Combinatorial Rounding

Section 7.3

1905026 - Wasif Hamid

Department of Computer Science and Engineering
Bangladesh University of Engineering and Technology

17 November 2024

Presentation Outline

Table of Contents

Combinatorial Rounding

- To efficiently solve Integer Linear Programs, we can relax them into linear programs that permit real values, making them easier to solve. However, this may result in non-feasible solutions.

Combinatorial Rounding

- To efficiently solve Integer Linear Programs, we can relax them into linear programs that permit real values, making them easier to solve. However, this may result in non-feasible solutions.
- Combinatorial Rounding applies various rounding strategies to convert these optimal real-valued solutions into integer values, yielding feasible solutions that also provide good approximation ratios.

Table of Contents

Minimum Weight Vertex Cover

A vertex cover problem with weights on vertices and we need to minimize the weight of the solution set. For a graph $G = (V, E)$ the ILP version of the problem can be defined below-

Min-WVC Integer Linear Program

$$
\begin{array}{lll}
\text{Minimize} & w_1x_1 + w_2x_2 + \cdots + w_nx_n & \\
\text{Subject to} & x_i + x_j \geq 1, & \text{for each } \{v_i, v_j\} \in E \\
 & x_i = 0 \text{ or } 1, & i = 1, 2, 3, \ldots, n
\end{array}
$$

Minimum Weight Vertex Cover

A vertex cover problem with weights on vertices and we need to minimize the weight of the solution set. For a graph $G = (V, E)$ the ILP version of the problem can be defined below-

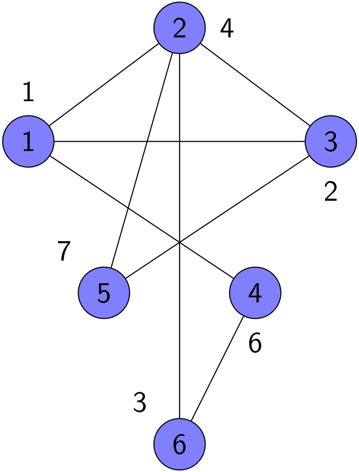
Min-WVC Integer Linear Program

Minimize	$w_1x_1 + w_2x_2 + \cdots + w_nx_n$	
Subject to	$x_i + x_j \geq 1,$	for each $\{v_i, v_j\} \in E$
	$x_i = 0$ or $1,$	$i = 1, 2, 3, \ldots, n$

Relaxation to Linear Program

$$x_i = 0 \text{ or } 1 \xrightarrow{\text{relaxation}} 0 \leq x_i \leq 1$$

Example



Example

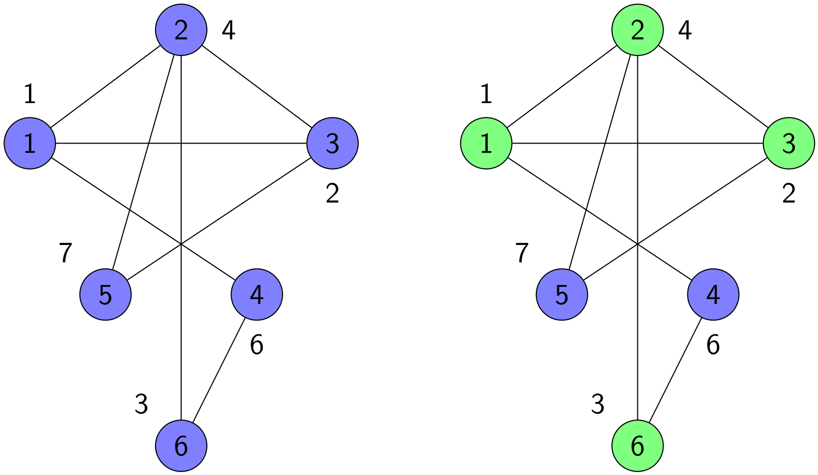


Figure: Minimum Weight Vertex Cover

Linear Programming Approximation Algorithm for Min-WVC

Input: A graph $G = (V, E)$ and a weight function $w : V \to \mathbb{N}$.

1. Convert the input into a 0-1 integer program, and construct the corresponding linear program.
2. Find an optimal solution x^* to the linear program (7.12).
3. For $i = 1, 2, \ldots, n$:

$$x_i^A = \begin{cases} 1, & \text{if } x_i^* \geq \frac{1}{2}, \\ 0, & \text{otherwise.} \end{cases}$$

4. Output x^A.

2-Approximation Ratio Proof

For each $\{v_i, v_j\} \in E$:

$$x_i^* + x_j^* \geq 1 \Rightarrow \text{at least one of } x_i^* \text{ or } x_j^* \geq \frac{1}{2}.$$

2-Approximation Ratio Proof

For each $\{v_i, v_j\} \in E$:

$$x_i^* + x_j^* \geq 1 \Rightarrow \text{at least one of } x_i^* \text{ or } x_j^* \geq \frac{1}{2}.$$

This implies that at least one of x_i^A or x_j^A is equal to 1, so x^A is a feasible solution to the integer program.

