Vertex Cover Problems

Consider a graph $G = (V, E)$
$S \subseteq V$ is a vertex cover if

$$\forall \{u, v\} \in E : u \in S \lor v \in S$$

minimum vertex cover (MIN-VCP):
find a vertex cover $S$ that minimizes $|S|$.
Motivation

☐ This problem has many applications

☐ Example: placing ATMs in a city

☐ Each additional ATM costs money

☐ Want to have an ATM in every street (block, district)

☐ Where should they be placed so that we need as little ATMs as possible?
Greedy Algorithm

Function greedyVC(V, E)

\[ C := \emptyset \]

While \( E \neq \emptyset \) do

Select any \( \{u, v\} \in E \)

\[ C := C \cup \{u, v\} \]

\[ C := C \cup \{u, v\} \]

Remove all edges incident to \( u \) or \( v \) from \( E \)

Return \( C \)

Exercise: explain how to implement the algorithm to run in time \( \mathcal{O}(|V| + |E|) \)
Greedy Algorithm

Function greedyVC(V, E)

$C := \emptyset$

while $E \neq \emptyset$ do

select any $\{u, v\} \in E$

$C := C \cup \{u, v\}$

remove all edges incident to $u$ or $v$ from $E$

return $C$

Exercise: explain how to implement the algorithm to run in time $\Theta(|V| + |E|)$
Greedy Algorithm

Function greedyVC(V, E)

\[ C := \emptyset \]

while \( E \neq \emptyset \) do

select any \( \{u, v\} \in E \)

\[ C := C \cup \{u, v\} \]

remove all edges incident to \( u \) or \( v \) from \( E \)

return \( C \)

Exercise: explain how to implement the algorithm to run in time \( \mathcal{O}(|V| + |E|) \)
Greedy Algorithm

Function greedyVC(V, E)

\[ C := \emptyset \]

while \( E \neq \emptyset \) do

select any \( \{u, v\} \in E \)

\[ C := C \cup \{u, v\} \]

remove all edges incident to \( u \) or \( v \) from \( E \)

return \( C \)

Exercise: explain how to implement the algorithm to run in time \( \mathcal{O}(|V| + |E|) \)
Theorem 1. Algorithm greedyVC computes a two-approximation of MIN-VCP.

Proof. Correctness: trivial since only covered edges are removed.

Quality: Let $A$ denote the set of edges selected by greedyVC. We have $|C| = 2|A|$. $A$ is a matching, i.e., no node covers two edges in $A$. Hence, any vertex cover contains at least one node from each edge in $A$, i.e., $\text{opt} \geq |A|$. \qed
Weighted Vertex Cover

Consider a graph $G = (V, E)$
$S \subseteq V$ is a vertex cover if

$$\forall \{u, v\} \in E : u \in S \lor v \in S$$

minimum WEIGHT vertex cover

(WEIGHT-VCP):
find a vertex cover $S$ that minimizes

$$\sum_{v \in S} c(s)$$
0-1 ILP Formulation

Assume \( V = \{1, \ldots, n\} \)

Variables: \( x_v = 1 \) iff \( v \in V \)

\textbf{minimize} \( c \cdot x \)

subject to

\( \forall \{u, v\} \in E : x_u + x_v \geq 1 \)

