Graph Algorithms

I. Shortest paths

- $D = (V, A)$ directed graph, $s, t \in V$.
- A walk is a sequence $P = (v_0, a_1, v_1, ..., a_k, v_k), k \geq 0$, where $a_i$ is an arc from $v_{i-1}$ to $v_i$, for $i = 1, ..., k$.
- $P$ is a path, if $v_0, ..., v_k$ are all different.
- If $s = v_0$ and $t = v_k$, $P$ is a s-t walk resp. s-t path of length $k$ (i.e., each arc has length 1).
- The distance from $s$ to $t$ is the minimum length of any s-t path (and $+\infty$ if no s-t path exists).

Shortest paths with unit lengths

Algorithm (Breadth-first search)

Initialization: $V_0 = \{s\}$

Iteration: $V_{i+1} = \{v \in V \setminus (V_0 \cup V_1 \cup \cdots \cup V_i) \mid (u, v) \in A, \text{ for some } u \in V_i\}$, until $V_{i+1} = \emptyset$.

Running time: $O(|A|)$

- $V_i$ is the set of nodes with distance $i$ from $s$.
- The algorithm computes shortest paths from $s$ to all reachable nodes.
- Can be described by a directed tree $T = (V', A')$ with root $s$ such that each $u$-$v$ path in $T$ is a shortest s-t path in $D$.

Shortest paths with non-negative lengths

- Length function $l : A \rightarrow \mathbb{Q}_+ = \{x \in \mathbb{Q} \mid x \geq 0\}$
- For a walk $P = (v_0, a_1, v_1, ..., a_k, v_k)$ define $l(P) = \sum_{i=1}^{k} l(a_i)$.

Algorithm (Dijkstra 1959)

Initialization: $U = V, f(s) = 0, f(v) = \infty$, for $v \in V \setminus \{s\}$

Iteration: Find $u \in U$ with $f(u) = \min\{f(v) \mid v \in U\}$.
For all $a = (u, v) \in A$ with $f(v) > f(u) + l(a)$ let $f(v) = f(u) + l(a)$.
Let $U \leftarrow U \setminus \{u\}$, until $U = \emptyset$.

Upon termination, $f(v)$ gives the length of a shortest path from $s$ to $v$.

Running time: $O(|V|^2)$ (can be improved to $O(|A| + |V| \log |V|)$.)

Application: Longest common subsequence

- Sequences $a = a_1, ..., a_m$ and $b = b_1, ..., b_n$
- Find the longest common subsequence of $a$ and $b$ (obtained by removing symbols in $a$ or $b$).
Modeling as a shortest path problem

- Grid graph with nodes \((i, j), 0 \leq i \leq m, 0 \leq j \leq n\).
- Horizontal and vertical arcs of length 1.
- Diagonal arcs \(((i - 1, j - 1), (i, j))\) of length 0, if \(a_i = b_j\).

The diagonal arcs on a shortest path from \((0, 0)\) to \((m, n)\) define a longest common subsequence.

Circuits of negative length

- Consider arbitrary length functions \(l : A \rightarrow \mathbb{Q}\).
- A directed circuit is a walk \(P = (v_0, a_1, v_1, \ldots, a_k, v_k)\) with \(k \geq 1\) and \(v_0 = v_k\) such that \(v_1, \ldots, v_k\) and \(a_1, \ldots, a_k\) are all different.
- If \(D = (V, A)\) contains a directed circuit of negative length, there exist \(s\)-\(t\) walks of arbitrary small negative length.

**Proposition**

Let \(D = (V, A)\) be a directed graph without circuits of negative length. For any \(s, t \in V\) for which there exists at least one \(s\)-\(t\) walk, there exists a shortest \(s\)-\(t\) walk, which is a path.

Shortest paths with arbitrary lengths

\(D = (V, A), n = |V|, l : A \rightarrow \mathbb{Q}\).

