack variables are integral, e.g., if all coefficients in their constraint are integral.
Split into integral and fractional part:

$$\lfloor x_i^* \rfloor + (x_i^* - \lfloor x_i^* \rfloor) = x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j + \sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j$$

Separate integral (left-hand side) and fractional (right-hand side):

$$\underbrace{x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor x_i^* \rfloor}_{\in \mathbb{Z}} = \underbrace{(x_i^* - \lfloor x_i^* \rfloor)}_{<1} - \underbrace{\sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j}_{\geq 0 \text{ for } x \geq 0}$$

Gomory Cuts

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Consider the case where all x_j with non-zero coefficients are integer variables. Is that case rare?
No! Many slack variables are integral, e.g., if all coefficients in their constraint are integral.
Split into integral and fractional part:

$$\lfloor x_i^* \rfloor + (x_i^* - \lfloor x_i^* \rfloor) = x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j + \sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j$$

Separate integral (left-hand side) and fractional (right-hand side):

$$\underbrace{x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor x_i^* \rfloor}_{\in \mathbb{Z}} = \underbrace{(x_i^* - \lfloor x_i^* \rfloor)}_{<1} - \underbrace{\sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j}_{\geq 0 \text{ for } x \geq 0}$$

Therefore, $x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor x_i^* \rfloor \leq 0 \Leftrightarrow x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor x_i^* \rfloor$ holds for all integer solutions.

GOMORY CUTS

A very important family of cuts are the so-called *Gomory cuts*.
Consider an (optimal) basic solution to a linear relaxation. In dictionary form, we have m equations of the form (which are valid constraints)

$$x_i = x_i^* - \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j \Leftrightarrow x_i^* = x_i + \sum_{j \in \mathcal{N}} \bar{a}_{ij} x_j$$

Consider the case where all x_j with non-zero coefficients are integer variables. Is that case rare?
No! Many slack variables are integral, e.g., if all coefficients in their constraint are integral.
Split into integral and fractional part:

$$\lfloor x_i^* \rfloor + (x_i^* - \lfloor x_i^* \rfloor) = x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j + \sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j$$

Separate integral (left-hand side) and fractional (right-hand side):

$$\underbrace{x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor x_i^* \rfloor}_{\in \mathbb{Z}} = \underbrace{(x_i^* - \lfloor x_i^* \rfloor)}_{<1} - \underbrace{\sum_{j \in \mathcal{N}} (\bar{a}_{ij} - \lfloor \bar{a}_{ij} \rfloor) x_j}_{\geq 0 \text{ for } x \geq 0}$$

Therefore, $x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j - \lfloor x_i^* \rfloor \leq 0 \Leftrightarrow x_i + \sum_{j \in \mathcal{N}} \lfloor \bar{a}_{ij} \rfloor x_j \leq \lfloor x_i^* \rfloor$ holds for all integer solutions.
This constraint is always violated in the current basic solution if $x_i^* \notin \mathbb{Z}$. Why?

Gomory Cut Example

With a given optimal dictionary, equivalent cuts (to the general scheme introduced before) can be found like in the following example.

$$
\begin{array}{rrrr}
\zeta = & \frac{179}{3} - & \frac{7}{27}w_1 - & \frac{73}{54}w_2 \\
\hline
x_1 = & \frac{11}{3} - & \frac{5}{54}w_1 - & \frac{1}{54}w_2 \\
x_2 = & \frac{7}{3} + & \frac{1}{27}w_1 + & \frac{5}{54}w_2 \\
w_3 = & 13 - & \frac{5}{9}w_1 - & \frac{8}{9}w_2
\end{array}
$$

Gomory Cut Example

With a given optimal dictionary, equivalent cuts (to the general scheme introduced before) can be found like in the following example.

$$
\begin{array}{rrrr}
\zeta = & \frac{179}{3} - & \frac{7}{27}w_1 - & \frac{73}{54}w_2 \\
\hline
x_1 = & \frac{11}{3} - & \frac{5}{54}w_1 - & \frac{1}{54}w_2 \\
x_2 = & \frac{7}{3} + & \frac{1}{27}w_1 + & \frac{5}{54}w_2 \\
w_3 = & 13 - & \frac{5}{9}w_1 - & \frac{8}{9}w_2
\end{array}
$$

x_1 is not integral. Reorganize equation so all variables are on one side:

$$x_1 + \frac{5}{54}w_1 + \frac{1}{54}w_2 = \frac{11}{3}.$$

GOMORY CUT EXAMPLE

With a given optimal dictionary, equivalent cuts (to the general scheme introduced before) can be found like in the following example.

