# Heuristics

Péter Hajnal

2025. Spring

# Exhaustive Search

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.
- For this decision-making part, it is easy to formulate an *exhaustive search* algorithm.

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.

- For this decision-making part, it is easy to formulate an *exhaustive search* algorithm. Let's consider all possible outcomes of the decisions.

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.

- For this decision-making part, it is easy to formulate an *exhaustive search* algorithm. Let's consider all possible outcomes of the decisions.

- However, these are usually exponential in the size of the input, making them impractical even for relatively small inputs.

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.
- For this decision-making part, it is easy to formulate an *exhaustive search* algorithm. Let's consider all possible outcomes of the decisions.
- However, these are usually exponential in the size of the input, making them impractical even for relatively small inputs.
- Many problems are $\mathcal{NP}$-complete, so this difficulty is to be expected.

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.
- For this decision-making part, it is easy to formulate an *exhaustive search* algorithm. Let's consider all possible outcomes of the decisions.
- However, these are usually exponential in the size of the input, making them impractical even for relatively small inputs.
- Many problems are $\mathcal{NP}$-complete, so this difficulty is to be expected. However, it may be necessary to solve such problems in practice.

# Exhaustive Search

- In combinatorial optimization problems, it is often possible to formulate a part of our algorithm as a finite number of decisions.

- For this decision-making part, it is easy to formulate an *exhaustive search* algorithm. Let's consider all possible outcomes of the decisions.

- However, these are usually exponential in the size of the input, making them impractical even for relatively small inputs.

- Many problems are $\mathcal{NP}$-complete, so this difficulty is to be expected. However, it may be necessary to solve such problems in practice.

- In such cases, we try to reduce the complete search of cases using heuristics.

# The Branch-and-Bound Scheme

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches.

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches. Describing the complete tree is too costly.

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches. Describing the complete tree is too costly.

- During the growth in each direction, we estimate the value of the objective function (**bound**).

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches. Describing the complete tree is too costly.

- During the growth in each direction, we estimate the value of the objective function (**bound**).

- These estimates sometimes allow us to exclude the possibility that the sought-after optimal location is *below* the current location.

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches. Describing the complete tree is too costly.
- During the growth in each direction, we estimate the value of the objective function (**bound**).
- These estimates sometimes allow us to exclude the possibility that the sought-after optimal location is *below* the current location.
- Thus, we do not increase the tree in certain directions.

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches. Describing the complete tree is too costly.
- During the growth in each direction, we estimate the value of the objective function (**bound**).
- These estimates sometimes allow us to exclude the possibility that the sought-after optimal location is *below* the current location.
- Thus, we do not increase the tree in certain directions. We trim large parts of it compared to the entire tree.

# The Branch-and-Bound Scheme

- The proliferation of cases is often described by the **branching** of a tree with dual branches. Describing the complete tree is too costly.

- During the growth in each direction, we estimate the value of the objective function (**bound**).

- These estimates sometimes allow us to exclude the possibility that the sought-after optimal location is *below* the current location.

- Thus, we do not increase the tree in certain directions. We trim large parts of it compared to the entire tree.

- Good heuristics can result in significant acceleration in special cases.

# Unconditional Optimization

# Unconditional Optimization

Our sample question is very simple:

| | |
|---|---|
| Minimize | $c(x)$-t |

where $c(x) : \mathbb{R}^n \to \mathbb{R}$.

# Unconditional Optimization

Our sample question is very simple:

| Minimize | $c(x)$-t |
|---|---|

where $c(x) : \mathbb{R}^n \to \mathbb{R}$.

• Our task is to determine or approximate the optimal value $p^*$ and the optimal location $x^*$.

# Unconditional Optimization

Our sample question is very simple:

| | |
|---|---|
| Minimize | $c(x)$-t |

where $c(x) : \mathbb{R}^n \to \mathbb{R}$.

- Our task is to determine or approximate the optimal value $p^*$ and the optimal location $x^*$.
- If we have no knowledge of $c$, then our situation is hopeless.

# Unconditional Optimization

Our sample question is very simple:

| Minimize | $c(x)$-t |
|---|---|

where $c(x) : \mathbb{R}^n \to \mathbb{R}$.

- Our task is to determine or approximate the optimal value $p^*$ and the optimal location $x^*$.

- If we have no knowledge of $c$, then our situation is hopeless.

**Definition: Rectangle**

$$[a_1, b_1] \times \ldots \times [a_n, b_n] \subset \mathbb{R}^n$$

shaped point sets — where $n$ is the dimension and $-\infty < a_i < b_i < \infty \quad i = 1, \ldots, n$ — are called rectangles.

# Desirable Properties of Nice Functions

# Desirable Properties of Nice Functions

1) Given a $T_0 = [a_1, b_1] \times \ldots \times [a_n, b_n]$ rectangle where $x^* \in T_0$,

# Desirable Properties of Nice Functions

1) Given a $T_0 = [a_1, b_1] \times \ldots \times [a_n, b_n]$ rectangle where $x^* \in T_0$,
2) For every $T$ rectangle, there are computable lower and upper bounds $a_T, f_T$ for $\min_{x \in T} c(x)$. Additionally, these bounds satisfy:

# Desirable Properties of Nice Functions

1) Given a $T_0 = [a_1, b_1] \times \ldots \times [a_n, b_n]$ rectangle where $x^* \in T_0$,
2) For every $T$ rectangle, there are computable lower and upper bounds $a_T, f_T$ for $\min_{x \in T} c(x)$. Additionally, these bounds satisfy:
   (A) If we divide a suitable side of $T$ into two halves:

   $$T = T_1 \mathring{\cup} T_2, \text{ i.e., for suitable } i \text{ and } v$$
   $$T_1 = [a_1, b_1] \times \ldots \times [a_{i-1}, b_{i-1}] \times [a_i, v] \times [a_{i+1}, b_{i+1}] \times \ldots \times [a_n, b_n],$$
   $$T_2 = [a_1, b_1] \times \ldots \times [a_{i-1}, b_{i-1}] \times [v, b_i] \times [a_{i+1}, b_{i+1}] \times \ldots \times [a_n, b_n],$$

   then $a_T \leq a_{T_1}, a_{T_2}$ and $f_{T_1}, f_{T_2} \leq f_T$ hold.

# Desirable Properties of Nice Functions

1) Given a $T_0 = [a_1, b_1] \times \ldots \times [a_n, b_n]$ rectangle where $x^* \in T_0$,
2) For every $T$ rectangle, there are computable lower and upper bounds $a_T, f_T$ for $\min_{x \in T} c(x)$. Additionally, these bounds satisfy:
   (A) If we divide a suitable side of $T$ into two halves:
   $T = T_1 \mathring{\cup} T_2$, i.e., for suitable $i$ and $v$
   $$T_1 = [a_1, b_1] \times \ldots \times [a_{i-1}, b_{i-1}] \times [a_i, v] \times [a_{i+1}, b_{i+1}] \times \ldots \times [a_n, b_n],$$
   $$T_2 = [a_1, b_1] \times \ldots \times [a_{i-1}, b_{i-1}] \times [v, b_i] \times [a_{i+1}, b_{i+1}] \times \ldots \times [a_n, b_n],$$
   then $a_T \leq a_{T_1}, a_{T_2}$ and $f_{T_1}, f_{T_2} \leq f_T$ hold.
   (B) Furthermore, for every $\varepsilon > 0$, there exists $\delta > 0$, such that for any $T$ rectangle, if $\forall_i : b_i - a_i \leq \delta$, then $0 \leq f_T - a_T \leq \varepsilon$, meaning if a rectangle's size approximates a point, then the upper and lower bounds will be close.

# Naive Algorithm

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

(M) // Examining each small rectangle

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

(M) // Examining each small rectangle
In this way, the lower and upper bounds corresponding to them will differ by at most $\varepsilon$.

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \le p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

(M) // Examining each small rectangle
In this way, the lower and upper bounds corresponding to them will differ by at most $\varepsilon$.