**Algorithm** (Bellman-Ford 1956/58)

Compute \(f_0, \ldots, f_n : V \rightarrow \mathbb{R} \cup \{\infty\}\) in the following way:

**Initialization:** \(f_0(s) = 0, f_0(v) = \infty\), for \(v \in V \setminus \{s\}\)

**Iteration:** For \(k = 1, \ldots, n\) and all \(v \in V\):

\[
f_k(v) = \min \{f_{k-1}(v), \min_{(u, v) \in A} (f_{k-1}(u) + l(u, v))\}
\]

**Running time:** \(O(|V||A|)\)

**Properties**

- For each \(k = 0, \ldots, n\) and each \(v \in V\):

\[
f_k(v) = \min \{l(P) \mid P \text{ is an } s\text{-}v \text{ walk traversing at most } k \text{ arcs} \}
\]

(by induction)

- If \(D\) contains no circuits of negative length, \(f_{n-1}(v)\) is the length of a shortest path from \(s\) to \(v\).

**Finding an explicit shortest path**

- When computing \(f_0, \ldots, f_n\) determine a predecessor function \(p : V \rightarrow V\) by setting \(p(v) = u\) whenever \(f_{k+1}(v) = f_k(u) + l(u, v)\).

- At termination, \(v, p(v), p(p(v)), \ldots, s\) gives the reverse of a shortest \(s\)-\(v\) path.
Theorem
Given \( D = (V, A), s, t \in V \) and \( l : A \to \mathbb{Q} \) such that \( D \) contains no circuit of negative length, a shortest \( s-t \) path can be found in time \( O(|V||A|) \).

Remark
\( D \) contains a circuit of negative length reachable from \( s \) if and only if \( l_n(v) \neq l_{n-1}(v) \), for some \( v \in V \).

\[ \text{NP-completeness} \]

For directed graphs containing circuits of negative length, the problem becomes NP-complete:

**Theorem**
The decision problem

\[ \text{Input: Directed graph } D = (V, A), s, t \in V, l : A \to \mathbb{Z}, L \in \mathbb{Z} \]

\[ \text{Question: Does there exist an } s-t \text{ path } P \text{ with } l(P) \leq L? \]

is NP-complete.

**Corollary**
The shortest path problem with arbitrary lengths is NP-complete.
The longest path problem with non-negative lengths is NP-complete.

\[ \text{Application: Knapsack problem} \]

- Knapsack, volume 8, 5 articles

<table>
<thead>
<tr>
<th>Article ( i )</th>
<th>Volume ( a_i )</th>
<th>Value ( c_i )</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>5</td>
<td>4</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>7</td>
</tr>
<tr>
<td>3</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>5</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>4</td>
</tr>
</tbody>
</table>

- Objective: Select articles fitting into the knapsack and maximizing the total value.

\[ \text{Possible models} \]

- **Linear 0-1 model**

\[ \max \{ 4x_1 + 7x_2 + 3x_3 + 5x_4 + 4x_5 \mid 5x_1 + 3x_2 + 2x_3 + 2x_4 + x_5 \leq 8, x_1, \ldots, x_5 \in \{0,1\} \} \]

- **Shortest path model**

  - Directed graph with nodes \((i, x), 0 \leq i \leq 6, 0 \leq x \leq 8\).
  - Arcs from \((i - 1, x)\) to \((i, x)\) resp. \((i, x + a)\) of length 0 resp. \(-c_i\), for \(0 \leq i \leq 5\).
  - Arcs from \((5, x)\) to \((6, 8)\) of length 0, for \(0 \leq x \leq 6\).
  - A shortest path from \((0, 0)\) to \((6, 8)\) gives an optimal solution.

  \[ \text{pseudo-polynomial algorithm} \]
II. Network flows

- **Network**
  - Directed graph $G = (V, E)$
  - *Source* $s \in V$, *sink* $t \in V$
  - *Edge capacities* $\text{cap} : E \to \mathbb{R}_+ = \{ x \in \mathbb{R} \mid x \geq 0 \}$

- **Flow** $f : E \to \mathbb{R}_+$ satisfying
  1. Flow conservation constraints
     \[ \sum_{e : \text{target}(e) = v} f(e) = \sum_{e : \text{source}(e) = v} f(e), \text{ for all } v \in V \setminus \{s, t\} \]
  2. Capacity constraints
     \[ 0 \leq f(e) \leq \text{cap}(e), \text{ for all } e \in E \]

**Maximum flow problem**

- **Excess** at node $v$: $\text{excess}(v) = \sum_{e : \text{target}(e) = v} f(e) - \sum_{e : \text{source}(e) = v} f(e)$
- If $f$ is a flow, then $\text{excess}(v) = 0$, for all $v \in V \setminus \{s, t\}$.
- **Value** of a flow: $\text{val}(f) \overset{\text{def}}{=} \text{excess}(t)$
- **Maximum flow problem**: \( \max \{ \text{val}(f) \mid f \text{ is a flow in } G \} \)
- Can be seen as a linear programming problem.