$$
\begin{array}{rrrr}
\zeta = & \frac{179}{3} - & \frac{7}{27}w_1 - & \frac{73}{54}w_2 \\
\hline
x_1 = & \frac{11}{3} - & \frac{5}{54}w_1 - & \frac{1}{54}w_2 \\
x_2 = & \frac{7}{3} + & \frac{1}{27}w_1 + & \frac{5}{54}w_2 \\
w_3 = & 13 - & \frac{5}{9}w_1 - & \frac{8}{9}w_2
\end{array}
$$

x_1 is not integral. Reorganize equation so all variables are on one side:

$$x_1 + \frac{5}{54}w_1 + \frac{1}{54}w_2 = \frac{11}{3}.$$

Rounding the left-hand side coefficients makes the left-hand side smaller and integral:

$$x_1 + 0w_1 + 0w_2 \leq \lfloor {}^{11}/_{3} \rfloor = 3 \Rightarrow x_1 \leq 3.$$

Gomory Cut Example Continued

$$
\begin{array}{rcrcrcr}
\zeta & = & \frac{179}{3} & - & \frac{7}{27}w_1 & - & \frac{73}{54}w_2 \\
\hline
x_1 & = & \frac{11}{3} & - & \frac{5}{54}w_1 & - & \frac{1}{54}w_2 \\
x_2 & = & \frac{7}{3} & + & \frac{1}{27}w_1 & + & \frac{5}{54}w_2 \\
w_3 & = & 13 & - & \frac{5}{9}w_1 & - & \frac{8}{9}w_2
\end{array}
$$

Gomory Cut Example Continued

$$
\begin{array}{rrrr}
\zeta = & \dfrac{179}{3} - & \dfrac{7}{27}w_1 - & \dfrac{73}{54}w_2 \\
\hline
x_1 = & \dfrac{11}{3} - & \dfrac{5}{54}w_1 - & \dfrac{1}{54}w_2 \\
x_2 = & \dfrac{7}{3} + & \dfrac{1}{27}w_1 + & \dfrac{5}{54}w_2 \\
w_3 = & 13 - & \dfrac{5}{9}w_1 - & \dfrac{8}{9}w_2
\end{array}
$$

Adding $x_1 \leq 3$ adds a (basic, integral!) slack variable $w_4 = 3 - x_1 = 3 - \dfrac{11}{3} + \dfrac{5}{54}w_1 + \dfrac{1}{54}w_2$:

Gomory Cut Example Continued

$$\begin{array}{rcrcrcr}
\zeta & = & \frac{179}{3} & - & \frac{7}{27}w_1 & - & \frac{73}{54}w_2 \\
\hline
x_1 & = & \frac{11}{3} & - & \frac{5}{54}w_1 & - & \frac{1}{54}w_2 \\
x_2 & = & \frac{7}{3} & + & \frac{1}{27}w_1 & + & \frac{5}{54}w_2 \\
w_3 & = & 13 & - & \frac{5}{9}w_1 & - & \frac{8}{9}w_2
\end{array}$$

Adding $x_1 \leq 3$ adds a (basic, integral!) slack variable $w_4 = 3 - x_1 = 3 - \frac{11}{3} + \frac{5}{54}w_1 + \frac{1}{54}w_2$:

$$\begin{array}{rcrcrcr}
\zeta & = & \frac{179}{3} & - & \frac{7}{27}w_1 & - & \frac{73}{54}w_2 \\
\hline
x_1 & = & \frac{11}{3} & - & \frac{5}{54}w_1 & - & \frac{1}{54}w_2 \\
x_2 & = & \frac{7}{3} & + & \frac{1}{27}w_1 & + & \frac{5}{54}w_2 \\
w_3 & = & 13 & - & \frac{5}{9}w_1 & - & \frac{8}{9}w_2 \\
w_4 & = & -\frac{2}{3} & + & \frac{5}{54}w_1 & + & \frac{1}{54}w_2
\end{array}$$

Gomory Cut Example Continued

$$
\begin{array}{rcrcrcr}
\zeta & = & \frac{179}{3} & - & \frac{7}{27}w_1 & - & \frac{73}{54}w_2 \\
\hline
x_1 & = & \frac{11}{3} & - & \frac{5}{54}w_1 & - & \frac{1}{54}w_2 \\
x_2 & = & \frac{7}{3} & + & \frac{1}{27}w_1 & + & \frac{5}{54}w_2 \\
w_3 & = & 13 & - & \frac{5}{9}w_1 & - & \frac{8}{9}w_2
\end{array}
$$

Adding $x_1 \leq 3$ adds a (basic, integral!) slack variable $w_4 = 3 - x_1 = 3 - \frac{11}{3} + \frac{5}{54}w_1 + \frac{1}{54}w_2$:

$$
\begin{array}{rcrcrcr}
\zeta & = & \frac{179}{3} & - & \frac{7}{27}w_1 & - & \frac{73}{54}w_2 \\
\hline
x_1 & = & \frac{11}{3} & - & \frac{5}{54}w_1 & - & \frac{1}{54}w_2 \\
x_2 & = & \frac{7}{3} & + & \frac{1}{27}w_1 & + & \frac{5}{54}w_2 \\
w_3 & = & 13 & - & \frac{5}{9}w_1 & - & \frac{8}{9}w_2 \\
w_4 & = & -\frac{2}{3} & + & \frac{5}{54}w_1 & + & \frac{1}{54}w_2
\end{array}
$$

We can continue with dual Simplex.