(Opt) // Optimization

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

(M) // Examining each small rectangle
In this way, the lower and upper bounds corresponding to them will differ by at most $\varepsilon$.

(Opt) // Optimization
Then calculate the lower bound for each small rectangle,

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

## Naive Algorithm

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

(M) // Examining each small rectangle
In this way, the lower and upper bounds corresponding to them will differ by at most $\varepsilon$.

(Opt) // Optimization
Then calculate the lower bound for each small rectangle,

(Out) // Output

# Naive Algorithm

- **Goal:** We want to minimize the nice function $c(x)$. More precisely, our goal is to find $x$ such that $c(x) \leq p^* + \varepsilon$.

**Naive Algorithm**

(T) // Building the entire tree (BRANCH)
Take a rectangle $T_0$ and divide it into small rectangles such that the edges of the resulting small rectangles are smaller than $\delta$.

(M) // Examining each small rectangle
In this way, the lower and upper bounds corresponding to them will differ by at most $\varepsilon$.

(Opt) // Optimization
Then calculate the lower bound for each small rectangle,

(Out) // Output
The smallest lower bound rectangle will have every point as a good output.

# Philosophy of Improvement: Branch-and-Bound

# Philosophy of Improvement: Branch-and-Bound

- The naive algorithm (M) step optimizes for exponentially many rectangles.

# Philosophy of Improvement: Branch-and-Bound

- The naive algorithm (M) step optimizes for exponentially many rectangles. Thus, its application may encounter difficulties in practice.

# Philosophy of Improvement: Branch-and-Bound

- The naive algorithm (M) step optimizes for exponentially many rectangles. Thus, its application may encounter difficulties in practice.

- We will now present an improved version.

# Philosophy of Improvement: Branch-and-Bound

• The naive algorithm (M) step optimizes for exponentially many rectangles. Thus, its application may encounter difficulties in practice.

• We will now present an improved version. Instead of slicing all rectangles in parallel, *thoughtlessly*, we only slice those that are the most promising.

# Philosophy of Improvement: Branch-and-Bound

• The naive algorithm (M) step optimizes for exponentially many rectangles. Thus, its application may encounter difficulties in practice.

• We will now present an improved version. Instead of slicing all rectangles in parallel, *thoughtlessly*, we only slice those that are the most promising.

• The BOUND step was missing from the above algorithm. The sequence of cuts was complete, and analysis only occurred when analyzing the final small cubes.

# Philosophy of Improvement: Branch-and-Bound

• The naive algorithm (M) step optimizes for exponentially many rectangles. Thus, its application may encounter difficulties in practice.

• We will now present an improved version. Instead of slicing all rectangles in parallel, *thoughtlessly*, we only slice those that are the most promising.

• The BOUND step was missing from the above algorithm. The sequence of cuts was complete, and analysis only occurred when analyzing the final small cubes.

• In our new procedure, we will have a $\mathcal{T}$ system of rectangles such that the interior points of its elements are disjoint, and $\cup\mathcal{T} = T_0$.

# Philosophy of Improvement: Branch-and-Bound

• The naive algorithm (M) step optimizes for exponentially many rectangles. Thus, its application may encounter difficulties in practice.

• We will now present an improved version. Instead of slicing all rectangles in parallel, *thoughtlessly*, we only slice those that are the most promising.

• The BOUND step was missing from the above algorithm. The sequence of cuts was complete, and analysis only occurred when analyzing the final small cubes.

• In our new procedure, we will have a $\mathcal{T}$ system of rectangles such that the interior points of its elements are disjoint, and $\cup\mathcal{T} = T_0$.

• We only slice one rectangle at a time.

# Branch-and-Bound Algorithm

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

(2) Select a dimension $i$ and an edge in the direction of $T$.

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

(2) Select a dimension $i$ and an edge in the direction of $T$.

(3) Split $T$ along this edge ($T'$ and $T''$ are the two resulting half-rectangles, $\mathcal{T} \leftarrow \mathcal{T} - \{T\} \cup \{T', T''\}$).

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

(2) Select a dimension $i$ and an edge in the direction of $T$.

(3) Split $T$ along this edge ($T'$ and $T''$ are the two resulting half-rectangles, $\mathcal{T} \leftarrow \mathcal{T} - \{T\} \cup \{T', T''\}$).

(4) For the new $\mathcal{T}$ system of rectangles, calculate $a_{\mathcal{T}} := \min_{T \in \mathcal{T}} a_T$ and $\Delta = f_T - a_T$, where $T$ is the rectangle whose lower bound is $a_{\mathcal{T}}$.

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

(2) Select a dimension $i$ and an edge in the direction of $T$.

(3) Split $T$ along this edge ($T'$ and $T''$ are the two resulting half-rectangles, $\mathcal{T} \leftarrow \mathcal{T} - \{T\} \cup \{T', T''\}$).

(4) For the new $\mathcal{T}$ system of rectangles, calculate $a_{\mathcal{T}} := \min_{T \in \mathcal{T}} a_T$ and $\Delta = f_T - a_T$, where $T$ is the rectangle whose lower bound is $a_{\mathcal{T}}$.

WHILE $\Delta > \varepsilon$, DO repeat steps (1)-(4).

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

(2) Select a dimension $i$ and an edge in the direction of $T$.

(3) Split $T$ along this edge ($T'$ and $T''$ are the two resulting half-rectangles, $\mathcal{T} \leftarrow \mathcal{T} - \{T\} \cup \{T', T''\}$).

(4) For the new $\mathcal{T}$ system of rectangles, calculate $a_{\mathcal{T}} := \min_{T \in \mathcal{T}} a_T$ and $\Delta = f_T - a_T$, where $T$ is the rectangle whose lower bound is $a_{\mathcal{T}}$.

WHILE $\Delta > \varepsilon$, DO repeat steps (1)-(4).

(STOP) If $f_{\mathcal{T}} - a_{\mathcal{T}} \leq \varepsilon$.

# Branch-and-Bound Algorithm

## Branch-and-Bound Algorithm

(0) Initially, let $\mathcal{T} = \{T_0\}$.

(1) Select a *promising* $T \in \mathcal{T}$.

(2) Select a dimension $i$ and an edge in the direction of $T$.

(3) Split $T$ along this edge ($T'$ and $T''$ are the two resulting half-rectangles, $\mathcal{T} \leftarrow \mathcal{T} - \{T\} \cup \{T', T''\}$).

(4) For the new $\mathcal{T}$ system of rectangles, calculate $a_{\mathcal{T}} := \min_{T \in \mathcal{T}} a_T$ and $\Delta = f_T - a_T$, where $T$ is the rectangle whose lower bound is $a_{\mathcal{T}}$.

WHILE $\Delta > \varepsilon$, DO repeat steps (1)-(4).

(STOP) If $f_{\mathcal{T}} - a_{\mathcal{T}} \leq \varepsilon$.

(OUTPUT): $a_{\mathcal{T}}, f_{\mathcal{T}}$, where $a_{\mathcal{T}} \leq p^* \leq f_{\mathcal{T}}$. Moreover, let $T : a_T = a_{\mathcal{T}}$ and output $x \in T$.

# Refinement (1)

# Refinement (1)

**(1)** Let $T$ be the rectangle with the smallest $a_T$.

# Refinement (1)

**(1)** Let $T$ be the rectangle with the smallest $a_T$.

• The advantage of this heuristic is summarized by the following observation.

**Observation**

If $T' \in \mathcal{T}$ is any rectangle such that $f_T \leq a_{T'}$

# Refinement (1)

**(1)** Let $T$ be the rectangle with the smallest $a_T$.

• The advantage of this heuristic is summarized by the following observation.

**Observation**

If $T' \in \mathcal{T}$ is any rectangle such that $f_T \leq a_{T'}$ (this is denoted as $[a_T, f_T] \leq [a_{T'}, f_{T'}]$),

# Refinement (1)

**(1)** Let $T$ be the rectangle with the smallest $a_T$.

• The advantage of this heuristic is summarized by the following observation.