2-Approximation Ratio Proof

For each $\{v_i, v_j\} \in E$:

$$x_i^* + x_j^* \geq 1 \Rightarrow \text{at least one of } x_i^* \text{ or } x_j^* \geq \frac{1}{2}.$$

This implies that at least one of x_i^A or x_j^A is equal to 1, so x^A is a feasible solution to the integer program.

Furthermore,

$$\sum_{i=1}^{n} w_i x_i^A \leq 2 \sum_{i=1}^{n} w_i x_i^*.$$

This shows that the cost of x^A is at most twice the cost of the optimal solution x^* to the linear program.

2-Approximation Ratio Proof

For each $\{v_i, v_j\} \in E$:

$$x_i^* + x_j^* \geq 1 \Rightarrow \text{at least one of } x_i^* \text{ or } x_j^* \geq \frac{1}{2}.$$

This implies that at least one of x_i^A or x_j^A is equal to 1, so x^A is a feasible solution to the integer program.

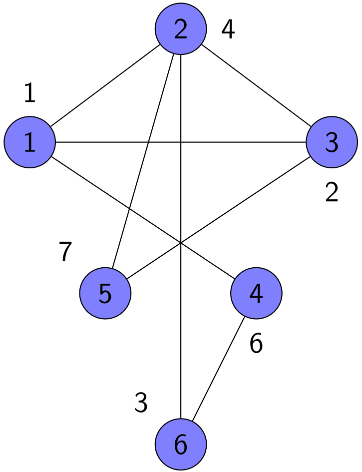
Furthermore,

$$\sum_{i=1}^{n} w_i x_i^A \leq 2 \sum_{i=1}^{n} w_i x_i^*.$$

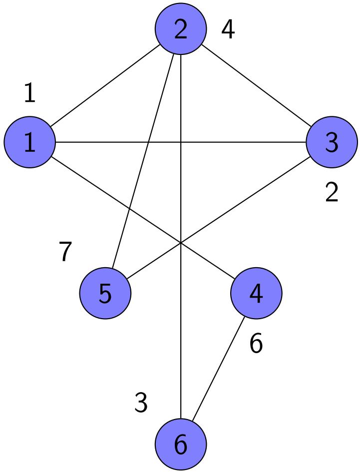
This shows that the cost of x^A is at most twice the cost of the optimal solution x^* to the linear program.

Therefore the proposed algorithm is a **polynomial-time 2-approximation** for Min-WVC.

Example



Example



Objective:

$$Z = x_1 + 4x_2 + 2x_3 + 6x_4 + 7x_5 + 3x_6$$

Constraints:

$$\begin{gathered}
x_1 + x_2 \geq 1, \\
x_1 + x_3 \geq 1, \\
x_1 + x_4 \geq 1, \\
x_2 + x_3 \geq 1, \\
x_2 + x_5 \geq 1, \\
x_2 + x_6 \geq 1, \\
x_3 + x_5 \geq 1 \\
x_4 + x_6 \geq 1 \\
x_1, x_2, x_3, x_4, x_5, x_6 \geq 0
\end{gathered}$$

Example(Cont.)

Solving the linear program we get the following:

$$min(Z) = 10$$
$$\text{for}$$
$$x^* = [1, 1, 1, 0, 0, 1]$$

The solution is already integers so we do not need to round up or down.

So, the solution set is

$$C = \{1, 2, 3, 6\}$$

Example(Cont.)

Solving the linear program we get the following:

$$min(Z) = 10$$
$$\text{for}$$
$$x^* = [1, 1, 1, 0, 0, 1]$$

The solution is already integers so we do not need to round up or down.

So, the solution set is

$$C = \{1, 2, 3, 6\}$$

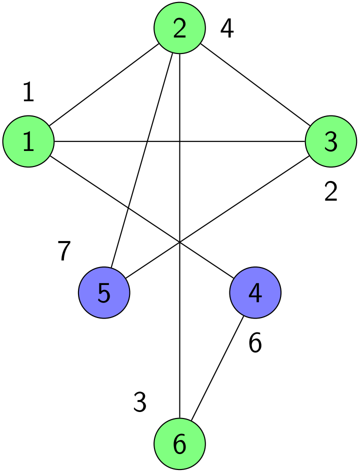


Table of Contents

Minimum 2-Satisfiability

Problem Statement

Given a Boolean formula in 2-CNF, determine whether it is satisfiable and, if it is, find a satisfying assignment that contains a minimum number of true variables.