**Maximum flow problem** (a)

**Lemma**
If $f$ is a flow, then $\text{excess}(t) = -\text{excess}(s)$.

**Proof:** We have
\[ \text{excess}(s) + \text{excess}(t) = \sum_{v \in V} \text{excess}(v) = 0. \]

- First “$=$”:\ $\text{excess}(v) = 0$, for $v \in V \setminus \{s, t\}$
- Second “$=$”:\ For any edge $e = (v, w)$, the flow through $e$ appears twice in the sum, positively in $\text{excess}(w)$ and negatively in $\text{excess}(v)$.

**Cuts**

- A cut is a partition $(S, T)$ of $V$, i.e., $T = V \setminus S$.
- $(S, T)$ is an $(s, t)$-cut if $s \in S$ and $t \in T$.
- **Capacity** of the cut $(S, T)$
  \[ \text{cap}(S, T) = \sum_{E \cap (S \times T)} \text{cap}(e) \]
- A cut is saturated by $f$ if $f(e) = \text{cap}(e)$, for all $e \in E \cap (S \times T)$, and $f(e) = 0$, for all $e \in E \cap (T \times S)$. 
**Lemma**

If $f$ is a flow and $(S, T)$ an $(s,t)$-cut, then

$$\text{val}(f) = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) \leq \text{cap}(S, T).$$

If $S$ is saturated by $f$, then $\text{val}(f) = \text{cap}(S, T)$.

**Proof:** We have

$$\text{val}(f) = -\text{excess}(s) = -\sum_{u \in S} \text{excess}(u) = \sum_{e \in E \cap (S \times T)} f(e) - \sum_{e \in E \cap (T \times S)} f(e) \leq \sum_{e \in E \cap (S \times T)} \text{cap}(e) = \text{cap}(S).$$

For a saturated cut, the inequality is an equality.

**Remarks**

- A saturated cut proves the optimality of a flow.
- To show: for every maximal flow there is a saturated cut proving its optimality.

**Residual network**

The residual network $G_f$ for a flow $f$ in $G = (V, E)$ indicates the capacity unused by $f$. It is defined as follows:

- $G_f$ has the same node set as $G$.
- For every edge $e = (v, w)$ in $G$, there are up to two edges $e'$ and $e''$ in $G_f$:
  1. if $f(e) < \text{cap}(e)$, there is an edge $e' = (v, w)$ in $G_f$ with residual capacity $r(e') = \text{cap}(e) - f(e)$.
  2. if $f(e) > 0$, there is an edge $e'' = (w, v)$ in $G_f$ with residual capacity $r(e'') = f(e)$.

**Theorem**

Let $f$ be an $(s,t)$-flow, let $G_f$ be the residual network w.r.t. $f$, and let $S$ be the set of all nodes reachable from $s$ in $G_f$. 

1. If \( t \in S \), then \( f \) is not maximum.

2. If \( t \not\in S \), then \( S \) is a saturated cut and \( f \) is maximum.

**Proof**

If \( t \) is reachable from \( s \) in \( G_f \), then \( f \) is not maximal.

- Let \( P \) be a (simple) path from \( s \) to \( t \) in \( G_f \).
- Let \( \delta \) be the minimum residual capacity of an edge in \( P \). By definition, \( r(e) > 0 \), for all edges \( e \) in \( G_f \). Therefore, \( \delta > 0 \).
- Construct a flow \( f' \) of value \( \text{val}(f) + \delta \):

\[
f'(e) = \begin{cases} 
  f(e) + \delta, & \text{if } e' \in P \\
  f(e) - \delta, & \text{if } e'' \in P \\
  f(e), & \text{if neither } e' \text{ nor } e'' \text{ belongs to } P.
\end{cases}
\]

- \( f' \) is a flow and \( \text{val}(f') = \text{val}(f) + \delta \).

**Example**

If \( t \) is not reachable from \( s \) in \( G_f \), then \( f \) is maximal.