**Observation**

If $T' \in \mathcal{T}$ is any rectangle such that $f_T \leq a_{T'}$ (this is denoted as $[a_T, f_T] \leq [a_{T'}, f_{T'}]$), then $T'$ will never be further subdivided.

# Refinement (1)

**(1)** Let $T$ be the rectangle with the smallest $a_T$.

• The advantage of this heuristic is summarized by the following observation.

**Observation**

If $T' \in \mathcal{T}$ is any rectangle such that $f_T \le a_{T'}$ (this is denoted as $[a_T, f_T] \le [a_{T'}, f_{T'}]$), then $T'$ will never be further subdivided.

• Indeed. There will always be a rectangle in our system that is a part of $T$.

# Refinement (1)

**(1)** Let $T$ be the rectangle with the smallest $a_T$.

• The advantage of this heuristic is summarized by the following observation.

**Observation**

If $T' \in \mathcal{T}$ is any rectangle such that $f_T \leq a_{T'}$ (this is denoted as $[a_T, f_T] \leq [a_{T'}, f_{T'}]$), then $T'$ will never be further subdivided.

• Indeed. There will always be a rectangle in our system that is a part of $T$. Its lower bound will be such that it *prevents* the selection of $T'$.

# Refinement (2)

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

- To exploit this heuristic, we will need further definitions.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

- To exploit this heuristic, we will need further definitions.

**Definition**

Let $T = [a_1 b_1] \times \ldots \times [a_n, b_n]$ be a rectangle.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

- To exploit this heuristic, we will need further definitions.

**Definition**

Let $T = [a_1 b_1] \times \ldots \times [a_n, b_n]$ be a rectangle.

(i) $\mathrm{vol}\,(T) = \Pi_{i=1}^{n}(b_i - a_i)$, the volume of $T$.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

- To exploit this heuristic, we will need further definitions.

**Definition**

Let $T = [a_1 b_1] \times \ldots \times [a_n, b_n]$ be a rectangle.

(i) vol $(T) = \Pi_{i=1}^{n}(b_i - a_i)$, the volume of $T$.

(ii) $|T| = \max_{i=1,\ldots,n}(b_i - a_i)$, the size of $T$.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

- To exploit this heuristic, we will need further definitions.

## Definition

Let $T = [a_1 b_1] \times \ldots \times [a_n, b_n]$ be a rectangle.

(i) vol $(T) = \Pi_{i=1}^{n}(b_i - a_i)$, the volume of $T$.

(ii) $|T| = \max_{i=1,\ldots,n}(b_i - a_i)$, the size of $T$.

(iii) $\mathrm{torz}(T) = \frac{\max_{i-1,\ldots,n}(b_i - a_i)}{\min_{i=1,\ldots,n}(b_i - a_i)}$ is the distortion of $T$.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

- To exploit this heuristic, we will need further definitions.

**Definition**

Let $T = [a_1 b_1] \times \ldots \times [a_n, b_n]$ be a rectangle.

(i) vol $(T) = \Pi_{i=1}^{n}(b_i - a_i)$, the volume of $T$.

(ii) $|T| = \max_{i=1,\ldots,n}(b_i - a_i)$, the size of $T$.

(iii) $\text{torz}(T) = \frac{\max_{i=1,\ldots,n}(b_i - a_i)}{\min_{i=1,\ldots,n}(b_i - a_i)}$ is the distortion of $T$.

- $\text{Torz}(T) \geq 1$ and equality holds if and only if our $n$-dimensional rectangle is a cube.

# Refinement (2)

**(2)** Let $i$ be the dimension of the longest side of $T$.

• To exploit this heuristic, we will need further definitions.

**Definition**

Let $T = [a_1 b_1] \times \ldots \times [a_n, b_n]$ be a rectangle.

(i) $\text{vol}\,(T) = \Pi_{i=1}^{n}(b_i - a_i)$, the volume of $T$.

(ii) $|T| = \max_{i=1,\ldots,n}(b_i - a_i)$, the size of $T$.

(iii) $\text{torz}(T) = \frac{\max_{i=1,\ldots,n}(b_i - a_i)}{\min_{i=1,\ldots,n}(b_i - a_i)}$ is the distortion of $T$.

• $\text{Torz}(T) \geq 1$ and equality holds if and only if our $n$-dimensional rectangle is a cube.

• The advantage of splitting the longest side is that we will never have overly *distorted* rectangles during our procedure.

# Remark and Proof

# Remark and Proof

## Remark

For any $T$ rectangle:

$$\text{torz}\,(T) \leq \max\{2, \mathrm{torz}(T_0)\},$$

where $T_0$ is the initial rectangle.

# Remark and Proof

**Remark**

For any $T$ rectangle:

$$\text{torz}\,(T) \leq \max\{2, \mathrm{torz}(T_0)\},$$

where $T_0$ is the initial rectangle.

- Let's take an arbitrary brick $T$ that satisfies the observation.

# Remark and Proof

## Remark

For any $T$ rectangle:

$$\text{torz}\,(T) \leq \max\{2, \text{torz}(T_0)\},$$

where $T_0$ is the initial rectangle.

- Let's take an arbitrary brick $T$ that satisfies the observation.
- We split the longest side of the original brick.

# Remark and Proof

**Remark**

For any $T$ rectangle:

$$\text{torz}\,(T) \leq \max\{2, \text{torz}(T_0)\},$$

where $T_0$ is the initial rectangle.

- Let's take an arbitrary brick $T$ that satisfies the observation.
- We split the longest side of the original brick.
- **Claim:** In this case, we obtain two bricks, each satisfying the assertion in the observation.

# Remark and Proof

**Remark**

For any $T$ rectangle:

$$\text{torz}\,(T) \leq \max\{2, \text{torz}(T_0)\},$$

where $T_0$ is the initial rectangle.

- Let's take an arbitrary brick $T$ that satisfies the observation.
- We split the longest side of the original brick.
- **Claim:** In this case, we obtain two bricks, each satisfying the assertion in the observation.
- From the claim, the observation follows by induction.

# Remark and Proof

## Remark

For any $T$ rectangle:

$$\text{torz}\,(T) \leq \max\{2, \text{torz}(T_0)\},$$

where $T_0$ is the initial rectangle.

- Let's take an arbitrary brick $T$ that satisfies the observation.
- We split the longest side of the original brick.
- **Claim:** In this case, we obtain two bricks, each satisfying the assertion in the observation.
- From the claim, the observation follows by induction.
- To prove the claim, consider the following two cases.

# Case Analysis

# Case Analysis

**Case 1:** *For the initial* $T$ *brick,* $torz(T) > 2$*.*

# Case Analysis

**Case 1:** *For the initial $T$ brick, torz $(T) > 2$.*

In this case, the original longest edge is halved, and the new brick's longest edge is at most the same size as the original longest edge.

# Case Analysis

**Case 1:** *For the initial $T$ brick, torz $(T) > 2$.*

In this case, the original longest edge is halved, and the new brick's longest edge is at most the same size as the original longest edge.

The length of the original shortest edge remains the same in the new brick.
Thus, the distortion of the new $T$ is at most that of $T$.

# Case Analysis

**Case 1:** *For the initial $T$ brick, torz $(T) > 2$.*

In this case, the original longest edge is halved, and the new brick's longest edge is at most the same size as the original longest edge.

The length of the original shortest edge remains the same in the new brick.
Thus, the distortion of the new $T$ is at most that of $T$.

**Case 2:** *Suppose torz $(T) \leq 2$.*

# Case Analysis

**Case 1:** *For the initial $T$ brick, torz $(T) > 2$.*

In this case, the original longest edge is halved, and the new brick's longest edge is at most the same size as the original longest edge.

The length of the original shortest edge remains the same in the new brick.
Thus, the distortion of the new $T$ is at most that of $T$.

**Case 2:** *Suppose torz $(T) \leq 2$.*

In this case, after halving, the original longest edge becomes the new brick's shortest edge, at half the size.