Min-2SAT Linear Program

$$
\begin{array}{lll}
\text{Minimize} & x_1 + x_2 + x_3 + \cdots + x_n & \\
\text{Subject to} & x_i + x_j \geq 1, & \text{for each clause}\{x_i \vee x_j\} \text{ in } F \\
 & (1 - x_i) + x_j \geq 1, & \text{for each clause}\{\bar{x}_i \vee x_j\} \text{ in } F \\
 & (1 - x_i) + (1 - x_j) \geq 1, & \text{for each clause}\{\bar{x}_i \vee \bar{x}_j\} \text{ in } F \\
 & 0 \leq x_i \leq 1, & i = 1, 2, 3, \ldots, n
\end{array}
$$

Linear Programming Approximation Algorithm for Min-2SAT

Step 1: Convert formula F into a linear program and find an optimal solution x^* for it.

Step 2: **for** $i \leftarrow 1$ to n **do**

  if $x_i^* > \frac{1}{2}$ **then**

    $x_i^A \leftarrow 1$

  else if $x_i^* < \frac{1}{2}$ **then**

    $x_i^A \leftarrow 0$

  end if

end for

Linear Programming Approximation Algorithm for Min-2SAT

Step 1: Convert formula F into a linear program and find an optimal solution x^* for it.

Step 2: **for** $i \leftarrow 1$ to n **do**
  if $x_i^* > \frac{1}{2}$ **then**
    $x_i^A \leftarrow 1$
  else if $x_i^* < \frac{1}{2}$ **then**
    $x_i^A \leftarrow 0$
  end if
end for

Step 3: Let F_1 be the collection of all clauses both of whose two variables have x^* value equal to $\frac{1}{2}$, and let $J \leftarrow \{j \mid 1 \leq j \leq n, x_j \text{ is in } F_1\}$.

Step 4: **for** $i \leftarrow 1$ to n **do**
  if $x_i^* = \frac{1}{2}$ and $i \notin J$ **then**
    $x_i^A \leftarrow 0$
  end if
end for

Linear Programming Approximation Algorithm for Min-2SAT

Step 1: Convert formula F into a linear program and find an optimal solution x^* for it.

Step 2: **for** $i \leftarrow 1$ to n **do**
  if $x_i^* > \frac{1}{2}$ **then**
    $x_i^A \leftarrow 1$
  else if $x_i^* < \frac{1}{2}$ **then**
    $x_i^A \leftarrow 0$
  end if
end for

Step 3: Let F_1 be the collection of all clauses both of whose two variables have x^* value equal to $\frac{1}{2}$, and let $J \leftarrow \{j \mid 1 \leq j \leq n, x_j \text{ is in } F_1\}$.

Step 4: **for** $i \leftarrow 1$ to n **do**
  if $x_i^* = \frac{1}{2}$ and $i \notin J$ **then**
    $x_i^A \leftarrow 0$
  end if
end for

Step 5: **if** F_1 is satisfiable **then**
  Let x_J^A be a satisfying assignment for F_1 and output x^A.
else
  Output "F is not satisfiable."
end if

Satisfiability and Performance Ratio of the algorithm

Satisfiability of Clauses

- By step (5), every clause in F_1 is satisfied by x^A.
- For a clause $(x_i \vee x_j)$ not in F_1:
  - $x_i^* + x_j^* \geq 1$ implies $x_i^* > \frac{1}{2}$ or $x_j^* > \frac{1}{2}$.
  - By step (2), either $x_i^A = 1$ or $x_j^A = 1$, ensuring the clause $(x_i \vee x_j)$ is satisfied.
- The same reasoning applies to other types of clauses, such as $(x_i \vee \bar{x}_j)$ or $(\bar{x}_i \vee \bar{x}_j)$.

Satisfiability and Performance Ratio of the algorithm

Satisfiability of Clauses

- By step (5), every clause in F_1 is satisfied by x^A.
- For a clause $(x_i \vee x_j)$ not in F_1:
  - $x_i^* + x_j^* \geq 1$ implies $x_i^* > \frac{1}{2}$ or $x_j^* > \frac{1}{2}$.
  - By step (2), either $x_i^A = 1$ or $x_j^A = 1$, ensuring the clause $(x_i \vee x_j)$ is satisfied.
- The same reasoning applies to other types of clauses, such as $(x_i \vee \bar{x}_j)$ or $(\bar{x}_i \vee \bar{x}_j)$.

Performance Ratio

- For each $i = 1, 2, \ldots, n$, we have $x_i^A \leq 2x_i^*$.
- Thus, x^A is an approximation with a performance ratio ≤ 2.