\( \forall v \in V : x_v \in \{0, 1\} \)
0-1 ILP Formulation

Assume $V = \{1, \ldots, n\}$

Variables: $x_v = 1$ iff $v \in V$

minimize $c \cdot x$

subject to

$\forall \{u, v\} \in E : x_u + x_v \geq 1$

$\forall v \in V : x_v \in \{0, 1\}$

Linear Relaxation

Assume $V = \{1, \ldots, n\}$

Variables: $x_v = 1$ iff $v \in V$

minimize $c \cdot x$

subject to

$\forall \{u, v\} \in E : x_u + x_v \geq 1$

$\forall v \in V : x_v \geq 0$
0-1 ILP Formulation

Assume $V = \{1, \ldots, n\}$

Variables: $x_v = 1$ iff $v \in V$

minimize $c \cdot x$

subject to

$\forall \{u, v\} \in E : x_u + x_v \geq 1$

$\forall v \in V : x_v \in \{0, 1\}$

Linear Relaxation

Assume $V = \{1, \ldots, n\}$

Variables: $x_v = 1$ iff $v \in V$

minimize $c \cdot x$

subject to

$\forall \{u, v\} \in E : x_u + x_v \geq 1$

$\forall v \in V : x_v \geq 0$

LP Rounding Algorithm for WEIGHT-VCP

Function $lpWeightedVC(V, E, c)$

$x := lpSolve(linearRelaxation(V, E, c))$

return $\{v \in V : x_v \geq 1/2\}$
Theorem 2. Algorithm \textit{lpWeightedVC} computes a two-approximation of \textit{WEIGHT-VCP}.

Correctness:
Consider any edge \(\{u, v\} \in E\).
We have \(x_u + x_v \geq 1\),
hence, \(\max \{x_u, x_v\} \geq 1/2\),
i.e., rounding will put at least one of \(\{u, v\}\) into the output.
Theorem 2. Algorithm $lpWeightedVC$ computes a two-approximation of WEIGHT-VCP.

Quality: Let

\(~x\) := the solution computed by $lpWeightedVC$

\(~x^*\) := the optimal solution, and

\(~\bar{x}\) := the optimal solution of the linear relaxation

\[\begin{align*}
  c \cdot x &= \sum_{\bar{x}_i \geq 1/2} c_i \\
  c \cdot x^* &\leq 2c \cdot \bar{x}
\end{align*}\]
Theorem 2. Algorithm $lpWeightedVC$ computes a two-approximation of $WEIGHT-VCP$.

Quality: Let

$x :=$ the solution computed by $lpWeightedVC$

$x^* :=$ the optimal solution, and

$ar{x} :=$ the optimal solution of the linear relaxation

\[
c \cdot x = \sum_{\bar{x}_i \geq 1/2} c_i \leq \sum_{\bar{x}_i \geq 1/2} 2\bar{x}_i c_i
\]
Theorem 2. Algorithm \textit{lpWeightedVC} computes a two-approximation of \textit{WEIGHT-VCP}.

Quality: Let

\begin{align*}
\mathbf{x} &: \text{ the solution computed by \textit{lpWeightedVC}} \\
\mathbf{x}^* &: \text{ the optimal solution, and} \\
\bar{x} &: \text{ the optimal solution of the linear relaxation}
\end{align*}

\[
\mathbf{c} \cdot \mathbf{x} = \sum_{\bar{x}_i \geq 1/2} c_i \leq \sum_{\bar{x}_i \geq 1/2} 2\bar{x}_i c_i \leq 2 \sum_{i=1}^n \bar{x}_i c_i
\]
Theorem 2. Algorithm lpWeightedVC computes a two-approximation of WEIGHT-VCP.

Quality: Let
\[ x \] be the solution computed by lpWeightedVC
\[ x^* \] be the optimal solution, and
\[ \bar{x} \] be the optimal solution of the linear relaxation

\[
    c \cdot x = \sum_{\bar{x}_i \geq 1/2} c_i \leq \sum_{\bar{x}_i \geq 1/2} 2\bar{x}_i c_i \leq 2 \sum_{i=1}^{n} \bar{x}_i c_i = 2c \cdot \bar{x}
\]
Theorem 2. Algorithm \text{lpWeightedVC} computes a two-approximation of WEIGHT-VCP.

\textbf{Quality:} Let

- \(\mathbf{x} := \) the solution computed by \text{lpWeightedVC} \\
- \(\mathbf{x}^* := \) the optimal solution, and \\
- \(\bar{\mathbf{x}} := \) the optimal solution of the linear relaxation

\[
\mathbf{c} \cdot \mathbf{x} = \sum_{\bar{x}_i \geq 1/2} c_i \leq \sum_{\bar{x}_i \geq 1/2} 2\bar{x}_i c_i \leq 2 \sum_{i=1}^{n} \bar{x}_i c_i = 2\mathbf{c} \cdot \bar{\mathbf{x}} \leq 2\mathbf{c} \cdot \mathbf{x}^*
\]
Iterated Rounding