# Case Analysis

**Case 1:** *For the initial $T$ brick, torz $(T) > 2$.*

In this case, the original longest edge is halved, and the new brick's longest edge is at most the same size as the original longest edge.

The length of the original shortest edge remains the same in the new brick.
Thus, the distortion of the new $T$ is at most that of $T$.

**Case 2:** *Suppose torz $(T) \leq 2$.*

In this case, after halving, the original longest edge becomes the new brick's shortest edge, at half the size.

The length of the longest edge cannot increase, so the distortion of the new $T$ is at most 2, by the initial condition.

# Remark

# Remark

## Remark

For any rectangular prism $T$,

$$|T| \leq \sqrt[n]{(\text{torz } T)^{n-1} \cdot \text{vol } T}.$$

# Remark

## Remark

For any rectangular prism $T$,

$$|T| \leq \sqrt[n]{(\text{torz } T)^{n-1} \cdot \text{vol } T}.$$

$$\text{vol } T = \prod_i (b_i - a_i) \geq \max_i (b_i - a_i) \left( \min_i (b_i - a_i) \right)^{n-1} =$$

$$= \frac{(\max_i (b_i - a_i))^n}{\left( \frac{\max_i (b_i - a_i)}{\min_i (b_i - a_i)} \right)^{n-1}} = \frac{|T|^n}{(\text{torz } T)^{n-1}}.$$

# Summary

# Summary

From the above, we get that:

# Summary

From the above, we get that:

**Remark**

The above algorithm terminates after a finite number of bisecting steps if points (1) and (2) are carried out according to the heuristic.

# Summary

From the above, we get that:

**Remark**

The above algorithm terminates after a finite number of bisecting steps if points (1) and (2) are carried out according to the heuristic.

- Apply the algorithm until $N$ bricks are formed.

# Summary

From the above, we get that:

**Remark**

The above algorithm terminates after a finite number of bisecting steps if points (1) and (2) are carried out according to the heuristic.

- Apply the algorithm until $N$ bricks are formed.
- Let $T$ be the brick with the smallest volume that the algorithm has constructed (so vol $T \leq$ vol $T_0/N$).

# Summary

From the above, we get that:

**Remark**

The above algorithm terminates after a finite number of bisecting steps if points (1) and (2) are carried out according to the heuristic.

- Apply the algorithm until $N$ bricks are formed.
- Let $T$ be the brick with the smallest volume that the algorithm has constructed (so vol $T \leq$ vol $T_0/N$).
- Then, by utilizing our observations, we obtain that $N$ can be chosen such that $|T| \leq \frac{\delta}{2}$.

# Summary

From the above, we get that:

**Remark**

The above algorithm terminates after a finite number of bisecting steps if points (1) and (2) are carried out according to the heuristic.

- Apply the algorithm until $N$ bricks are formed.
- Let $T$ be the brick with the smallest volume that the algorithm has constructed (so vol $T \leq$ vol $T_0/N$).
- Then, by utilizing our observations, we obtain that $N$ can be chosen such that $|T| \leq \frac{\delta}{2}$.
- This can only happen if $T$ was obtained by bisecting $T^-$ such that $|T^-| \leq \delta$.

# Summary

From the above, we get that:

**Remark**

The above algorithm terminates after a finite number of bisecting steps if points (1) and (2) are carried out according to the heuristic.

- Apply the algorithm until $N$ bricks are formed.
- Let $T$ be the brick with the smallest volume that the algorithm has constructed (so vol $T \leq$ vol $T_0/N$).
- Then, by utilizing our observations, we obtain that $N$ can be chosen such that $|T| \leq \frac{\delta}{2}$.
- This can only happen if $T$ was obtained by bisecting $T^-$ such that $|T^-| \leq \delta$.
- However, in this case, the algorithm should have stopped (see condition (3) for the beauty of $c$).

# Break

# The Basic Question

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, d \in \{0, 1\}^\nu$, and $c, f_i$ are convex functions.

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, d \in \{0, 1\}^\nu$, and $c, f_i$ are convex functions.
• If the condition $d \in \{0, 1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =?$

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, d \in \{0, 1\}^\nu$, and $c, f_i$ are convex functions.

- If the condition $d \in \{0, 1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =?$
- However, the condition poses difficulty, as the naive solution:

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, d \in \{0, 1\}^\nu$, and $c, f_i$ are convex functions.

• If the condition $d \in \{0, 1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =?$

• However, the condition poses difficulty, as the naive solution:

## Naive algorithm

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n$, $d \in \{0, 1\}^\nu$, and $c, f_i$ are convex functions.

- If the condition $d \in \{0, 1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =$?
- However, the condition poses difficulty, as the naive solution:

**Naive algorithm**

(1) Fix $d$ in all possible ways.

# The Basic Question

| | |
|---|---|
| Minimize | $c(x,d)$-t |
| subject to | $f_i(x,d) \leq 0$, if $i = 1,2,\ldots,k$ |

where $x \in \mathbb{R}^n$, $d \in \{0,1\}^\nu$, and $c, f_i$ are convex functions.

• If the condition $d \in \{0,1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =?$

• However, the condition poses difficulty, as the naive solution:

**Naive algorithm**

(1) Fix $d$ in all possible ways.

(2) Handle the $2^\nu$ resulting problems.

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, d \in \{0,1\}^\nu$, and $c, f_i$ are convex functions.

• If the condition $d \in \{0,1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =?$

• However, the condition poses difficulty, as the naive solution:

**Naive algorithm**

(1) Fix $d$ in all possible ways.

(2) Handle the $2^\nu$ resulting problems.

(3) The best obtained value is the optimum.

# The Basic Question

| | |
|---|---|
| Minimize | $c(x, d)$-t |
| subject to | $f_i(x, d) \leq 0$, if $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, d \in \{0, 1\}^\nu$, and $c, f_i$ are convex functions.

• If the condition $d \in \{0, 1\}^\nu$ were absent, we would have an easily manageable problem. $p^* =?$

• However, the condition poses difficulty, as the naive solution:

**Naive algorithm**

(1) Fix $d$ in all possible ways.

(2) Handle the $2^\nu$ resulting problems.

(3) The best obtained value is the optimum.

Even for small $\nu$ values, $2^\nu$ becomes too large for efficient handling.

# Estimations

# Estimations

- Lower and upper estimates can be provided for the optimal value using relaxation methods.

# Estimations

- Lower and upper estimates can be provided for the optimal value using relaxation methods.
- Relax the condition $d \in \{0,1\}^\nu$ to $0 \preceq d \preceq 1$ $(0, d, 1 \in \mathbb{R}^\nu)$.

# Estimations

- Lower and upper estimates can be provided for the optimal value using relaxation methods.
- Relax the condition $d \in \{0,1\}^\nu$ to $0 \preceq d \preceq 1$ ($0, d, 1 \in \mathbb{R}^\nu$).
- The resulting continuous convex $p^*_{\mathbb{R}}$ underestimates the optimal value $p^*$.

# Estimations

- Lower and upper estimates can be provided for the optimal value using relaxation methods.
- Relax the condition $d \in \{0,1\}^\nu$ to $0 \preceq d \preceq 1$ $(0, d, 1 \in \mathbb{R}^\nu)$.
- The resulting continuous convex $p^*_{\mathbb{R}}$ underestimates the optimal value $p^*$.
- An upper estimate can be obtained by evaluating the $c$ objective function at a *good* feasible solution.

# Estimations

• Lower and upper estimates can be provided for the optimal value using relaxation methods.

• Relax the condition $d \in \{0,1\}^\nu$ to $0 \preceq d \preceq 1$ ($0, d, 1 \in \mathbb{R}^\nu$).

• The resulting continuous convex $p^*_{\mathbb{R}}$ underestimates the optimal value $p^*$.

• An upper estimate can be obtained by evaluating the $c$ objective function at a *good* feasible solution.

• A *good* feasible solution is obtained by rounding the $[0,1]$-valued components of the optimal position $x^*_{\mathbb{R}}$ of the continuous problem to integers (i.e., to $\{0,1\}$).