Polynomial 2-SAT Algorithm

Input: A 2-CNF formula F_1 over variables $x_1, x_2, \ldots, x_n$.

1. Construct a digraph $G(F_1) = (V, E)$ as follows:

$$V \leftarrow \{x_i, \bar{x}_i \mid 1 \leq i \leq n\},$$

$$E \leftarrow \{(\neg y_i, y_j), (\neg y_j, y_i) \mid (y_i \vee y_j) \text{ is a clause in } F_1\},$$

where y_i denotes a literal x_i or $\bar{x}_i$.

Polynomial 2-SAT Algorithm

Input: A 2-CNF formula F_1 over variables $x_1, x_2, \ldots, x_n$.

1. Construct a digraph $G(F_1) = (V, E)$ as follows:

$$V \leftarrow \{x_i, \bar{x}_i \mid 1 \leq i \leq n\},$$

$$E \leftarrow \{(\neg y_i, y_j), (\neg y_j, y_i) \mid (y_i \vee y_j) \text{ is a clause in } F_1\},$$

where y_i denotes a literal x_i or $\bar{x}_i$.

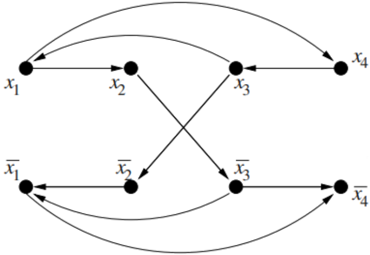


Figure: Digraph for
$F_1 = (\neg x_1 \vee x_2) \wedge (\neg x_2 \vee \neg x_3) \wedge (\neg x_1 \vee x_3) \wedge (x_3 \vee \neg x_4) \wedge (x_1 \vee \neg x_4)$

Polynomial 2-SAT Algorithm

Input: A 2-CNF formula F_1 over variables $x_1, x_2, \ldots, x_n$.

1. Construct a digraph $G(F_1) = (V, E)$ as follows:

$$V \leftarrow \{x_i, \bar{x}_i \mid 1 \leq i \leq n\},$$

$$E \leftarrow \{(\neg y_i, y_j), (\neg y_j, y_i) \mid (y_i \vee y_j) \text{ is a clause in } F_1\},$$

   where y_i denotes a literal x_i or $\bar{x}_i$.

2. **For** $i \leftarrow 1$ to n **do**

   If vertices x_i and $\bar{x}_i$ are strongly connected, **then** output "F_1 is not satisfiable" and halt.

Polynomial 2-SAT Algorithm

Input: A 2-CNF formula F_1 over variables $x_1, x_2, \ldots, x_n$.

1. Construct a digraph $G(F_1) = (V, E)$ as follows:

$$V \leftarrow \{x_i, \bar{x}_i \mid 1 \leq i \leq n\},$$

$$E \leftarrow \{(\neg y_i, y_j), (\neg y_j, y_i) \mid (y_i \vee y_j) \text{ is a clause in } F_1\},$$

   where y_i denotes a literal x_i or $\bar{x}_i$.

2. **For** $i \leftarrow 1$ to n **do**

   If vertices x_i and $\bar{x}_i$ are strongly connected, **then** output "F_1 is not satisfiable" and halt.

3. **For** $i \leftarrow 1$ to n **do**
   - **If** there is a path from x_i to $\bar{x}_i$, **then** for each literal y_j that is reachable from $\bar{x}_i$, set $\tau(y_j) \leftarrow 1$.
   - **If** there is a path from $\bar{x}_i$ to x_i, **then** for each literal y_j that is reachable from x_i, set $\tau(y_j) \leftarrow 1$.

Polynomial 2-SAT Algorithm

Input: A 2-CNF formula F_1 over variables $x_1, x_2, \ldots, x_n$.

1. Construct a digraph $G(F_1) = (V, E)$ as follows:

$$V \leftarrow \{x_i, \bar{x}_i \mid 1 \leq i \leq n\},$$

$$E \leftarrow \{(\neg y_i, y_j), (\neg y_j, y_i) \mid (y_i \vee y_j) \text{ is a clause in } F_1\},$$

   where y_i denotes a literal x_i or $\bar{x}_i$.

2. **For** $i \leftarrow 1$ to n **do**

   If vertices x_i and $\bar{x}_i$ are strongly connected, **then** output "F_1 is not satisfiable" and halt.

3. **For** $i \leftarrow 1$ to n **do**
   - **If** there is a path from x_i to $\bar{x}_i$, **then** for each literal y_j that is reachable from $\bar{x}_i$, set $\tau(y_j) \leftarrow 1$.
   - **If** there is a path from $\bar{x}_i$ to x_i, **then** for each literal y_j that is reachable from x_i, set $\tau(y_j) \leftarrow 1$.