[Vazirani Section 23.2]

Function iteratedLpWeightedVC($V, E, c$)

\[ M := \emptyset \]

while $|E| > 0$ do

\[ x := \text{lpSolve}(\text{linearRelaxation}(V, E, c)) \]

let $v$ denote the node which maximizes $x_v$

\[ M := M \cup \{v\} \]

\[ V := V \setminus \{v\} \]

\[ E := E \setminus \{\{u, v\} \in E\} \]

return $M$
Iterated Rounding: Discussion

- Might give better solutions for many inputs
- No better approximation guarantees for VC
- Larger (still polynomial) execution time
- But: Resolving an LP is often quite fast
- Important technique for other problems
A Randomized Algorithm

[Ausielo et al. Section 5.1]

**Function** randWeightedVC(V, E, c)

\[
C := \emptyset
\]

while \( E \neq \emptyset \) do

select any \( \{v, t\} \in E \)

flip a coin with sides \( \{v, t\} \) and

\[
\Pr [v] = \frac{c_t}{c_v + c_t}
\]

\( x := \) upper side of coin

\( C := C \cup \{x\} \)

remove all edges incident to \( x \) from \( E \)

return \( C \)
Theorem 3. Algorithm randWeightedVC computes a vertex cover $x$ with $\mathbb{E}[c \cdot x] \leq 2c \cdot x^*$. 

Correctness: as for greedyVC.
Theorem: Algorithm randWeightedVC computes a vertex cover $x$ with $\mathbb{E}[c \cdot x] \leq 2c \cdot x^\ast$.

**Quality:** Define the random variables

$$X_v := \begin{cases} c_v & \text{if } v \in x \\ 0 & \text{otherwise} \end{cases} \quad (1)$$

$$X_{\{v,t\},v} := \begin{cases} c_v & \text{if } \{v,t\} \text{ is selected and } v \in x \\ 0 & \text{otherwise} \end{cases} \quad (2)$$

Note that $X_v = \sum_{\{t: \{v,t\} \in E\}} X_{\{v,t\},v}$.
Lemma 4. $\mathbb{E}[X_{\{v,t\},v}] = \mathbb{E}[X_{\{v,t\},t}]$

Proof.

$\mathbb{E}[X_{\{v,t\},v}] = c_v \mathbb{P}[\{v,t\} \text{ is selected}] \mathbb{P}[v \in x]$
Lemma 4. \( \mathbb{E}[X_{\{v,t\},v}] = \mathbb{E}[X_{\{v,t\},t}] \)

Proof.

\[
\mathbb{E}[X_{\{v,t\},v}] = c_v \mathbb{P}[\{v,t\} \text{ is selected }] \mathbb{P}[v \in x]
\]

\[
\mathbb{E}[X_{\{v,t\},v}] = c_v \mathbb{P}[\{v,t\} \text{ is selected }] \frac{c_t}{c_v + c_t}
\]
Lemma 4. \[ \mathbb{E}[X_{\{v,t\},v}] = \mathbb{E}[X_{\{v,t\},t}] \]

Proof.

\[
\mathbb{E}[X_{\{v,t\},v}] = c_v \mathbb{P}[\{v, t\} \text{ is selected }] \mathbb{P}[v \in \mathbf{x}]
\]

\[
\mathbb{E}[X_{\{v,t\},v}] = c_v \mathbb{P}[\{v, t\} \text{ is selected }] \frac{c_t}{c_v + c_t}
\]

\[
= c_t \mathbb{P}[\{v, t\} \text{ is selected }] \frac{c_v}{c_v + c_t}
\]

\[
= \mathbb{E}[X_{\{v,t\},t}]
\]
Lemma 5. $\sum_{v \not\in x^*} \mathbb{E}[X_v] \leq \sum_{t \in x^*} \mathbb{E}[X_t]$

Proof.