# Estimations

- Lower and upper estimates can be provided for the optimal value using relaxation methods.
- Relax the condition $d \in \{0,1\}^\nu$ to $0 \preceq d \preceq 1$ $(0, d, 1 \in \mathbb{R}^\nu)$.
- The resulting continuous convex $p^*_{\mathbb{R}}$ underestimates the optimal value $p^*$.
- An upper estimate can be obtained by evaluating the $c$ objective function at a *good* feasible solution.
- A *good* feasible solution is obtained by rounding the $[0,1]$-valued components of the optimal position $x^*_{\mathbb{R}}$ of the continuous problem to integers (i.e., to $\{0,1\}$).
- Using this, for the $F$ problem, we can provide a lower $a_F$ and an upper $f_F$ estimate for the optimal value.

# Branching

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.
- Select a component $i \in \{1, \ldots, k\}$ that is *free*.

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.
- Select a component $i \in \{1, \ldots, k\}$ that is *free*.
- With a decision of $d_i = 0$ or $d_i = 1$, we get a subproblem each.

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.
- Select a component $i \in \{1, \dots, k\}$ that is *free*.
- With a decision of $d_i = 0$ or $d_i = 1$, we get a subproblem each.
- Consider the subproblem with $d_i = 0$, whose optimal value is $p_0^*$. This has one discrete variable less than the original, unmanageable problem. Nevertheless, the lower/upper estimate technique can be applied to it.

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.
- Select a component $i \in \{1, \ldots, k\}$ that is *free*.
- With a decision of $d_i = 0$ or $d_i = 1$, we get a subproblem each.
- Consider the subproblem with $d_i = 0$, whose optimal value is $p_0^*$. This has one discrete variable less than the original, unmanageable problem. Nevertheless, the lower/upper estimate technique can be applied to it.
- Consider the subproblem with $d_i = 1$, handle it similarly.

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.

- Select a component $i \in \{1, \ldots, k\}$ that is *free*.

- With a decision of $d_i = 0$ or $d_i = 1$, we get a subproblem each.

- Consider the subproblem with $d_i = 0$, whose optimal value is $p_0^*$. This has one discrete variable less than the original, unmanageable problem. Nevertheless, the lower/upper estimate technique can be applied to it.

- Consider the subproblem with $d_i = 1$, handle it similarly.

- The two new problems can be appended to the previous tree: From $\ell$ with $d_i = 0$ and $\ell$ with $d_i = 1$, $\ell_0$ and $\ell_1$ descendants are created.

# Branching

- Take the original problem. View it as a root of a tree containing a set of problems with the root/one leaf $\ell$ node.

- Select a component $i \in \{1, \ldots, k\}$ that is *free*.

- With a decision of $d_i = 0$ or $d_i = 1$, we get a subproblem each.

- Consider the subproblem with $d_i = 0$, whose optimal value is $p_0^*$. This has one discrete variable less than the original, unmanageable problem. Nevertheless, the lower/upper estimate technique can be applied to it.

- Consider the subproblem with $d_i = 1$, handle it similarly.

- The two new problems can be appended to the previous tree: From $\ell$ with $d_i = 0$ and $\ell$ with $d_i = 1$, $\ell_0$ and $\ell_1$ descendants are created.

- The original problems and the two subproblems can be represented in a rooted tree.

# The Branch-and-Bound Algorithm

# The Branch-and-Bound Algorithm

## The Branch-and-Bound Algorithm

# The Branch-and-Bound Algorithm

## The Branch-and-Bound Algorithm

(0) Let $T$ be a 1-node rooted tree, with its only node (and leaf) representing the initial problem.

# The Branch-and-Bound Algorithm

## The Branch-and-Bound Algorithm

(0) Let $T$ be a 1-node rooted tree, with its only node (and leaf) representing the initial problem.
Calculate the lower/upper estimates $a, f$ for this problem.

WHILE $f - a > \varepsilon$

(1) Select a leaf/problem $\ell$ from $T$.

(2) Choose one non-fixed component $d$ from the selected leaf/problem. Denote this component as $i$.

# The Branch-and-Bound Algorithm (Continued)

# The Branch-and-Bound Algorithm (Continued)

## The Branch-and-Bound Algorithm (Continued)

(3a) Let $\ell_0$ be the problem obtained from $\ell$ with $d_i = 0$ selection.

($3b_0$) Relax the remaining components and compute the $a_0, f_0$ lower and upper estimates in the same manner as before.

($3b_1$) Let $\ell_1$ be the problem obtained from $\ell$ with $d_i = 1$ selection. Relax the remaining components and compute the $a_1, f_1$ lower and upper estimates in the same manner as before.

(4) Let $T$ be the tree obtained from $\ell$ by branching into the $\ell_0$ and $\ell_1$ leaves. Set $a = \min\{a, a_0, a_1\}$ and $f = \min\{f, f_0, f_1\}$.

# Clarifying the Details

# Clarifying the Details

- In step (1), we choose the leaf with the smallest lower estimate.

# Clarifying the Details

- In step (1), we choose the leaf with the smallest lower estimate.

- In step (2), for the relaxed problem of the leaf, based on the optimal position, we choose the component closest to $\frac{1}{2}$.

# Clarifying the Details

- In step (1), we choose the leaf with the smallest lower estimate.

- In step (2), for the relaxed problem of the leaf, based on the optimal position, we choose the component closest to $\frac{1}{2}$.

- It is evident here as well that for some subproblems, there might not be a need for further *bisection*.

# Clarifying the Details

- In step (1), we choose the leaf with the smallest lower estimate.

- In step (2), for the relaxed problem of the leaf, based on the optimal position, we choose the component closest to $\frac{1}{2}$.

- It is evident here as well that for some subproblems, there might not be a need for further *bisection*.

- Our tree cannot grow beyond the full depth binary tree with $\nu$ levels.

# A New Fundamental Question

# A New Fundamental Question

| | |
|---|---|
| Minimize | $\vert\{i : x_i \neq 0\}\vert$-t |
| subject to | $Ax \preceq b,$ |

where $x \in \mathbb{R}^n, A \in \mathbb{R}^{k\times n}, b \in \mathbb{R}^k$.

# A New Fundamental Question

| | |
|---|---|
| Minimize | $\|\{i : x_i \neq 0\}\|$-t |
| subject to | $Ax \preceq b$, |

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{k \times n}$, $b \in \mathbb{R}^k$.

• In other words, we want to solve a linear inequality system, where the solution has the fewest possible non-zero components.

# A New Fundamental Question

| | |
|---|---|
| Minimize | $\lvert\{i : x_i \neq 0\}\rvert$-t |
| subject to | $Ax \preceq b$, |

where $x \in \mathbb{R}^n$, $A \in \mathbb{R}^{k\times n}$, $b \in \mathbb{R}^k$.

• In other words, we want to solve a linear inequality system, where the solution has the fewest possible non-zero components.

• We reduce our task to a mixed convex-integer type problem.

# Auxiliary LP Problems

# Auxiliary LP Problems

To achieve the reduction, first solve the following $2n$ LP problems $(i = 1, \ldots, n)$

# Auxiliary LP Problems

To achieve the reduction, first solve the following $2n$ LP problems $(i = 1, \ldots, n)$

| | |
|---|---|
| Minimize | $x_i$-t |
| subject to | $Ax \preceq b$, |

# Auxiliary LP Problems

To achieve the reduction, first solve the following $2n$ LP problems $(i = 1, \ldots, n)$

| | |
|---|---|
| Minimize | $x_i$-t |
| subject to | $Ax \preceq b$, |

and

| | |
|---|---|
| Maximize | $x_i$-t |
| subject to | $Ax \preceq b$. |

# Auxiliary LP Problems

To achieve the reduction, first solve the following $2n$ LP problems ($i = 1, \ldots, n$)

| | |
|---|---|
| Minimize | $x_i$-t |
| subject to | $Ax \preceq b$, |

and

| | |
|---|---|
| Maximize | $x_i$-t |
| subject to | $Ax \preceq b$. |

Let the optimum of the first $n$ LP problems be $m_i$,

# Auxiliary LP Problems

To achieve the reduction, first solve the following $2n$ LP problems ($i = 1, \ldots, n$)

| | |
|---|---|
| Minimize | $x_i$-t |
| subject to | $Ax \preceq b$, |

and

| | |
|---|---|
| Maximize | $x_i$-t |
| subject to | $Ax \preceq b$. |

Let the optimum of the first $n$ LP problems be $m_i$, and of the second $n$ be $M_i$.