Gomory Cut Example Continued

After one dual Simplex pivot:

$$
\begin{array}{rcccc}
\zeta = & \frac{289}{5} - & \frac{14}{5} w_4 - & \frac{13}{10} w_2 \\
\hline
x_1 = & 3 - & w_4 & \\
x_2 = & \frac{13}{5} + & \frac{2}{5} w_4 + & \frac{23}{270} w_2 \\
w_3 = & 9 - & 6 w_4 - & \frac{7}{9} w_2 \\
w_1 = & \frac{36}{5} + & \frac{54}{5} w_4 - & \frac{1}{5} w_2
\end{array}
$$

GOMORY CUT EXAMPLE CONTINUED

After one dual Simplex pivot:

$$
\begin{array}{rrrr}
\zeta = & \frac{289}{5} - & \frac{14}{5}w_4 - & \frac{13}{10}w_2 \\
\hline
x_1 = & 3 - & w_4 & \\
x_2 = & \frac{13}{5} + & \frac{2}{5}w_4 + & \frac{23}{270}w_2 \\
w_3 = & 9 - & 6w_4 - & \frac{7}{9}w_2 \\
w_1 = & \frac{36}{5} + & \frac{54}{5}w_4 - & \frac{1}{5}w_2
\end{array}
$$

Gomory cut on $x_2 - \frac{2}{5}w_4 - \frac{23}{270}w_2 = \frac{13}{5}$: $\quad x_2 - w_4 - w_2 \leq 2.$

Gomory Cut Example Continued

After one dual Simplex pivot:

$$
\begin{array}{rrrr}
\zeta = & \frac{289}{5} - & \frac{14}{5} w_4 - & \frac{13}{10} w_2 \\
\hline
x_1 = & 3 - & w_4 & \\
x_2 = & \frac{13}{5} + & \frac{2}{5} w_4 + & \frac{23}{270} w_2 \\
w_3 = & 9 - & 6 w_4 - & \frac{7}{9} w_2 \\
w_1 = & \frac{36}{5} + & \frac{54}{5} w_4 - & \frac{1}{5} w_2
\end{array}
$$

Gomory cut on $x_2 - \frac{2}{5} w_4 - \frac{23}{270} w_2 = \frac{13}{5}$: $\quad x_2 - w_4 - w_2 \leq 2.$

$$
\begin{array}{rrrr}
\zeta = & \frac{289}{5} - & \frac{14}{5} w_4 - & \frac{13}{10} w_2 \\
\hline
x_1 = & 3 - & w_4 & \\
x_2 = & \frac{13}{5} + & \frac{2}{5} w_4 + & \frac{23}{270} w_2 \\
w_3 = & 9 - & 6 w_4 - & \frac{7}{9} w_2 \\
w_1 = & \frac{36}{5} + & \frac{54}{5} w_4 - & \frac{1}{5} w_2 \\
w_5 = & -\frac{3}{5} + & \frac{3}{5} w_4 + & \frac{247}{270} w_2
\end{array}
$$

Gomory Cuts

We could continue this for a while.
Recall that a *cutting plane* must, in general, have two properties:

(1) it must *cut off* the current LP relaxation solution, i.e., be violated by it,

(2) it must be satisfied by all integral solutions of the original (M)IP.

Gomory Cuts

We could continue this for a while.
Recall that a *cutting plane* must, in general, have two properties:

(1) it must *cut off* the current LP relaxation solution, i.e., be violated by it,

(2) it must be satisfied by all integral solutions of the original (M)IP.

Because we only add constraints that are satisfied by *all* integral solutions:

- If we reach an integral solution of the LP relaxation purely by adding cuts, it is optimal.
- If we reach an infeasible LP, there is no feasible integral solution.

Gomory Cuts

We could continue this for a while.
Recall that a *cutting plane* must, in general, have two properties:

(1) it must *cut off* the current LP relaxation solution, i.e., be violated by it,

(2) it must be satisfied by all integral solutions of the original (M)IP.

Because we only add constraints that are satisfied by *all* integral solutions:

- If we reach an integral solution of the LP relaxation purely by adding cuts, it is optimal.
- If we reach an infeasible LP, there is no feasible integral solution.

Because we never generate constraints that are satisfied by the current LP relaxation solution:

- We never get *stuck*, i.e., the solution will continue to change.
- One can actually prove that, for purely integral linear programs, this will eventually terminate (in theory).

Gomory Cuts

We could continue this for a while.
Recall that a *cutting plane* must, in general, have two properties:

(1) it must *cut off* the current LP relaxation solution, i.e., be violated by it,

(2) it must be satisfied by all integral solutions of the original (M)IP.

Because we only add constraints that are satisfied by *all* integral solutions:

- If we reach an integral solution of the LP relaxation purely by adding cuts, it is optimal.
- If we reach an infeasible LP, there is no feasible integral solution.

Because we never generate constraints that are satisfied by the current LP relaxation solution:

- We never get *stuck*, i.e., the solution will continue to change.
- One can actually prove that, for purely integral linear programs, this will eventually terminate (in theory).