- Let \( S \) be the set of nodes reachable from \( s \) in \( G_f \), and let \( T = V \setminus S \).
- There is no edge \( (v, w) \) in \( G_f \) with \( v \in S \) and \( w \in T \).
- Hence
  - \( f(e) = \text{cap}(e) \), for any \( e \in E \cap (S \times T) \), and
  - \( f(e) = 0 \), for any \( e \in E \cap (T \times S) \).
- Thus \( S \) is saturated and, by the Lemma, \( f \) is maximal.

**Ford-Fulkerson Algorithm**

1. Start with the zero flow, i.e., \( f(e) = 0 \), for all \( e \in E \).
2. Construct the residual network \( G_f \).
3. Check whether \( t \) is reachable from \( s \) in \( G_f \).
• if not, stop.
• if yes, increase the flow along an augmenting path, and iterate.

Analysis

• Let $|V| = n$ and $|E| = m$.
• Each iteration takes time $O(n + m)$.
• If capacities are arbitrary reals, the algorithm may run forever.

Integer capacities

• Suppose capacities are integers, bounded by $C$.
• $v^* \triangleq$ value of maximum flow $\leq Cn$.
• All flows constructed are integral (proof by induction).
• Every augmentation increases flow value by at least 1.
• Running time $O((n + m)v^*) \rightarrow$ pseudo-polynomial algorithm

Edmonds-Karp Algorithm

• Compute a shortest augmenting path, i.e. with a minimum number of arcs.
• Apply breadth-first search (or Dijkstra's algorithm).
• Number of iterations is bound by $nm$, leads to an $O(nm^2)$ maximum flow algorithm.
• Works also for irrational capacities.

Max-Flow Min-Cut Theorem

Theorem
For a network $(V, E, s, t)$ with capacities $\text{cap} : E \to \mathbb{R}_+$ the maximum value of a flow is equal to the minimum capacity of an $(s, t)$-cut:

$$\max\{\text{val}(f) \mid f \text{ is a flow}\} = \min\{\text{cap}(S, T) \mid (S, T) \text{ is an } (s, t)-\text{cut}\}$$

Corollary
For integer capacities $\text{cap} : E \to \mathbb{Z}_+$, there exists an integer-valued maximum flow $f : E \to \mathbb{Z}_+$. 

III. Matching

- $G = (V, E)$ undirected graph
- **Matching**: Subset of edges $M \subseteq E$, no two of which share an endpoint.
- **Maximum matching**: Matching of maximum cardinality
- **Perfect matching**: Every vertex in $V$ is matched.

**Augmenting paths**

- Let $M$ be a matching in $G = (V, E)$.
- A path $P = (v_0, v_1, ..., v_t)$ in $G$ is called $M$-augmenting if:
  - $t$ is odd,
  - $v_1 v_2, v_3 v_4, v_{t-2} v_{t-1} \in M$,
  - $v_0, v_t \notin \bigcup M = \bigcup_{e \in M} e$.
- If $P$ is an $M$-augmenting path and $E(P)$ the edge set of $P$, then

$$M' = M \triangle E(P) = (M \setminus E(P)) \cup (E(P) \setminus M)$$

is a matching in $G$ of size $|M'| = |M| + 1$.

**Berge’s Theorem**

**Theorem** (Berge’57)
Let $M$ be a matching in the graph $G = (V, E)$. Then either $M$ is a maximum cardinality matching or there exists an $M$-augmenting path.

**Generic Matching Algorithm**

*Initialization*: $M \leftarrow \emptyset$

*Iteration*: If there exists an $M$-augmenting path $P$, replace $M \leftarrow M \triangle E(P)$.

→ how can one find an $M$-augmenting path?

- Difficult in general → Edmonds’ matching algorithm (Edmonds’65)
- Easy for bipartite graphs

**Bipartite graphs**

A graph $G = (V, E)$ is **bipartite** if there exist $A, B \subseteq V$ with $A \cup B = V, A \cap B = \emptyset$ and each edge in $E$ has one end in $A$ and one end in $B$.