4. **For** $i \leftarrow 1$ to n **do**
   - **If** $\tau(x_i)$ is undefined, **then** for each literal y_j that is reachable from x_i, set $\tau(y_j) \leftarrow 1$.

Polynomial 2-SAT Algorithm

Input: A 2-CNF formula F_1 over variables $x_1, x_2, \ldots, x_n$.

1. Construct a digraph $G(F_1) = (V, E)$ as follows:

$$V \leftarrow \{x_i, \bar{x}_i \mid 1 \leq i \leq n\},$$

$$E \leftarrow \{(\neg y_i, y_j), (\neg y_j, y_i) \mid (y_i \vee y_j) \text{ is a clause in } F_1\},$$

   where y_i denotes a literal x_i or $\bar{x}_i$.

2. **For** $i \leftarrow 1$ to n **do**

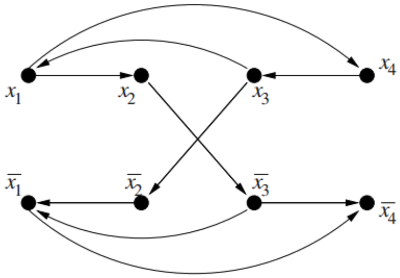
   If vertices x_i and $\bar{x}_i$ are strongly connected, **then** output "F_1 is not satisfiable" and halt.

3. **For** $i \leftarrow 1$ to n **do**
   - **If** there is a path from x_i to $\bar{x}_i$, **then** for each literal y_j that is reachable from $\bar{x}_i$, set $\tau(y_j) \leftarrow 1$.
   - **If** there is a path from $\bar{x}_i$ to x_i, **then** for each literal y_j that is reachable from x_i, set $\tau(y_j) \leftarrow 1$.

4. **For** $i \leftarrow 1$ to n **do**
   - **If** $\tau(x_i)$ is undefined, **then** for each literal y_j that is reachable from x_i, set $\tau(y_j) \leftarrow 1$.

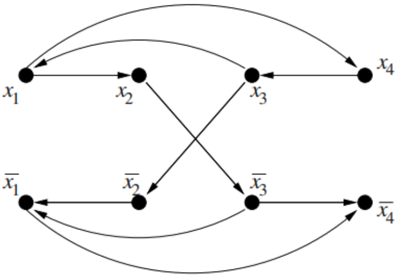
5. **Output** τ.

Example of 2-SAT Assignment



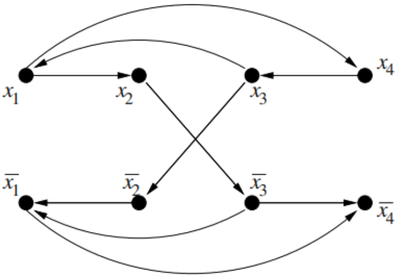
- Given $F_1 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_4)$, we constructed the above digraph.

Example of 2-SAT Assignment



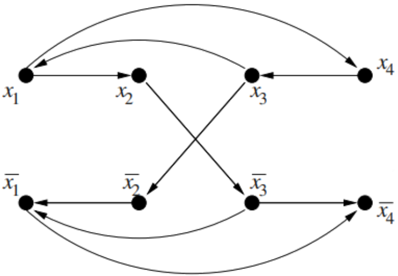
- Given $F_1 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_4)$, we constructed the above digraph.
- None of any x_i and $\bar{x}_i$ pair are strongly connected. So we continue to step 3.

Example of 2-SAT Assignment



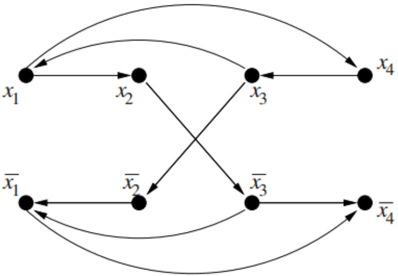
- Given $F_1 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_4)$, we constructed the above digraph.
- At the start $\tau = [-, -, -, -]$
  - $\bar{x}_1$ is reachable from x_1.
  - $\bar{x}_4$ is reachable from $\bar{x}_1$:
  - Using Step 3.1, Set $\bar{x}_4 = 1$, so $x_4 = 0$.
  - No other negation of a literal reachable from itself gives us more assignments.

Example of 2-SAT Assignment



- Given $F_1 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_4)$, we constructed the above digraph.
- After previous step $\tau = [-, -, -, 0]$
  - x_1 is still undefined.
  - $x_2, \bar{x}_3, \bar{x}_1$ is reachable from x_1:
  - Set $x_2 = 1$, $\bar{x}_3 = 1$ and $\bar{x}_1 = 1$, so we get $x_2 = 1, x_3 = 0$ and $x_4 = 0$.