However, this does not really work in practice: it is both inefficient and introduces terrible numerical problems.

GOMORY CUTS

We could continue this for a while.
Recall that a *cutting plane* must, in general, have two properties:

(1) it must *cut off* the current LP relaxation solution, i.e., be violated by it,

(2) it must be satisfied by all integral solutions of the original (M)IP.

Because we only add constraints that are satisfied by *all* integral solutions:

- If we reach an integral solution of the LP relaxation purely by adding cuts, it is optimal.
- If we reach an infeasible LP, there is no feasible integral solution.

Because we never generate constraints that are satisfied by the current LP relaxation solution:

- We never get *stuck*, i.e., the solution will continue to change.
- One can actually prove that, for purely integral linear programs, this will eventually terminate (in theory).

However, this does not really work in practice: it is both inefficient and introduces terrible numerical problems.
What does work in practice? Combining cutting planes and Branch & Bound into an algorithmic paradigm called Branch & Cut. This is the basis of all competitive modern MIP solvers.

Branch & Cut

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

Branch & Cut

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:

Branch & Cut

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.

BRANCH & CUT

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):

BRANCH & CUT

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.

Branch & Cut

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.

Branch & Cut

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
    - If x^i is integral, update $B = x^i$, $v_B = \zeta^i$, and continue with next P_i.

BRANCH & CUT

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
    - If x^i is integral, update $B = x^i$, $v_B = \zeta^i$, and continue with next P_i.
    - Attempt to find a good new cutting plane (a, b) with $a^T x^i > b$.

BRANCH & CUT

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
    - If x^i is integral, update $B = x^i$, $v_B = \zeta^i$, and continue with next P_i.
    - Attempt to find a good new cutting plane (a, b) with $a^T x^i > b$.
    - If successful, add (a, b) to C. Otherwise, stop.

BRANCH & CUT

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
    - If x^i is integral, update $B = x^i$, $v_B = \zeta^i$, and continue with next P_i.
    - Attempt to find a good new cutting plane (a, b) with $a^T x^i > b$.
    - If successful, add (a, b) to C. Otherwise, stop.
  - Select non-integral variable x with value θ from x^i.
  - Add $P_i \cup \{x \leq \lfloor\theta\rfloor\}$ and $P_i \cup \{x \geq \lceil\theta\rceil\}$ to Q.

BRANCH & CUT

Extension of Branch & Bound: We maintain a stack (or (priority) queue) Q of unexplored search nodes, a set of cutting planes C, and a best current solution B with value v_B and assume maximization (minimization is analogous).

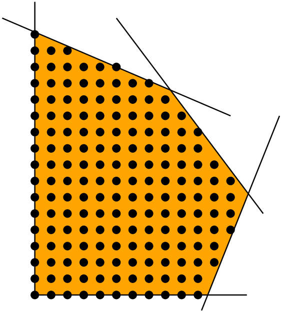
- Initialize Q with P_0, the original problem.
- Initialize B, v_B with the best known solution (or set $v_B = -\infty, B = \perp$), set $C = \emptyset$.
- While Q is non-empty:
  - Take the next P_i out of Q.
  - Repeat (as long as it seems promising to do so):
    - Compute the optimal solution x^i with value ζ^i for the LP relaxation of $P_i \cup C$.
    - If P_i is infeasible or $\zeta^i \leq v_B$, continue with next P_i.
    - If x^i is integral, update $B = x^i$, $v_B = \zeta^i$, and continue with next P_i.
    - Attempt to find a good new cutting plane (a, b) with $a^T x^i > b$.
    - If successful, add (a, b) to C. Otherwise, stop.
  - Select non-integral variable x with value θ from x^i.
  - Add $P_i \cup \{x \leq \lfloor\theta\rfloor\}$ and $P_i \cup \{x \geq \lceil\theta\rceil\}$ to Q.
- If $B = \perp$, report infeasibility. Otherwise, return optimal solution B.

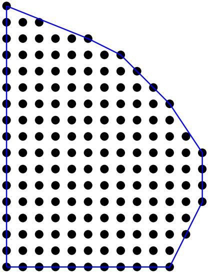
The Convex Hull

Geometrically: constraints of LP with n variables define n-dimensional polyhedron which contains all feasible points.
Basic solutions correspond to vertices (corners) of this polyhedron.
How about integer linear programs?

The Convex Hull

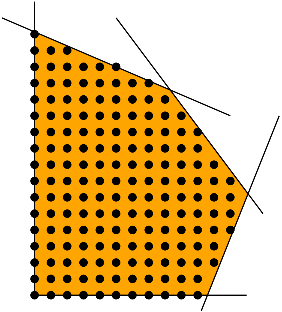
Geometrically: constraints of LP with n variables define n-dimensional polyhedron which contains all feasible points.
Basic solutions correspond to vertices (corners) of this polyhedron.
How about integer linear programs?