# The New Fundamental Question as MIP

# The New Fundamental Question as MIP

- From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.

# The New Fundamental Question as MIP

- From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.
- We add the constraints $m_i y_i \leq x_i \leq M_i y_i$ to our conditions, where $y_i \in \{0, 1\}$, new Boolean variables.

# The New Fundamental Question as MIP

• From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.

• We add the constraints $m_i y_i \leq x_i \leq M_i y_i$ to our conditions, where $y_i \in \{0, 1\}$, new Boolean variables.

• When $y_i = 1$, the constraint is trivial, while when $y_i = 0$, it forces $x_i = 0$.

# The New Fundamental Question as MIP

- From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.
- We add the constraints $m_i y_i \leq x_i \leq M_i y_i$ to our conditions, where $y_i \in \{0, 1\}$, new Boolean variables.
- When $y_i = 1$, the constraint is trivial, while when $y_i = 0$, it forces $x_i = 0$.
- Thus, our aim is to maximize the number of $y_i = 0$, i.e., minimize $\sum_{i=1}^{n} y_i$.

# The New Fundamental Question as MIP

- From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.
- We add the constraints $m_i y_i \leq x_i \leq M_i y_i$ to our conditions, where $y_i \in \{0, 1\}$, new Boolean variables.
- When $y_i = 1$, the constraint is trivial, while when $y_i = 0$, it forces $x_i = 0$.
- Thus, our aim is to maximize the number of $y_i = 0$, i.e., minimize $\sum_{i=1}^{n} y_i$.
- Therefore, the equivalent mixed IP problem is:

# The New Fundamental Question as MIP

- From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.
- We add the constraints $m_i y_i \leq x_i \leq M_i y_i$ to our conditions, where $y_i \in \{0,1\}$, new Boolean variables.
- When $y_i = 1$, the constraint is trivial, while when $y_i = 0$, it forces $x_i = 0$.
- Thus, our aim is to maximize the number of $y_i = 0$, i.e., minimize $\sum_{i=1}^n y_i$.
- Therefore, the equivalent mixed IP problem is:

| | | |
|---|---|---|
| Minimize | $\sum_{i=1}^n y_i$-t | |
| subject to | $Ax \preceq b$ | |
| | $m_i y_i \leq x_i \leq M_i y_i,$ | $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, y = (y_i)_{i=1}^n \in \{0,1\}^n,\ A \in \mathbb{R}^{k\times n}, b \in \mathbb{R}^k$.

# The New Fundamental Question as MIP

- From $Ax \preceq b$, it follows that $m_i \leq x_i \leq M_i$.
- We add the constraints $m_i y_i \leq x_i \leq M_i y_i$ to our conditions, where $y_i \in \{0, 1\}$, new Boolean variables.
- When $y_i = 1$, the constraint is trivial, while when $y_i = 0$, it forces $x_i = 0$.
- Thus, our aim is to maximize the number of $y_i = 0$, i.e., minimize $\sum_{i=1}^n y_i$.
- Therefore, the equivalent mixed IP problem is:

| | | |
|---|---|---|
| Minimize | $\sum_{i=1}^n y_i$-t | |
| subject to | $Ax \preceq b$ | |
| | $m_i y_i \leq x_i \leq M_i y_i,$ | $i = 1, 2, \ldots, k$ |

where $x \in \mathbb{R}^n, y = (y_i)_{i=1}^n \in \{0, 1\}^n,\ A \in \mathbb{R}^{k \times n}, b \in \mathbb{R}^k$.

- Our previous method can be applied.

# Break

# Fundamental Question: IP

# Fundamental Question: IP

- Reminder: What do we know about the *logic* of solving linear systems of equations?

# Fundamental Question: IP

- Reminder: What do we know about the *logic* of solving linear systems of equations?
- We already learned in elementary school that

# Fundamental Question: IP

- Reminder: What do we know about the *logic* of solving linear systems of equations?
- We already learned in elementary school that

(i) Inequalities can be added (assuming we always arrange them so that the smaller side is on the left).

# Fundamental Question: IP

• Reminder: What do we know about the *logic* of solving linear systems of equations?

• We already learned in elementary school that

(i) Inequalities can be added (assuming we always arrange them so that the smaller side is on the left).

(ii) Inequalities can be multiplied by non-negative numbers (multiplying by 0 leads to — obviously true — $0 \leq 0$ inequality).

# Fundamental Question: IP

- Reminder: What do we know about the *logic* of solving linear systems of equations?
- We already learned in elementary school that

(i) Inequalities can be added (assuming we always arrange them so that the smaller side is on the left).

(ii) Inequalities can be multiplied by non-negative numbers (multiplying by 0 leads to — obviously true — $0 \leq 0$ inequality).

- The inequalities *derived* this way are consequences of the initial conditions.

# Fundamental Question: IP

• Reminder: What do we know about the *logic* of solving linear systems of equations?

• We already learned in elementary school that

(i) Inequalities can be added (assuming we always arrange them so that the smaller side is on the left).

(ii) Inequalities can be multiplied by non-negative numbers (multiplying by 0 leads to — obviously true — $0 \leq 0$ inequality).

• The inequalities *derived* this way are consequences of the initial conditions. Adding them to our system of conditions does not change the set of possible solutions.

# Fundamental Question: IP

• Reminder: What do we know about the *logic* of solving linear systems of equations?

• We already learned in elementary school that

(i) Inequalities can be added (assuming we always arrange them so that the smaller side is on the left).

(ii) Inequalities can be multiplied by non-negative numbers (multiplying by 0 leads to — obviously true — $0 \leq 0$ inequality).

• The inequalities *derived* this way are consequences of the initial conditions. Adding them to our system of conditions does not change the set of possible solutions.

• If we obtain a new inequality in this manner, we say that we have made an L-inference.

# Fundamental Question: IP

- Reminder: What do we know about the *logic* of solving linear systems of equations?
- We already learned in elementary school that

(i) Inequalities can be added (assuming we always arrange them so that the smaller side is on the left).

(ii) Inequalities can be multiplied by non-negative numbers (multiplying by 0 leads to — obviously true — $0 \leq 0$ inequality).

- The inequalities *derived* this way are consequences of the initial conditions. Adding them to our system of conditions does not change the set of possible solutions.
- If we obtain a new inequality in this manner, we say that we have made an L-inference. (L$\equiv$ linear, real.)

# Integer Solutions

# Integer Solutions

- The above logic can, of course, be applied while seeking integer solutions to inequality systems.

# Integer Solutions

- The above logic can, of course, be applied while seeking integer solutions to inequality systems.

- By taking non-negative linear combinations of our inequalities, we obtain a consequence of our conditions.

# Integer Solutions

- The above logic can, of course, be applied while seeking integer solutions to inequality systems.
- By taking non-negative linear combinations of our inequalities, we obtain a consequence of our conditions.
- Specifically, we do not lose possible real solutions.

# Integer Solutions

- The above logic can, of course, be applied while seeking integer solutions to inequality systems.

- By taking non-negative linear combinations of our inequalities, we obtain a consequence of our conditions.

- Specifically, we do not lose possible real solutions.

- However, if the possible solutions are integers, then we aim not to lose integer solutions.

# Integer Solutions

- The above logic can, of course, be applied while seeking integer solutions to inequality systems.

- By taking non-negative linear combinations of our inequalities, we obtain a consequence of our conditions.

- Specifically, we do not lose possible real solutions.