**Proposition**
A graph $G = (V, E)$ is bipartite if and only if each circuit of $G$ has even length.

**Bipartite matching**
Matching augmenting algorithm for bipartite graphs

**Input:** Bipartite graph $G = (A \cup B, E)$ with matching $M$.

**Output:** Matching $M'$ with $|M'| > |M|$ or proof that no such matching exists.

**Description:** Construct a directed graph $D_M$ with the same node set as $G$.

- For each edge $e = \{a, b\}$ in $G$ with $a \in A, b \in B$:
  - if $e \in M$, there is the arc $(b, a)$ in $D_M$.
  - if $e \not\in M$, there is the arc $(a, b)$ in $D_M$.

Let $A_M = A \setminus \bigcup M$ and $B_M = B \setminus \bigcup M$.

$M$-augmenting paths in $G$ correspond to directed paths in $D_M$ starting in $A_M$ and ending in $B_M$.

**Theorem**

A maximum-cardinality matching in a bipartite graph $G = (V, E)$ can be found in time $O(|V||E|)$.

**Bipartite matching as a maximum flow problem**

- Add a source $s$ and edges $(s, a)$ for $a \in A$, with capacity 1.
- Add a sink $t$ and edges $(b, t)$ for $b \in B$, with capacity 1.
- Direct edges in $G$ from $A$ to $B$, with capacity 1.

- Integral flows $f$ correspond to matchings $M$, with $\text{val}(f) = |M|$.
- Ford-Fulkerson takes time $O(nm)$, since $\nu^* \leq n$.
- Can be improved to $O(\sqrt{nm})$.

**Marriage theorem**

**Theorem (Hall)**

A bipartite graph $G = (A \cup B, E)$, with $|A| = |B| = n$, has a perfect matching if and only if for all $B' \subseteq B$, $|B'| \leq |N(B')|$, where $N(B')$ is the set of all neighbors of nodes in $B'$. 
Proof

- Let \((S, T)\) be an \((s, t)\)-cut in the corresponding network.
- Let \(A_S = A \cap S, A_T = A \cap T, B_S = B \cap S, B_T = B \cap T\).

\[
\text{cap}(S, T) = \sum_{e \in E \cap S \times T} \text{cap}(e) = |A_T| + |B_S| + |N(B_T) \cap A_S| \\
\geq |N(B_T) \cap A_T| + |N(B_T) \cap A_S| + |B_S| \\
= |N(B_T)| + |B_S| \\
\geq |B_T| + |B_S| = |B| = n
\]

- By the max-flow min-cut theorem, the maximum flow is at least \(n\).

**Konig's theorem**

- \(G = (V, E)\) undirected graph
- \(C \subseteq V\) is a **vertex covering** if every edge of \(G\) has at least one end in \(C\).
- **Lemma:** For any matching \(M\) and any vertex covering \(C\), we have \(|M| \leq |C|\).
- **Theorem (Konig)** For a bipartite graph \(G\),

\[
\max\{|M| : M \text{ a matching }\} = \min\{|C| : C \text{ a vertex covering }\}.
\]

**Network connectivity**

- \(G = (V, E)\) directed graph, \(s, t \in V, s \neq t\) non-adjacent.
- **Theorem (Menger)** The maximum number of **arc-disjoint** paths from \(s\) to \(t\) equals the minimum number of arcs whose removal disconnects all paths from \(s\) to \(t\).
- **Theorem (Menger)** The maximum number of **node-disjoint** paths from \(s\) to \(t\) equals the minimum number of nodes (different from \(s\) and \(t\)) whose removal disconnects all paths from \(s\) to \(t\).
Duality in linear programming

• Primal problem

\[ z_P = \max \{ c^T x \mid Ax \leq b, x \in \mathbb{R}^n \} \]  
(P)

• Dual problem

\[ w_D = \min \{ b^T u \mid A^T u = c, u \geq 0 \} \]  
(D)

General form

<table>
<thead>
<tr>
<th>(P)</th>
<th>(D)</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \min ) ( c^T x )</td>
<td>( \max ) ( u^T b )</td>
</tr>
<tr>
<td>w.r.t. ( A_i x \geq b_i, \quad i \in M_1 )</td>
<td>w.r.t. ( u_i \geq 0, \quad i \in M_1 )</td>
</tr>
<tr>
<td>( A_i x \leq b_i, \quad i \in M_2 )</td>
<td>( u_i \leq 0, \quad i \in M_2 )</td>
</tr>
<tr>
<td>( A_i x = b_i, \quad i \in M_3 )</td>
<td>( u_i \text{ free,} \quad i \in M_3 )</td>
</tr>
<tr>
<td>( x_j \geq 0, \quad j \in N_1 )</td>
<td>( (A_{ij})^T u \leq c_j, \quad j \in N_1 )</td>
</tr>
<tr>
<td>( x_j \leq 0, \quad j \in N_2 )</td>
<td>( (A_{ij})^T u \geq c_j, \quad j \in N_2 )</td>
</tr>
<tr>
<td>( x_j \text{ free,} \quad j \in N_3 )</td>
<td>( (A_{ij})^T u = c_j, \quad j \in N_3 )</td>
</tr>
</tbody>
</table>