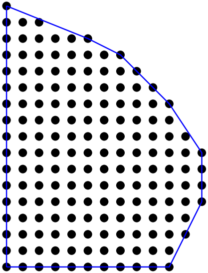




THE CONVEX HULL

Geometrically: constraints of LP with n variables define n-dimensional polyhedron which contains all feasible points.
Basic solutions correspond to vertices (corners) of this polyhedron.
How about integer linear programs?

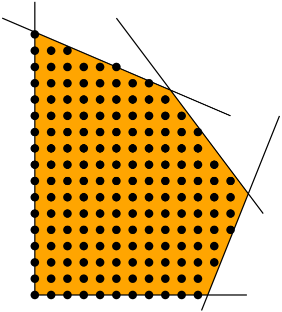


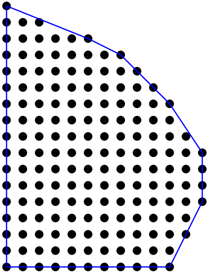


If we had a set of linear constraints that defines the convex hull of the integral points, Simplex would give us an integral solution (for any objective function), and thus solve the integer program! However, the number of constraints is too high and they are not usually easy to find!

The Convex Hull

Geometrically: constraints of LP with n variables define n-dimensional polyhedron which contains all feasible points.
Basic solutions correspond to vertices (corners) of this polyhedron.
How about integer linear programs?





If we had a set of linear constraints that defines the convex hull of the integral points, Simplex would give us an integral solution (for any objective function), and thus solve the integer program! However, the number of constraints is too high and they are not usually easy to find!

The strongest cutting planes we can add define so-called *facets* of the convex hull. Can we hope to always do that?

Separation Problem

Given a polyhedron $\mathcal{Q} \subseteq \mathbb{R}^n$ and a vector $\tilde{x} \in \mathbb{R}^n$, determine whether $\tilde{x} \in \mathcal{Q}$, and if not, determine a linear constraint (a, b) such that $a^T\tilde{x} > b$ and $a^T x \leq b$ for all $x \in \mathcal{Q}$.

Separation Problem

Given a polyhedron $\mathcal{Q} \subseteq \mathbb{R}^n$ and a vector $\tilde{x} \in \mathbb{R}^n$, determine whether $\tilde{x} \in \mathcal{Q}$, and if not, determine a linear constraint (a, b) such that $a^T \tilde{x} > b$ and $a^T x \leq b$ for all $x \in \mathcal{Q}$.

Usually, when we talk about this:

- $\mathcal{Q}$ is the convex hull of integral solutions, and
- $\tilde{x}$ is the solution to the linear relaxation.

Separation Problem

Given a polyhedron $\mathcal{Q} \subseteq \mathbb{R}^n$ and a vector $\tilde{x} \in \mathbb{R}^n$, determine whether $\tilde{x} \in \mathcal{Q}$, and if not, determine a linear constraint (a, b) such that $a^T\tilde{x} > b$ and $a^T x \leq b$ for all $x \in \mathcal{Q}$.

Usually, when we talk about this:

- $\mathcal{Q}$ is the convex hull of integral solutions, and
- $\tilde{x}$ is the solution to the linear relaxation.

The *Ellipsoid Method*, a method to solve LP in polynomial time, can optimize over $\mathcal{Q}$ without an explicit description of $\mathcal{Q}$, based solely on a *separation oracle*.

Separation Problem

Given a polyhedron $\mathcal{Q} \subseteq \mathbb{R}^n$ and a vector $\tilde{x} \in \mathbb{R}^n$, determine whether $\tilde{x} \in \mathcal{Q}$, and if not, determine a linear constraint (a, b) such that $a^T\tilde{x} > b$ and $a^T x \leq b$ for all $x \in \mathcal{Q}$.

Usually, when we talk about this:

- $\mathcal{Q}$ is the convex hull of integral solutions, and
- $\tilde{x}$ is the solution to the linear relaxation.

The *Ellipsoid Method*, a method to solve LP in polynomial time, can optimize over $\mathcal{Q}$ without an explicit description of $\mathcal{Q}$, based solely on a *separation oracle*.

The method has polynomial runtime in n, $\log T$ and $\log ||c||$ if the separation oracle has polynomial runtime, where

- n is the number of variables, i.e., the maximum dimension of $\mathcal{Q}$,
- T is the maximum numerator or denominator in any coordinate of any extreme point of $\mathcal{Q}$,
- $c \in \mathbb{R}^n$ is the objective coefficient vector.

Separation Problem

Given a polyhedron $\mathcal{Q} \subseteq \mathbb{R}^n$ and a vector $\tilde{x} \in \mathbb{R}^n$, determine whether $\tilde{x} \in \mathcal{Q}$, and if not, determine a linear constraint (a, b) such that $a^T\tilde{x} > b$ and $a^T x \leq b$ for all $x \in \mathcal{Q}$.

Usually, when we talk about this:

- $\mathcal{Q}$ is the convex hull of integral solutions, and
- $\tilde{x}$ is the solution to the linear relaxation.

The *Ellipsoid Method*, a method to solve LP in polynomial time, can optimize over $\mathcal{Q}$ without an explicit description of $\mathcal{Q}$, based solely on a *separation oracle*.