Example of 2-SAT Assignment



- Given $F_1 = (\bar{x}_1 \vee x_2) \wedge (\bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_3) \wedge (x_3 \vee \bar{x}_4) \wedge (x_1 \vee \bar{x}_4)$, we constructed the above digraph.
- Resulting assignment:

$$\tau(x_1) = 0, \quad \tau(x_2) = 1, \quad \tau(x_3) = 0, \quad \tau(x_4) = 0$$

- This assignment satisfies all the clauses in F_1

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:

$$\tau(y) = 1 \implies \tau(z) = 1.$$

- This property extends to all pairs y, z where there is a path from y to z.

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).
- If there is a path from $y \to z$, then there is a path from $\bar{z} \to \bar{y}$. This ensures that no variable is assigned both 1 and 0 at the same time in step 3 and step 4.

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).
- If there is a path from $y \to z$, then there is a path from $\bar{z} \to \bar{y}$. This ensures that no variable is assigned both 1 and 0 at the same time in step 3 and step 4.
  - Suppose a variable w is assigned both. So, $\tau(w) = 1$, so there must be path $\bar{u} \to u \to w$. So there must be a path from $\bar{w}$ to $\bar{u}$.

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).
- If there is a path from $y \rightarrow z$, then there is a path from $\bar{z} \rightarrow \bar{y}$. This ensures that no variable is assigned both 1 and 0 at the same time in step 3 and step 4.
  - Suppose a variable w is assigned both. So, $\tau(w) = 1$, so there must be path $\bar{u} \rightarrow u \rightarrow w$. So there must be a path from $\bar{w}$ to $\bar{u}$.
  - Again, $\tau(\bar{w}) = 1$, so there must be path $\bar{v} \rightarrow v \rightarrow \bar{w}$. So there must be a path from w to $\bar{v}$.

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).
- If there is a path from $y \to z$, then there is a path from $\bar{z} \to \bar{y}$. This ensures that no variable is assigned both 1 and 0 at the same time in step 3 and step 4.
  - Suppose a variable w is assigned both. So, $\tau(w) = 1$, so there must be path $\bar{u} \to u \to w$. So there must be a path from $\bar{w}$ to $\bar{u}$.
  - Again, $\tau(\bar{w}) = 1$, so there must be path $\bar{v} \to v \to \bar{w}$. So there must be a path from w to $\bar{v}$.
  - So we get a path, $w \to \bar{v} \to \bar{w} \to \bar{u} \to w$ which is a cycle. If that exists step 2 would have already discarded the problem.

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).
- If there is a path from $y \rightarrow z$, then there is a path from $\bar{z} \rightarrow \bar{y}$. This ensures that no variable is assigned both 1 and 0 at the same time in step 3 and step 4.
- Each clause $(y_i \vee y_j)$ in F_1 generates two edges $(\bar{y}_i, y_j)$ and $(\bar{y}_j, y_i)$ in E. From steps (3) and (4), we see that it is not possible to assign $\tau(y_i) = \tau(y_j) = 0$

Correctness of the Algorithm

- If there is an edge (y, z) in E, any satisfying assignment τ must satisfy:
$$\tau(y) = 1 \implies \tau(z) = 1.$$
- This property extends to all pairs y, z where there is a path from y to z.
- Thus, if any variable x_i and its negation $\bar{x}_i$ are in the same strongly connected component F_1 is **unsatisfiable**. So, the Algorithm terminates correctly in Step (2).
- If there is a path from $y \to z$, then there is a path from $\bar{z} \to \bar{y}$. This ensures that no variable is assigned both 1 and 0 at the same time in step 3 and step 4.
- Each clause $(y_i \vee y_j)$ in F_1 generates two edges $(\bar{y}_i, y_j)$ and $(\bar{y}_j, y_i)$ in E. From steps (3) and (4), we see that it is not possible to assign $\tau(y_i) = \tau(y_j) = 0$
- τ must be a satisfying assignment for F_1.

Table of Contents

Scheduling on Unrelated Parallel Machines

Problem Statement

Given n jobs, m machines and, for each $1 \leq i \leq m$ and each $1 \leq j \leq n$, the amount of time t_{ij} required for the ith machine to process the jth job, find the schedule for all n jobs on these m machines that minimizes the maximum processing time over all machines.

Scheduling on Unrelated Parallel Machines

Problem Statement

Given n jobs, m machines and, for each $1 \leq i \leq m$ and each $1 \leq j \leq n$, the amount of time t_{ij} required for the ith machine to process the jth job, find the schedule for all n jobs on these m machines that minimizes the maximum processing time over all machines.