- However, if the possible solutions are integers, then we aim not to lose integer solutions.

- We can also make new logical inferences.

# Extended Logic

# Extended Logic

Consider the following integer programming problem:

| | |
|---|---|
| Minimize | $a^T x$-t |
| subject to | $Ax \preceq b$ |
| | $x \succeq 0,\ x \in \mathbb{Z}^n$ |

# Extended Logic

Consider the following integer programming problem:

| | |
|---|---|
| Minimize | $a^T x$-t |
| subject to | $Ax \preceq b$ |
| | $x \succeq 0$, $x \in \mathbb{Z}^n$ |

## A New Inference

Let $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta$ be an inequality that is an $L$-consequence of the conditions.

# Extended Logic

Consider the following integer programming problem:

| | |
|---|---|
| Minimize | $a^T x$-t |
| subject to | $Ax \preceq b$ |
| | $x \succeq 0,\ x \in \mathbb{Z}^n$ |

## A New Inference

Let $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta$ be an inequality that is an $L$-consequence of the conditions. That is, all possible real solutions satisfy it.

# Extended Logic

Consider the following integer programming problem:

| | |
|---|---|
| Minimize | $a^T x$-t |
| subject to | $Ax \preceq b$ |
| | $x \succeq 0,\ x \in \mathbb{Z}^n$ |

## A New Inference

Let $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta$ be an inequality that is an $L$-consequence of the conditions. That is, all possible real solutions satisfy it.

Then

$$\lfloor \alpha_1 \rfloor x_1 + \lfloor \alpha_2 \rfloor x_2 + \ldots + \lfloor \alpha_n \rfloor x_n \leq \lfloor \beta \rfloor$$

is also a consequence of the conditions.

# Extended Logic

Consider the following integer programming problem:

| | |
|---|---|
| Minimize | $a^T x$-t |
| subject to | $Ax \preceq b$ |
| | $x \succeq 0,\ x \in \mathbb{Z}^n$ |

## A New Inference

Let $\alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta$ be an inequality that is an $L$-consequence of the conditions. That is, all possible real solutions satisfy it.

Then

$$\lfloor\alpha_1\rfloor x_1 + \lfloor\alpha_2\rfloor x_2 + \ldots + \lfloor\alpha_n\rfloor x_n \leq \lfloor\beta\rfloor$$

is also a consequence of the conditions. By adding it to our conditions, we do not lose possible (integer) solutions.

# Proof

# Proof

- Indeed: $\lfloor \alpha \rfloor \leq \alpha$.

# Proof

- Indeed: $\lfloor\alpha\rfloor \leq \alpha$.

- Thus, for non-negative $x$, $\lfloor\alpha\rfloor x \leq \alpha x$.

# Proof

- Indeed: $\lfloor \alpha \rfloor \leq \alpha$.

- Thus, for non-negative $x$, $\lfloor \alpha \rfloor x \leq \alpha x$.

- Generally,

$$\lfloor \alpha_1 \rfloor x_1 + \lfloor \alpha_2 \rfloor x_2 + \ldots + \lfloor \alpha_n \rfloor x_n \leq \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta.$$

# Proof

- Indeed: $\lfloor \alpha \rfloor \leq \alpha$.

- Thus, for non-negative $x$, $\lfloor \alpha \rfloor x \leq \alpha x$.

- Generally,

$$\lfloor \alpha_1 \rfloor x_1 + \lfloor \alpha_2 \rfloor x_2 + \ldots + \lfloor \alpha_n \rfloor x_n \leq \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta.$$

- Moreover, if the $x_i$ are integers, then the left side is also an integer. Thus, the upper bound $\beta$ can be improved to $\lfloor \beta \rfloor$.

# Proof

- Indeed: $\lfloor \alpha \rfloor \leq \alpha$.

- Thus, for non-negative $x$, $\lfloor \alpha \rfloor x \leq \alpha x$.

- Generally,

$$\lfloor \alpha_1 \rfloor x_1 + \lfloor \alpha_2 \rfloor x_2 + \ldots + \lfloor \alpha_n \rfloor x_n \leq \alpha_1 x_1 + \alpha_2 x_2 + \ldots + \alpha_n x_n \leq \beta.$$

- Moreover, if the $x_i$ are integers, then the left side is also an integer. Thus, the upper bound $\beta$ can be improved to $\lfloor \beta \rfloor$.

- This proves the claim.

# Example (ALGEBRA)

# Example (ALGEBRA)

- The simple argument above has a straightforward geometric interpretation.

# Example (ALGEBRA)

- The simple argument above has a straightforward geometric interpretation.

- Consider the following problem:

| | |
|---|---|
| Minimize | $-2x-5y$-t |
| subject to | $10x+3y\leq 45$ |
| | $4x+20y\leq 65$ |
| | $x,y\in\mathbb{N}$ |

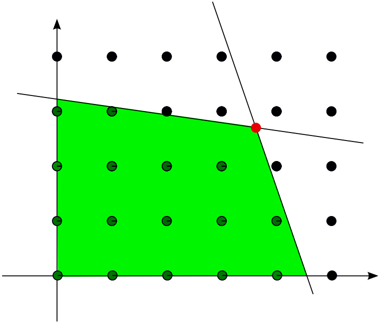
# Example (GEOMETRY)

# Example (GEOMETRY)

The diagram shows the possible solutions.

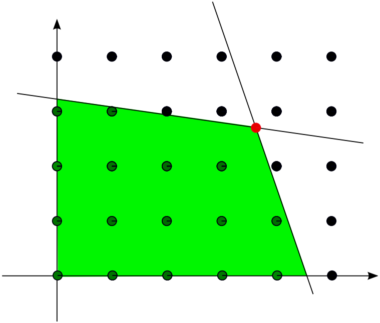
# Example (GEOMETRY)

The diagram shows the possible solutions.



# Example (GEOMETRY)

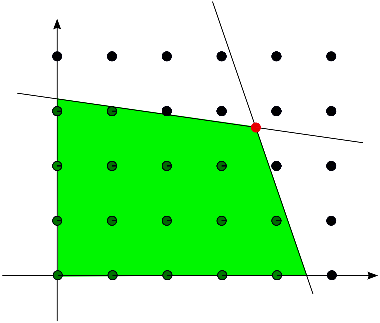
The diagram shows the possible solutions.



The green region is the polytope of LP relaxation.

# Example (GEOMETRY)

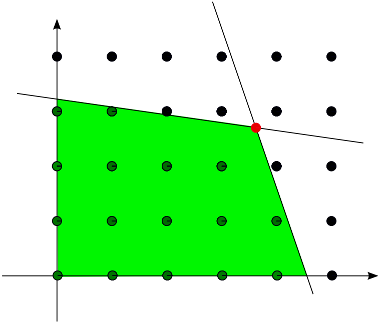
The diagram shows the possible solutions.



The green region is the polytope of LP relaxation. The dark green discrete set is the finite set of possible solutions to the integer problem.

# Example (GEOMETRY)

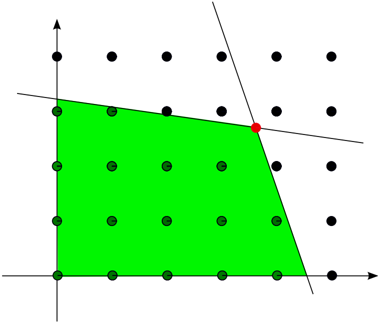
The diagram shows the possible solutions.



The green region is the polytope of LP relaxation. The dark green discrete set is the finite set of possible solutions to the integer problem. The LP problem's optimum is the red point: $\left(\frac{15}{4}, \frac{5}{2}\right) = (3{,}75, 2{,}5)$,

# Example (GEOMETRY)

The diagram shows the possible solutions.



The green region is the polytope of LP relaxation. The dark green discrete set is the finite set of possible solutions to the integer problem. The LP problem's optimum is the red point: $\left(\frac{15}{4}, \frac{5}{2}\right) = (3{,}75, 2{,}5)$, which is not a feasible solution for the integer problem.