$$\sum_{v \not\in x^*} \mathbb{E}[X_v] = \sum_{v \not\in x^*} \mathbb{E} \left[ \sum_{\{t:\{v,t\} \in E\}} X_{\{v,t\},v} \right] \quad (X_v = \sum_{\{t:\{v,t\} \in E\}} X_{\{v,t\},v})$$
Lemma 5. \( \sum_{v \not\in x^*} \mathbb{E}[X_v] \leq \sum_{t \in x^*} \mathbb{E}[X_t] \)

Proof.

\[
\sum_{v \not\in x^*} \mathbb{E}[X_v] = \sum_{v \not\in x^*} \mathbb{E} \left[ \sum_{\{t: \{v,t\} \in E\}} X_{\{v,t\},v} \right] \quad (X_v = \sum_{\{t: \{v,t\} \in E\}} X_{\{v,t\},v})
\]

\[
= \sum_{v \not\in x^*} \sum_{\{t: \{v,t\} \in E\}} \mathbb{E}[X_{\{v,t\},v}] \quad \text{Linearity of } \mathbb{E}[\cdot]
\]
Lemma 5. \[ \sum_{v \not\in x^*} \mathbb{E}[X_v] \leq \sum_{t \in x^*} \mathbb{E}[X_t] \]

Proof.

\[ \sum_{v \not\in x^*} \mathbb{E}[X_v] = \sum_{v \not\in x^*} \mathbb{E} \left[ \sum_{\{t:\{v,t\} \in E\}} X_{\{v,t\},v} \right] \]

\[ = \sum_{v \not\in x^*} \sum_{\{t:\{v,t\} \in E\}} \mathbb{E}[X_{\{v,t\},v}] \quad \text{Linearity of } \mathbb{E}[\cdot] \]

\[ = \sum_{v \not\in x^*} \sum_{\{t:\{v,t\} \in E\}} \mathbb{E}[X_{\{v,t\},t}](*). \quad \text{Lemma 4} \]
Lemma 5. \[ \sum_{v \not\in x^*} \mathbb{E}[X_v] \leq \sum_{t \in x^*} \mathbb{E}[X_t] \]

Proof.

\[ \sum_{v \not\in x^*} \mathbb{E}[X_v] = \sum_{v \not\in x^*} \mathbb{E} \left[ \sum_{\{t : \{v,t\} \in E\}} X_{\{v,t\},v} \right] \]
\[ = \sum_{v \not\in x^*} \sum_{\{t : \{v,t\} \in E\}} \mathbb{E}[X_{\{v,t\},v}] \quad \text{Linearity of } \mathbb{E}[\cdot] \]
\[ = \sum_{v \not\in x^*} \sum_{\{t : \{v,t\} \in E\}} \mathbb{E}[X_{\{v,t\},t}] (*). \quad \text{Lemma 4} \]

But also \[ \sum_{t \in x^*} \mathbb{E}[X_t] = \sum_{t \in x^*} \sum_{\{v : \{v,t\} \in E\}} \mathbb{E}[X_{\{v,t\},t}] (**). \]

Every term in (*) shows up in (**).
Theorem: Algorithm randWeightedVC computes a vertex cover $x$ with $\mathbb{E}[c \cdot x] \leq 2c \cdot x^*$. 

Quality: (Finishing Up)

$$\sum_{v \in V} \mathbb{E}[X_v] = \sum_{v \notin x^*} \mathbb{E}[X_v] + \sum_{t \in x^*} \mathbb{E}[X_t]$$
Theorem: Algorithm randWeightedVC computes a vertex cover $x$ with $\mathbb{E}[c \cdot x] \leq 2c \cdot x^*$. 

**Quality: (Finishing Up)**

\[
\sum_{v \in V} \mathbb{E}[X_v] = \sum_{v \notin x^*} \mathbb{E}[X_v] + \sum_{t \in x^*} \mathbb{E}[X_t]
\]

**Lemma 5**

\[
\leq 2 \sum_{t \in x^*} \mathbb{E}[X_t]
\]
Theorem: Algorithm randWeightedVC computes a vertex cover \( \mathbf{x} \) with \( \mathbb{E}[c \cdot \mathbf{x}] \leq 2c \cdot \mathbf{x}^* \).