The method has polynomial runtime in n, $\log T$ and $\log ||c||$ if the separation oracle has polynomial runtime, where

- n is the number of variables, i.e., the maximum dimension of $\mathcal{Q}$,
- T is the maximum numerator or denominator in any coordinate of any extreme point of $\mathcal{Q}$,
- $c \in \mathbb{R}^n$ is the objective coefficient vector.

Very important corollary: If we can solve the separation problem in polynomial time, we can solve linear optimization problems over $\mathcal{Q}$ in polynomial time!

The Template Paradigm

Because the separation problem in general seems hard, one usually looks at restricted/simplified versions of it to find cutting planes. Typically, either:

- the solution has special properties that allow finding cuts (e.g., Gomory cuts which need basic solutions),
- the cuts have a special format so they can be efficiently found,
- the separation problem is solved heuristically.

The Template Paradigm

Because the separation problem in general seems hard, one usually looks at restricted/simplified versions of it to find cutting planes. Typically, either:

- the solution has special properties that allow finding cuts (e.g., Gomory cuts which need basic solutions),
- the cuts have a special format so they can be efficiently found,
- the separation problem is solved heuristically.

There are a lot of special *cut templates* for which either separation can be done efficiently, or for which there are good heuristics.

THE TEMPLATE PARADIGM

Because the separation problem in general seems hard, one usually looks at restricted/simplified versions of it to find cutting planes. Typically, either:

- the solution has special properties that allow finding cuts (e.g., Gomory cuts which need basic solutions),
- the cuts have a special format so they can be efficiently found,
- the separation problem is solved heuristically.

There are a lot of special *cut templates* for which either separation can be done efficiently, or for which there are good heuristics.

Many of these are already implemented in modern MIP solvers. There are usually parameters to tune how aggressively they should be generated and applied.

The Template Paradigm

Because the separation problem in general seems hard, one usually looks at restricted/simplified versions of it to find cutting planes. Typically, either:

- the solution has special properties that allow finding cuts (e.g., Gomory cuts which need basic solutions),
- the cuts have a special format so they can be efficiently found,
- the separation problem is solved heuristically.

There are a lot of special *cut templates* for which either separation can be done efficiently, or for which there are good heuristics.

Many of these are already implemented in modern MIP solvers. There are usually parameters to tune how aggressively they should be generated and applied.

Knowledge of the concrete problem can often yield more cutting planes and can help identify them easier; sometimes, it can pay off to implement this domain knowledge and add additional constraints during or before the solve.

CLIQUE CUTS

Suppose we want to solve a program with some $\{0,1\}$-variables. Let $x, y, z \in \{0,1\}$ be such variables.

Suppose we have the constraints

$$x + y \leq 1, \text{ and}$$

$$x + 2y + z \geq 2.$$

Clique Cuts

Suppose we want to solve a program with some $\{0,1\}$-variables. Let $x, y, z \in \{0,1\}$ be such variables.

Suppose we have the constraints

$$x + y \leq 1, \text{ and}$$

$$x + 2y + z \geq 2.$$

What happens when we set $x = 1$? What happens when we set $x = 0$?

Clique Cuts

Suppose we want to solve a program with some $\{0,1\}$-variables. Let $x, y, z \in \{0,1\}$ be such variables.

Suppose we have the constraints

$$x + y \leq 1, \text{ and}$$

$$x + 2y + z \geq 2.$$

What happens when we set $x = 1$? What happens when we set $x = 0$?

$x = 1$: We must set $y = 0$ and thus also $z = 1$!

$x = 0$: We must set $y = 1$!

Clique Cuts

Suppose we want to solve a program with some $\{0,1\}$-variables. Let $x, y, z \in \{0,1\}$ be such variables.

Suppose we have the constraints

$$x + y \leq 1, \text{ and}$$

$$x + 2y + z \geq 2.$$

What happens when we set $x = 1$? What happens when we set $x = 0$?

$x = 1$: We must set $y = 0$ and thus also $z = 1$!

$x = 0$: We must set $y = 1$!

Making inferences such as these from setting some variables is also called *propagation*. It is crucial for SAT and CP solver performance, but a bit less so for integer programs; it is still usually built into MIP solvers (when branching).

Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0, 1\}$-variable x_i and an edge between settings that are contradictory.

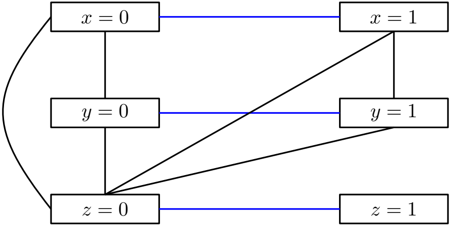
Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0,1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0,1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



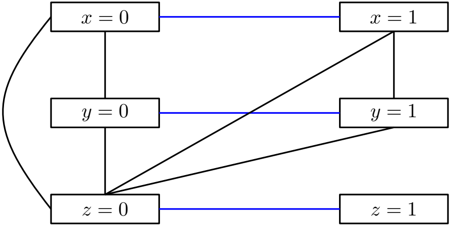
Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0, 1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



- What can we do with a clique C in G (without the blue edges)?