# Example (LOGIC)

# Example (LOGIC)

Let's write down some L-inferences.

# Example (LOGIC)

Let's write down some L-inferences.

- Multiplying the first inequality by $1/10$:

$$x + 0{,}3y \leq 4{,}5.$$

# Example (LOGIC)

Let's write down some L-inferences.

- Multiplying the first inequality by $1/10$:

$$x + 0{,}3y \leq 4{,}5.$$

- Multiplying the second inequality by $1/4$.

$$x + 5y \leq 16{,}25.$$

# Example (LOGIC)

Let's write down some L-inferences.

- Multiplying the first inequality by $1/10$:

$$x + 0{,}3y \leq 4{,}5.$$

- Multiplying the second inequality by $1/4$.

$$x + 5y \leq 16{,}25.$$

- Adding the first inequality twice to the second one, then dividing by 24, we get that

$$x + 1\frac{1}{12}y \leq 6\frac{11}{24}.$$

# Example (LOGIC)

Let's write down some L-inferences.

- Multiplying the first inequality by $1/10$:

$$x + 0{,}3y \leq 4{,}5.$$

- Multiplying the second inequality by $1/4$.

$$x + 5y \leq 16{,}25.$$

- Adding the first inequality twice to the second one, then dividing by 24, we get that

$$x + 1\frac{1}{12}y \leq 6\frac{11}{24}.$$

(GEOMETRY: All three inferences describe a half-plane whose boundary passes through the red point (why?).)

# Example (LOGIC II)

# Example (LOGIC II)

- Apply both L- and I-inferences to the above.

# Example (LOGIC II)

- Apply both L- and I-inferences to the above.
- We obtain the following three inequalities:

$$x \leq 4,$$

$$x + 5y \leq 16,$$

$$x + y \leq 6.$$

# Example (LOGIC II)

- Apply both L- and I-inferences to the above.
- We obtain the following three inequalities:

$$x \leq 4,$$

$$x + 5y \leq 16,$$

$$x + y \leq 6.$$

- None of these inequalities *excludes* integer-coordinate points from the solution set.

# Example (GEOMETRY II)

# Example (GEOMETRY II)

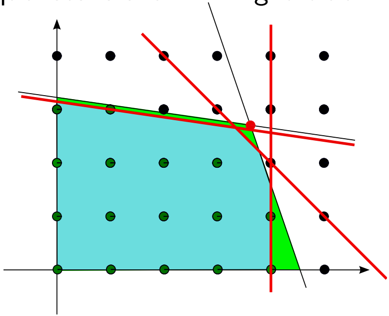
The half-planes described by
the above I-inferences are marked in red.

# Example (GEOMETRY II)

The half-planes described by
the above I-inferences are marked in red. The solution set after adding the consequences is shown in light blue.

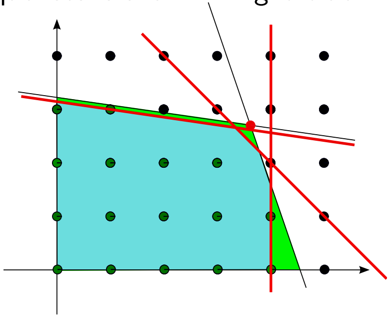
# Example (GEOMETRY II)

The half-planes described by
the above I-inferences are marked in red. The solution set after adding the consequences is shown in light blue.



# Example (GEOMETRY II)

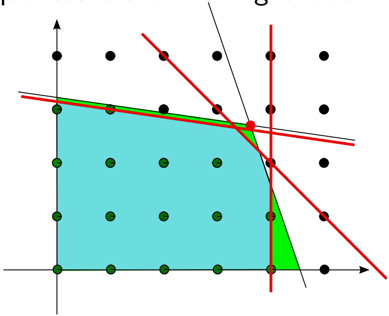
The half-planes described by
the above I-inferences are marked in red. The solution set after adding the consequences is shown in light blue.



The decrease in the described polytope is obvious.

# Example (GEOMETRY II)

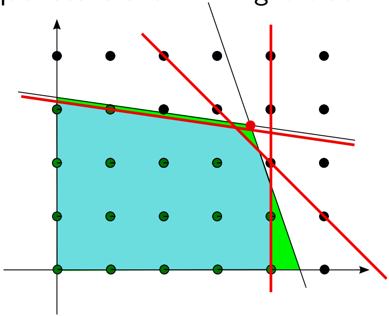
The half-planes described by
the above I-inferences are marked in red. The solution set after adding the consequences is shown in light blue.



The decrease in the described polytope is obvious. During the decrease, the dark green points did not leave the solution set.

# Example (GEOMETRY II)

The half-planes described by
the above I-inferences are marked in red. The solution set after adding the consequences is shown in light blue.



The decrease in the described polytope is obvious. During the decrease, the dark green points did not leave the solution set. The polytope described by the LP relaxation approached the convex hull of the dark green points.

# General Scheme

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

(R) // RELAXATION step

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

(R) // RELAXATION step
Solve the LP relaxation of the IP problem.

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

(R) // RELAXATION step
Solve the LP relaxation of the IP problem.

(L) // LUCK

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

(R) // RELAXATION step
Solve the LP relaxation of the IP problem.

(L) // LUCK
If we obtain an integer-coordinate optimum, then we have solved our problem.

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

(R) // RELAXATION step
Solve the LP relaxation of the IP problem.

(L) // LUCK
If we obtain an integer-coordinate optimum, then we have solved our problem.

(L) // LOGIC

# General Scheme

Following this, a possible scheme to solve the posed IP problem is the following:

**Algorithm**

(R) // RELAXATION step
Solve the LP relaxation of the IP problem.

(L) // LUCK
If we obtain an integer-coordinate optimum, then we have solved our problem.

(L) // LOGIC
Make L- and I-inferences to add new linear inequalities to our problem. Return to the relaxation step.

# Gomory's Algorithm

# Gomory's Algorithm

- The above-described procedure is just a scheme.

# Gomory's Algorithm

- The above-described procedure is just a scheme. The essential part lies in the general step.

# Gomory's Algorithm

- The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.

# Gomory's Algorithm

• The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.

• Many attempts/solutions, many algorithms handling relatively large special problems.

# Gomory's Algorithm

- The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.
- Many attempts/solutions, many algorithms handling relatively large special problems.
- Gomory provided a procedure where he realized that his algorithm finds the integer optimum in a finite number of steps.

# Gomory's Algorithm

- The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.
- Many attempts/solutions, many algorithms handling relatively large special problems.
- Gomory provided a procedure where he realized that his algorithm finds the integer optimum in a finite number of steps.
- He uses the simplex method to generate new inequalities.

# Gomory's Algorithm

• The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.

• Many attempts/solutions, many algorithms handling relatively large special problems.

• Gomory provided a procedure where he realized that his algorithm finds the integer optimum in a finite number of steps.

• He uses the simplex method to generate new inequalities. There's no time to describe the algorithm.

# Gomory's Algorithm

• The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.

• Many attempts/solutions, many algorithms handling relatively large special problems.

• Gomory provided a procedure where he realized that his algorithm finds the integer optimum in a finite number of steps.

• He uses the simplex method to generate new inequalities. There's no time to describe the algorithm.

• As we saw with the Edmonds' polyhedron theorem, for the LP description of the IP problem, exponentially many inequalities may be needed.

# Gomory's Algorithm

• The above-described procedure is just a scheme. The essential part lies in the general step. How should we choose the inferences? Many questions to clarify.

• Many attempts/solutions, many algorithms handling relatively large special problems.

• Gomory provided a procedure where he realized that his algorithm finds the integer optimum in a finite number of steps.

• He uses the simplex method to generate new inequalities. There's no time to describe the algorithm.

• As we saw with the Edmonds' polyhedron theorem, for the LP description of the IP problem, exponentially many inequalities may be needed.

• The same situation occurs with the Gomory algorithm, it is generally not polynomial-time.

# This is the End!

Thank you for your attention!