Quality: (Finishing Up)

\[
\sum_{v \in V} \mathbb{E}[X_v] = \sum_{v \notin \mathbf{x}^*} \mathbb{E}[X_v] + \sum_{t \in \mathbf{x}^*} \mathbb{E}[X_t]
\]

Lemma 5

\[
\leq 2 \sum_{t \in \mathbf{x}^*} \mathbb{E}[X_t]
\]

\[
\leq 2 \sum_{t \in \mathbf{x}^*} c_t \quad X_t = 0 \text{ or } X_t = c_t
\]
Theorem: Algorithm randWeightedVC computes a vertex cover $x$ with $E[c \cdot x] \leq 2c \cdot x^*$. 

**Quality: (Finishing Up)**

$$\sum_{v \in V} E[X_v] = \sum_{v \notin x^*} E[X_v] + \sum_{t \in x^*} E[X_t]$$

**Lemma 5**

$$\leq 2 \sum_{t \in x^*} E[X_t]$$

$$\leq 2 \sum_{t \in x^*} c_t \quad X_t = 0 \text{ or } X_t = c_t$$

$$= 2c \cdot x^*$$
More on Vertex Cover

- There are simple deterministic linear time 2-approximations. (Special case of set covering)

- Best known algorithm: ratio $2 - \Theta(1/\sqrt{\log n})$

- Fixed parameter algorithms: [Niedermeyer Rossmanith] find optimal solution in time $\mathcal{O}(kn + k^21.292^k)$ if $|x| \leq k$. Key idea: (clever) exhaustive search + problem reductions. Example: include nodes of degree $\geq k$. Include neighbors of degree 1 nodes
Scheduling on Unrelated Parallel Machines

[Vazirani Chapter 17]

\( J \): set of \( n \) jobs

\( M \): set of \( m \) machines

\( p_{ij} \): processing time of job \( j \) on machine \( i \)

\( x(j) \): Machine where job \( j \) is executed

\( L_i \): \( \sum_{\{j: x(j) = i\}} p_{ij} \), load of machine \( i \)

Objective: Minimize Makespan \( L_{\text{max}} = \max_i L_i \)
A Misguided ILP model

\[
\begin{align*}
\text{minimize } & \ t \\
\text{subject to } & \\
\forall j \in J : & \sum_{i \in M} x_{ij} = 1 \\
\forall i \in M : & \sum_{j \in J} x_{ij} p_{ij} \leq t \\
\forall i \in M, j \in J : & x_{ij} \in \{0, 1\}
\end{align*}
\]
The problem with this formulation

\[
\begin{align*}
\text{minimize} & \quad t \\
\text{subject to} & \\
\forall j \in J : & \sum_{i \in M} x_{ij} = 1 \\
\forall i \in M : & \sum_{j \in J} x_{ij} p_{ij} \leq t \\
\forall i \in M, j \in J : & x_{ij} \in \{0, 1\}
\end{align*}
\]

One Job, size \(m\) everywhere.
Linear relaxation: makespan 1
Optimal solution: makespan \(m\)

The linear relaxation is far away from the optimal solution and hence yields little useful information
LP-speak: integrality gap \(m\)
The problem with this formulation

minimize $t$ subject to
\[ \forall j \in J : \sum_{i \in M} x_{ij} = 1 \]
\[ \forall i \in M : \sum_{j \in J} x_{ij} p_{ij} \leq t \]
\[ \forall i \in M, j \in J : x_{ij} \in \{0, 1\} \]

In ILP, we always have $x_{ij} = 0$ if $p_{ij} > t$

This is lost in the linear relaxation: some $x_{ij}$ may get small values

We cannot add this constraint since it is not a linear constraint
A Refined LP Relaxation (Parametric Pruning)

guess makespan \( T \) e.g., binary search
feasible assignments: \( S_T := \{(i, j) : p_{ij} \leq T \} \)
A Refined LP Relaxation (Parametric Pruning)