SCHEDULE-UPM Linear Program

$$
\begin{aligned}
\text{Minimize} \quad & t \\
\text{Subject to} \quad & \sum_{i=1}^{m} x_{ij} = 1, && 1 \leq j \leq n, \\
& \sum_{j=1}^{n} x_{ij} t_{ij} \leq t, && 1 \leq i \leq m, \\
& 0 \leq x_{ij} \leq 1, && 1 \leq i \leq m, 1 \leq j \leq n.
\end{aligned}
$$

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x^*_{ij} < 1\}$ and $M = \{1, \ldots, m\}$.

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x^*_{ij} < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x^*_{ij} < 1\}.$$

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x^*_{ij} < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x^*_{ij} < 1\}.$$

- We need to show, H contains a matching covering J.For that, it suffices to show each connected component of H contains a matching covering all jobs in the component.

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x_{ij}^* < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x_{ij}^* < 1\}.$$

- We need to show, H contains a matching covering J.For that, it suffices to show each connected component of H contains a matching covering all jobs in the component.
- Consider a connected component $H' = (M', J', E')$ of H.

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x^*_{ij} < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x^*_{ij} < 1\}.$$

- We need to show, H contains a matching covering J.For that, it suffices to show each connected component of H contains a matching covering all jobs in the component.
- Consider a connected component $H' = (M', J', E')$ of H.
- Fix $x_{ij} = x^*_{ij}$ for $i \notin M'$ or $j \notin J'$.

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x^*_{ij} < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x^*_{ij} < 1\}.$$

- We need to show, H contains a matching covering J.For that, it suffices to show each connected component of H contains a matching covering all jobs in the component.
- Consider a connected component $H' = (M', J', E')$ of H.
- Fix $x_{ij} = x^*_{ij}$ for $i \notin M'$ or $j \notin J'$.
- Remaining variables form a new LP with extreme point

$$x' = (x^*_{ij})_{i \in M', j \in J'}$$

.

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x_{ij}^* < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x_{ij}^* < 1\}.$$

- We need to show, H contains a matching covering J.For that, it suffices to show each connected component of H contains a matching covering all jobs in the component.
- Consider a connected component $H' = (M', J', E')$ of H.
- Fix $x_{ij} = x_{ij}^*$ for $i \notin M'$ or $j \notin J'$.
- Remaining variables form a new LP with extreme point

$$x' = (x_{ij}^*)_{i \in M', j \in J'}$$

- x' is determined by $|M'||J'|$ active constraints(constraint that are at equality).

Rounding Using Bipartite Graphs

- Let $J = \{j \mid 0 < x^*_{ij} < 1\}$ and $M = \{1, \ldots, m\}$.
- Define $H = (M, J, E)$, where:

$$E = \{(i, j) \mid 0 < x^*_{ij} < 1\}.$$

- We need to show, H contains a matching covering J.For that, it suffices to show each connected component of H contains a matching covering all jobs in the component.
- Consider a connected component $H' = (M', J', E')$ of H.
- Fix $x_{ij} = x^*_{ij}$ for $i \notin M'$ or $j \notin J'$.
- Remaining variables form a new LP with extreme point

$$x' = (x^*_{ij})_{i \in M', j \in J'}$$

- x' is determined by $|M'||J'|$ active constraints(constraint that are at equality).
- At most $|M'| + |J'|$ non-integral components, implying H' has at most $|M'| + |J'|$ edges.

Case 1: H' is a Tree

- Root the tree at any vertex $r \in J'$.
- A vertex $j \in J'$ cannot be a leaf, as $\sum_{i \in M'} x_{ij} = 1$ implies at least two edges incident on j.
- For each $j \in J'$, match it to one of its children.

Matching in H'

Case 1: H' is a Tree

- Root the tree at any vertex $r \in J'$.
- A vertex $j \in J'$ cannot be a leaf, as $\sum_{i \in M'} x_{ij} = 1$ implies at least two edges incident on j.
- For each $j \in J'$, match it to one of its children.

Case 2: H' is a Tree Plus an Edge

- The extra edge forms a cycle and H' is the cycle plus some trees growing out.
- Match all vertices on the cycle.(This is always possible as H' is bipartite guaranteeing even length cycle).
- Contract the cycle into a root point $\rightarrow$ remaining graph becomes a tree.
- Match internal vertices as in Case 1.

Final Approximation Strategy

Steps:

- Assign jobs with $x_{ij}^* = 1$ directly to machine i.
- For partially assigned jobs, find a matching in H and assign jobs accordingly.

Final Approximation Strategy

Steps:

- Assign jobs with $x_{ij}^* = 1$ directly to machine i.
- For partially assigned jobs, find a matching in H and assign jobs accordingly.