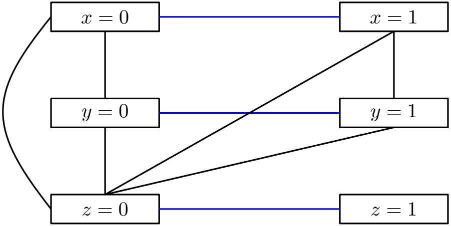
Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0, 1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



- What can we do with a clique C in G (without the blue edges)?
- The variable assignments in C are *mutually exclusive*.

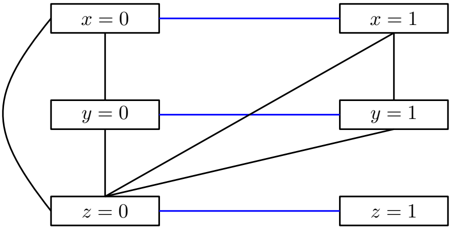
CLIQUE CUTS — PROBING

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0,1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0,1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



- What can we do with a clique C in G (without the blue edges)?
- The variable assignments in C are *mutually exclusive*.
- At most one of them can be simultaneously true!

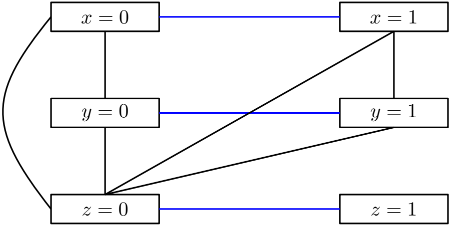
CLIQUE CUTS — PROBING

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0,1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0,1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



- What can we do with a clique C in G (without the blue edges)?
- The variable assignments in C are *mutually exclusive*.
- At most one of them can be simultaneously true!
- $(1-x)+(1-y)+(1-z) \leq 1 \Leftrightarrow x+y+z \geq 2$

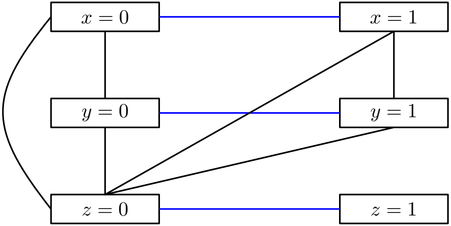
Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0, 1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



- What can we do with a clique C in G (without the blue edges)?
- The variable assignments in C are *mutually exclusive*.
- At most one of them can be simultaneously true!
- $(1 - x) + (1 - y) + (1 - z) \leq 1 \Leftrightarrow x + y + z \geq 2$
- $x + y + (1 - z) \leq 1 \Leftrightarrow z \geq x + y$

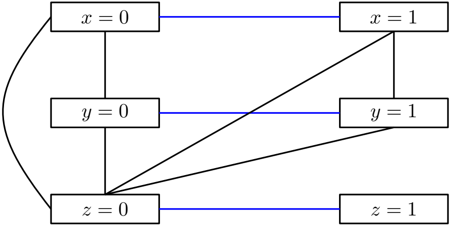
Clique Cuts — Probing

Before beginning to solve (during *presolve*), we can *probe* for logical implications, e.g., by setting each individual $\{0, 1\}$ variable (or some subset of them) to 0 and 1, each time performing propagation.

This can fix variables or detect infeasibility if variables have only one (or no) possible value.

We also get a *conflict graph* G with two vertices $x_i = 0$, $x_i = 1$ for each $\{0, 1\}$-variable x_i and an edge between settings that are contradictory.

$$x + y \leq 1$$
$$x + 2y + z \geq 2$$
$$x + z \geq 1$$



- What can we do with a clique C in G (without the blue edges)?
- The variable assignments in C are *mutually exclusive*.
- At most one of them can be simultaneously true!
- $(1 - x) + (1 - y) + (1 - z) \leq 1 \Leftrightarrow x + y + z \geq 2$
- $x + y + (1 - z) \leq 1 \Leftrightarrow z \geq x + y$
- Neither of those are linear combinations of the original constraints!

MORE PRESOLVE

Bounds strengthening ($\{0, 1\}x, y, z$):

$$x + 2y + 4z = 4, x, y, z \in \{0, 1\} \Rightarrow z \geq \frac{1}{4}(4 - 2 - 1) \Rightarrow z = 1, x = y = 0.$$

GCD reduction (pure integer x, y, z):

$3x + 6y + 9z \leq 11$, divide by 3 and round: $x + 2y + 3z \leq 3$

Coefficient reduction (binary x, y):

$2x + y \geq 1$: have a slack of ≥ 1 for $x = 1 \Rightarrow x + y \geq 1$

Much, much more...