**guess** makespan $T$  
e.g., binary search

feasible assignments: $S_T := \{(i, j) : p_{ij} \leq T\}$

\[
\text{LP}(T):
\begin{align*}
\forall j \in J : \quad & \sum_{\{i : (i, j) \in S_T\}} x_{ij} = 1 \\
\forall i \in M : \quad & \sum_{\{j : (i, j) \in S_T\}} x_{ij} p_{ij} \leq T \\
\forall (i, j) \in S_T : \quad & x_{ij} \geq 0
\end{align*}
\]

No objective function! We only look for a *feasible* solution
More LP-speak

Consider a solution $x$ of a given LP. $x$ is an extreme point solution if it cannot be expressed as a convex combination $\alpha x' + (1 - \alpha) x''$ with $\alpha \in (0, 1)$ of two other feasible solutions $x'$ and $x''$.

**Theorem 6.** $x \in \mathbb{R}^r$ is an extreme point solution iff it corresponds to setting $r$ linearly independent constraints to equality.

*Proof.* not here.
\( S_T := \{ (i, j) : p_{ij} \leq T \} \)

**LP(\(T)\):**
\[\forall j \in J : \sum_{\{i: (i,j) \in S_T\}} x_{ij} = 1\]
\[\forall i \in M : \sum_{\{j: (i,j) \in S_T\}} x_{ij} p_{ij} \leq T\]
\[\forall (i, j) \in S_T : x_{ij} \geq 0\]

**Lemma 7.** An extreme point solution of \(LP(T)\) has at most \(n + m\) nonzero variables.

**Proof.** \( r = |S_T| \) variables
\(n + m\) constraints (except \(\geq 0\))
\(T^{hm6} \implies \geq r - (n + m)\) of the \(\geq 0\) constraints are tight. \(\square\)
Lemma 7. An extreme point solution of $LP(T)$ has at most $n + m$ nonzero variables.

Corollary 8. An extreme point solution of $LP(T)$ sets $\geq n - m$ jobs integrally.

Proof.

$a$ integrally set jobs $\rightsquigarrow a$ nonzero entries in $x$

$n - a$ fractionally set jobs $\rightsquigarrow \geq 2(n - a)$ nonzero entries in $x$

Lemma 7 $\rightsquigarrow$

$2(n - a) + a \leq n + m$

$\Leftrightarrow a \geq n - m$
One Reason why LP Relaxation is Useful

Theorem 6 often implies that only few variables need to be rounded to obtain an solution of the ILP.

... this does not mean rounding the remaining ones is easy.
The Algorithm: Top Level

\[ \alpha := \text{makespan one gets by assigning each job to the fastest machine for it} \]
\[ \alpha \text{ is an upper bound for the optimal makespan} \]
The Algorithm: Top Level

\( \alpha := \) makespan one gets by assigning each job to the fastest machine for it
\( \alpha \) is an upper bound for the optimal makespan
Use binary search in the range \([\alpha/m, \alpha]\)
to find the smallest \(T\) such that \(LP(T)\) has a feasible solution \(x\)
The Algorithm: Top Level

$\alpha := \text{makespan one gets by assigning each job to the fastest machine for it}$

$\alpha$ is an upper bound for the optimal makespan

Use binary search in the range $[\alpha/m, \alpha]$ to find the smallest $T$ such that $LP(T)$ has a feasible solution

For this $T$, find an extremal point solution $x$
The Algorithm: Top Level

\( \alpha := \) makespan one gets by assigning each job to the fastest machine for it 
\( \alpha \) is an upper bound for the optimal makespan
Use binary search in the range \( [\alpha/m, \alpha] \) to find the smallest \( T \) such that \( LP(T) \) has a feasible solution
For this \( T \), find an extremal point solution \( x \) 
assign integrally set jobs in \( x \)
The Algorithm: Top Level

\[ \alpha := \text{makespan one gets by assigning each job to the fastest machine for it} \]
\[ \alpha \text{ is an upper bound for the optimal makespan} \]
Use binary search in the range \( \lfloor \alpha / m \rfloor, \alpha \)

to find the smallest \( T \) such that \( LP(T) \) has a feasible solution
For this \( T \), find an extremal point solution \( x \)
assign integrally set jobs in \( x \)
deal with the fractionally set jobs // Rounding
### Example