Approximation Bound:

$$\text{Makespan} \leq \text{opt} + \max_{1 \leq i \leq m, 1 \leq j \leq n} t_{ij},$$

where opt is the minimum makespan. But we can't bound $\max t_{ij}$ by a constant times opt as it can be much greater than opt.

Final Approximation Strategy

Steps:

- Assign jobs with $x_{ij}^* = 1$ directly to machine i.
- For partially assigned jobs, find a matching in H and assign jobs accordingly.

Approximation Bound:

$$\text{Makespan} \ \leq \text{opt} + \max_{1\leq i\leq m, 1\leq j\leq n} t_{ij},$$

where opt is the minimum makespan. But we can't bound $\max t_{ij}$ by a constant times opt as it can be much greater than opt.

Observation: If $t_{ij} > \text{opt}$, job j cannot be assigned to machine i in the optimal solution. Therefore, we can prune the variable x_{ij} from the LP, and expect to get the same solution

Pruning and LP-Based Approximation

- If $t_{ij} > T$, remove the variable x_{ij} from the LP effectively creating a bound T to find feasible solutions while ensuring $T \geq$ opt.

Pruning and LP-Based Approximation

- If $t_{ij} > T$, remove the variable x_{ij} from the LP effectively creating a bound T to find feasible solutions while ensuring $T \geq \text{opt}$.
- Solve the following LP (7.16) to find the minimum T:

$$
\begin{aligned}
\text{Minimize} \quad & t \\
\text{Subject to} \quad & \sum_{1 \leq i \leq m, t_{ij} \leq T} x_{ij} = 1, && 1 \leq j \leq n, \\
& \sum_{1 \leq j \leq n, t_{ij} \leq T} x_{ij} t_{ij} \leq t, && 1 \leq i \leq m, \\
& 0 \leq x_{ij} \leq 1, && 1 \leq i \leq m, 1 \leq j \leq n.
\end{aligned}
$$

Pruning and LP-Based Approximation

- If $t_{ij} > T$, remove the variable x_{ij} from the LP effectively creating a bound T to find feasible solutions while ensuring $T \geq \text{opt}$.
- Solve the following LP (7.16) to find the minimum T:

$$
\begin{array}{lll}
\text{Minimize} & t & \\
\text{Subject to} & \displaystyle\sum_{1 \leq i \leq m, t_{ij} \leq T} x_{ij} = 1, & 1 \leq j \leq n, \\
& \displaystyle\sum_{1 \leq j \leq n, t_{ij} \leq T} x_{ij} t_{ij} \leq t, & 1 \leq i \leq m, \\
& 0 \leq x_{ij} \leq 1, & 1 \leq i \leq m, 1 \leq j \leq n.
\end{array}
$$

- Use bisection to determine the minimum T^* for feasibility.

Pruning and LP-Based Approximation

- If $t_{ij} > T$, remove the variable x_{ij} from the LP effectively creating a bound T to find feasible solutions while ensuring $T \geq \text{opt}$.
- Solve the following LP (7.16) to find the minimum T:

$$
\begin{aligned}
\text{Minimize} \quad & t \\
\text{Subject to} \quad & \sum_{1 \leq i \leq m, t_{ij} \leq T} x_{ij} = 1, && 1 \leq j \leq n, \\
& \sum_{1 \leq j \leq n, t_{ij} \leq T} x_{ij} t_{ij} \leq t, && 1 \leq i \leq m, \\
& 0 \leq x_{ij} \leq 1, && 1 \leq i \leq m, 1 \leq j \leq n.
\end{aligned}
$$

- Use bisection to determine the minimum T^* for feasibility.
- $T^* \leq \text{opt}$, and $t_{ij} \leq T^*$ for all $x^*_{ij} > 0$.

Pruning and LP-Based Approximation

- If $t_{ij} > T$, remove the variable x_{ij} from the LP effectively creating a bound T to find feasible solutions while ensuring $T \geq \text{opt}$.
- Solve the following LP (7.16) to find the minimum T:

$$
\begin{aligned}
\text{Minimize} \quad & t \\
\text{Subject to} \quad & \sum_{1 \le i \le m, t_{ij} \le T} x_{ij} = 1, && 1 \le j \le n, \\
& \sum_{1 \le j \le n, t_{ij} \le T} x_{ij} t_{ij} \le t, && 1 \le i \le m, \\
& 0 \le x_{ij} \le 1, && 1 \le i \le m, 1 \le j \le n.
\end{aligned}
$$

- Use bisection to determine the minimum T^* for feasibility.
- $T^* \leq \text{opt}$, and $t_{ij} \leq T^*$ for all $x^*_{ij} > 0$.
- This yields a **2-approximation** for SCHEDULE-UPM.

Thank You!

Any Questions?