ZERO-HALF CUTS

Suppose $x_1, \ldots, x_5 \in \mathbb{Z}$ and we have the constraints

$$x_1 + x_2 + x_3 + 3x_4 + 2x_5 \leq 10,$$

$$x_1 + x_2 + 3x_3 + x_4 + 2x_5 \leq 5.$$

Zero-Half Cuts

Suppose $x_1, \ldots, x_5 \in \mathbb{Z}$ and we have the constraints

$$x_1 + x_2 + x_3 + 3x_4 + 2x_5 \leq 10,$$

$$x_1 + x_2 + 3x_3 + x_4 + 2x_5 \leq 5.$$

Adding them gives

$$2x_1 + 2x_2 + 4x_3 + 4x_4 + 4x_5 \leq 15.$$

Zero-Half Cuts

Suppose $x_1, \ldots, x_5 \in \mathbb{Z}$ and we have the constraints

$$x_1 + x_2 + x_3 + 3x_4 + 2x_5 \leq 10,$$

$$x_1 + x_2 + 3x_3 + x_4 + 2x_5 \leq 5.$$

Adding them gives

$$2x_1 + 2x_2 + 4x_3 + 4x_4 + 4x_5 \leq 15.$$

The right-hand side is *odd*, the left-hand side is *even*. Dividing by 2 and round:

$$x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 \leq 7$$

Zero-Half Cuts

Suppose $x_1, \dots, x_5 \in \mathbb{Z}$ and we have the constraints

$$x_1 + x_2 + x_3 + 3x_4 + 2x_5 \leq 10,$$

$$x_1 + x_2 + 3x_3 + x_4 + 2x_5 \leq 5.$$

Adding them gives

$$2x_1 + 2x_2 + 4x_3 + 4x_4 + 4x_5 \leq 15.$$

The right-hand side is *odd*, the left-hand side is *even*. Dividing by 2 and round:

$$x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 \leq 7$$

In general: Try to find ways to add up constraints such that all variable coefficients are even but the right-hand side is odd.

Zero-Half Cuts

Suppose $x_1, \ldots, x_5 \in \mathbb{Z}$ and we have the constraints

$$x_1 + x_2 + x_3 + 3x_4 + 2x_5 \leq 10,$$

$$x_1 + x_2 + 3x_3 + x_4 + 2x_5 \leq 5.$$

Adding them gives

$$2x_1 + 2x_2 + 4x_3 + 4x_4 + 4x_5 \leq 15.$$

The right-hand side is *odd*, the left-hand side is *even*. Dividing by 2 and round:

$$x_1 + x_2 + 2x_3 + 2x_4 + 2x_5 \leq 7$$

In general: Try to find ways to add up constraints such that all variable coefficients are even but the right-hand side is odd.

There are decent heuristics to find such cutting planes; in general, the separation problem for these cuts is NP-hard.

Cover Cuts

The previous cutting planes (except for Gomory cuts) were generated from multiple valid inequalities. Here, *valid inequality* means an inequality that is not violated by any integer feasible solution.

Cover Cuts

The previous cutting planes (except for Gomory cuts) were generated from multiple valid inequalities. Here, *valid inequality* means an inequality that is not violated by any integer feasible solution.

There are also ways to generate cuts from a single valid inequality (this is a well-researched topic). A single inequality is often called a Knapsack constraint, because Knapsack is:

$$\max c^T x$$

$$a^T x \leq z$$

Cover Cuts

The previous cutting planes (except for Gomory cuts) were generated from multiple valid inequalities. Here, *valid inequality* means an inequality that is not violated by any integer feasible solution.

There are also ways to generate cuts from a single valid inequality (this is a well-researched topic). A single inequality is often called a Knapsack constraint, because Knapsack is:

$$\max c^T x$$
$$a^T x \leq z$$

Suppose $x \in \{0,1\}^n$. It is easy to generate *minimal covers* for a Knapsack constraint: minimal sets C with

$$\sum_{i \in C} a_i x_i > z.$$

Cover Cuts

The previous cutting planes (except for Gomory cuts) were generated from multiple valid inequalities. Here, *valid inequality* means an inequality that is not violated by any integer feasible solution.

There are also ways to generate cuts from a single valid inequality (this is a well-researched topic). A single inequality is often called a Knapsack constraint, because Knapsack is:

$$\max c^T x$$

$$a^T x \leq z$$

Suppose $x \in \{0,1\}^n$. It is easy to generate *minimal covers* for a Knapsack constraint: minimal sets C with

$$\sum_{i \in C} a_i x_i > z.$$

$$\text{Cover Cut: } \sum_{i \in C} x_i \leq |C| - 1.$$

More Cutting Planes

There are many more cutting plane templates:

- There are generalizations of Gomory cuts for mixed integer programs (mixed-integer rounding (MIR) cuts).
- There are attempts for solving the separation problem for mixed-integer Knapsacks exactly, i.e., taking a single mixed-integer constraint and separating w.r.t. the convex hull $\text{conv}(\{a^T x \leq b, x \in \mathbb{Z}^k \times \mathbb{R}^\ell\})$; this requires solving mixed-integer Knapsack problems many times.
- There are cuts based on detecting network flow-type problems in the problem.
- ...

A lot of research goes into finding efficient routines to generate effective cuts.
It takes a lot of work to balance time needed for finding cuts vs. time saved by better bounds and earlier pruning. It is very hard to build a competitive MIP solver!