<table>
<thead>
<tr>
<th>$p_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Four machines, five jobs

For each job, the best machine for it is marked in blue.
### Example

<table>
<thead>
<tr>
<th>$p_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Each job on fastest machine:

<table>
<thead>
<tr>
<th></th>
<th>6</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\alpha$</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>3</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Initial guess for the makespan is 6

Using binary search, we find smallest makespan in the range $[6/4, 6]$ that can be achieved using a fractional assignment.
Example

<table>
<thead>
<tr>
<th>$p_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

Each job on fastest machine:

<table>
<thead>
<tr>
<th></th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>=</td>
<td>$\alpha$</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>5</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Solution of LP(3):

<table>
<thead>
<tr>
<th>$x_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>4</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>
Dealing with Fractionally Set Jobs

Consider the bipartite graph

\[ H := (J' \cup M', E') \] where

\[ J' := \{ j \in J : \exists i : 0 < x_{ij} < 1 \} \]

\[ M' := \{ i \in M : \exists j : 0 < x_{ij} < 1 \} \]

\[ E' := \{ \{ i, j \} : x_{ij} \neq 0, i \in M', j \in J' \} \]

Idea: Find a perfect matching in \( H \)
assign jobs according to that matching
Matching

A set of edges $M$ that do not have any nodes in common, i.e., $(V, M)$ has maximum degree one.

Perfect Matching

A matching of size $|V|/2$, i.e., all nodes are matched.
**Lemma 9.** \( H \) is a *pseudo forest*, i.e., each connected component \( H_C = (V_C, E_C) \) has \( |E_C| \leq |V_C| \) (a tree plus, possibly, one edge)

**Proof.** It suffices to show this for the larger graph \( G := (J \cup M, E) \) where

\[
E := \{ \{i, j\} : x_{ij} \neq 0, i \in M, j \in J \}
\]

Consider a connected component \( H_C \) of \( G \).

restrict \( x \) and \( LP(T) \) to \( H_C: x_C, LP_C(T) \)

\( x_C \) is extreme point solution of \( LP_C(T) \)

(Otherwise, \( x \) itself could not be extreme point solution)

Lemma 7 \( \Rightarrow \) \( LP_C(T) \) has \( \leq |V_C| \) nonzero vars.,

i.e., \( H_C \) has \( \leq |V_C| \) edges.  \( \square \)
### Example

<table>
<thead>
<tr>
<th>$p_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>2</td>
<td>2</td>
<td>4</td>
<td>2</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>3</td>
<td>3</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>3</td>
<td>2</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>4</td>
<td>4</td>
<td>4</td>
<td>2</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>$x_{ij}$</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
<tr>
<td>2</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>0</td>
<td>0</td>
<td>$\frac{1}{2}$</td>
<td>$\frac{1}{2}$</td>
<td>0</td>
</tr>
</tbody>
</table>

| 4        | $\frac{1}{2}$ | 0 | 0 | 0 | 1 |

$T^* = 3$
Lemma 10. \(H\) has a perfect matching

Proof. We give an algorithm:

\[M := \emptyset\]

**invariant** \(H\) is a bipartite pseudo forest

**invariant** all degree one nodes are machines

**while** \(\exists i \in M'\) with degree one **do**

\[e = \{i, j\} := \text{the sole edge incident to } i\]

\[M := M \cup \{e\}\]

remove \(i, j\) and incident edges

**assert** \(H\) is a collection of disjoint even cycles

**foreach** cycle \(C \in H\) **do**

match alternating edges in \(C\)
Theorem 11. The algorithm achieves an approximation guarantee of factor 2 for scheduling unrelated parallel machines.

Proof. Consider solution \( x \) of LPT(\( T^* \)) makespan due to jobs set integrally in \( x \) is \( \leq T^* \leq \text{opt} \).

In addition, each machine \( i \) receives \( \leq 1 \) job \( j \) from the matching \( M \subseteq H \).

\( p_{i,j} \leq T^* \leq \text{opt} \) since otherwise \( \{i, j\} \notin H \). \qed