Duality theorems

• **Weak duality** If \( x^* \) is primal and \( u^* \) is dual feasible, then

\[ c^T x^* \leq z_P \leq w_D \leq b^T u^*. \]

• **Strong duality** If both (P) and (D) have a finite optimum, then \( z_P = w_D \).

• **Only four possibilities**

1. \( z_P \) and \( w_D \) are both finite and equal.
2. \( z_P = +\infty \) and (D) is infeasible.
3. \( w_D = -\infty \) and (P) is infeasible.
4. (P) and (D) are both infeasible.

Maximum flow and duality
• Primal problem

$$\max \sum_{e: \text{source}(e)=s} x_e - \sum_{e: \text{target}(e)=s} x_e$$

s.t. $$\sum_{e: \text{target}(e)=v} x_e - \sum_{e: \text{source}(e)=v} x_e = 0, \quad \forall v \in V \setminus \{s,t\}$$

$$0 \leq x_e \leq c_e, \quad \forall e \in E$$

• Dual problem

$$\min \sum_{e \in E} c_e y_e$$

s.t. $$z_w - z_v + y_e \geq 0, \quad \forall e = (v,w) \in E$$

$$z_s = 1, z_t = 0$$

$$y_e \geq 0, \quad \forall e \in E$$

Maximum flow and duality

• Let $$(y^*, z^*)$$ be an optimal solution of the dual.

• Define $S = \{v \in V \mid z^*_v > 0\}$ and $T = V \setminus S$.

• $$(S, T)$$ is a minimum cut.

• Max-flow min-cut theorem is a special case of linear programming duality.

Total unimodularity

• A matrix $A$ is totally unimodular if each subdeterminant of $A$ is 0, +1 or −1.

• Theorem (Hoffman and Kruskal) $A \in \mathbb{Z}^{m \times n}$ is totally unimodular iff the polyhedron $P = \{x \in \mathbb{R}^n \mid Ax \leq b, x \geq 0\}$ is integral, i.e., $P = \text{conv}(P \cap \mathbb{Z}^n)$, for any $b \in \mathbb{Z}^m$.

• Corollary $A \in \mathbb{Z}^{m \times n}$ is totally unimodular iff for any $b \in \mathbb{Z}^m, c \in \mathbb{Z}^n$ both optima in the LP duality equation

$$\max \{c^T x \mid Ax \leq b, x \geq 0\} = \{\min b^T u \mid A^T u \geq c, u \geq 0\}$$

are attained by integral vectors (if they are finite).

• Proposition The constraint matrix $A$ arising in a maximum flow problem is totally unimodular.

Matching and linear programming

• $G = (V, E)$ undirected graph, $M \subseteq E$ matching

• Incidence vector: $\chi^M : E \rightarrow \mathbb{R}, \chi^M(e) = \begin{cases} 1, & \text{if } e \in M, \\ 0, & \text{if } e \notin M. \end{cases}$

• Maximum matching as an integer linear program

$$\max \left\{ \sum_{e \in E} x_e \mid \sum_{e \ni v} x_e \leq 1, \forall v \in V, \; x_e \in \{0,1\}, \forall e \in E \right\}$$
• For bipartite graphs the constraint matrix is totally unimodular \( \Rightarrow \) linear program

\[
\max \left\{ \sum_{e \in E} x_e \mid \sum_{v \in V} x_e \leq 1, \forall v \in V, \ x_e \geq 0, \forall e \in E \right\}
\]

• Dual linear program

\[
\min \left\{ \sum_{v \in V} y_v \mid y_v + y_w \geq 1, \forall e = \{v, w\} \in E, \ y_v \geq 0, \forall v \in V \right\}
\]

\( \Rightarrow \) minimum vertex cover

References


